



Productive
Computing

Creating Efficiency Through Automation



Developer's Guide

March 15, 2022

Table of Contents

I. INTRODUCTION	3
II. INTEGRATION STEPS	4
1) Installation Components - Prerequisites for Plug-in Installation	4
2) Installing the Plug-in with the Installer	5
3) Installing the Plug-in with the Demo File	6
4) Troubleshooting Plug-in Installation	7
5) Registering the Plug-in.....	8
Why do I need to Register?	9
How do I hard code the registration process?.....	9
6) FileMaker 16 Plug-in Script Steps	10
7) Integration Tips and Tricks.....	12
8) Authorization - QuickBooks Online Connection and Establishing a Session	14
Authorization	14
Multi-User Session Management.....	20
9) Server-Side Deployment.....	22
Installation	22
10) Talking to QuickBooks Online.....	27
Request and Response Explained	28
Create a Request	29
Post the Request	31
Parse the Response.....	32
Send Invoice as PDF	34
Custom Fields.....	34
Minor Versions.....	35
Batch Request and Response Handling	35
Working with Attachable Files	37
Attachable Notes.....	40
Additional Fields	41
Item Hierarchy Change.....	42
Change Data Capture (CDC)	43
JSON Handling.....	44
11) QuickBooks Online Accounting API Reference	45
12) Supported and Unsupported areas using QBO.....	50
13) Entity Query Support	51
Counting Entities.....	53
14) Sample Scripts and Queries	54
15) Error Handling.....	61
16) Tips.....	63
17) Known Issues.....	64
III. CONTACT US	68

I. Introduction

Description:

The FM Books Connector Online plug-in is a powerful tool used to move data between FileMaker® Pro and Intuit's® QuickBooks Online. The Developer's Guide will explain the necessary integration steps, how the plug-in "talks" to QuickBooks Online and the concept of how to create your FileMaker scripts which can be applied to any area in QuickBooks Online. 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:

Intermediate to advanced developers or persons with knowledge of FileMaker Pro or FileMaker Pro Advanced, especially in the areas of 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) Familiarize yourself with basic accounting practices and QuickBooks Online
- 2) Use the Intuit's Accounting API Reference guide
<https://developer.intuit.com/app/developer/qbo/docs/api/accounting/most-commonly-used/account>
- 3) Visit the Help Center to review documents, FAQs, and videos
<https://help.productivecomputing.com/help/fm-books-connector-online>
 - a. Developer's Guide
 - b. Functions Guide
 - c. Migration Guide (for migrating from QuickBooks Desktop to QuickBooks Online)
- 4) Visit the product page for:
<https://www.productivecomputing.com/products/quickbooks-online-filemaker-integration/>
 - a. Downloading the current plug-in and demo version
 - b. System requirements
 - c. Version history
- 5) QuickBooks Online integration training is available for purchase through Productive Computing University
<https://www.productivecomputinguniversity.com/courses/connect-filemaker-to-quickbooks-online>

II. Integration Steps

Accessing and using the plug-in functions involve the following steps.

1) Installation Components - Prerequisites for Plug-in Installation

Installing the Microsoft Visual C++ 2013 Redistributable Package:

Included in the package is a download link.

Name of link is: "Download Microsoft Visual C++ 2013 Redistributable Package (x86)"

This link will direct you to download the Microsoft Visual C++ Redistributable Package (x86). Some systems do not have a Visual C++ 2013 Redistributable Package installed by default. However, certain programs may have added it to your machine during their installation process.

If the plug-in fails to be recognized by FileMaker after installation (i.e. does not show up in the Edit > Preferences > Plug-ins section), then please install the included redistributable package.

Machines running 64-bit versions of Windows need to install the 64-bit ("x64") version of the redistributable package, which is also available from Microsoft.

Please note: For older versions, use the 2008 redistributable package.

Installing the Microsoft .NET 4.5 Framework

A copy of the Microsoft .NET 4.5 Framework can be downloaded from the following link:

<http://www.microsoft.com/net/downloads>

Accessing and using the plug-in functions involve the following steps.

2) Installing the Plug-in with the Installer

We are happy to announce the introduction of installers for our latest plug-in releases. These installers will not only install the FileMaker plug-in, but will also install the third-party software needed for the plug-in to function, the demo file, and additional resources you may need. We recommend using the installers to ensure that all components necessary for the plug-in to function are properly installed.

Windows Installer:

- 1) Run the "setup.exe" file that is included in the bundle downloaded from our website.
- 2) If prompted, install the Visual C++ 2013 Runtime Libraries.
- 3) If you are currently running FileMaker, please close FileMaker so that the plug-in will be installed and initialized correctly.
- 4) Accept the License.
- 5) Select the location to install the plug-in*.
- 6) Confirm the installation.
- 7) If prompted by Windows security, allow the installer to run.
- 8) Your installation is complete!

*In order for FileMaker to properly recognize the plug-in, we suggest you do not change this default location. The FileMaker plug-in needs to be installed in a valid Extensions folder recognized by the application. By default, the plug-in will be installed to the base FileMaker/Extensions folder and will be available across multiple versions of FileMaker. However, if you wish to install the plug-in at a version specific location like "FileMaker Pro Advanced 15/Extensions", you may browse to the folder location to do so.

Mac Installer:

- 1) Run the "Install FM Books Connector Online.dmg" file that is included in the bundle you downloaded from our website.
- 2) Run the "Install FM Books Connector Online" application that is in the installer.
- 3) If you are currently running FileMaker, please close FileMaker so that the plug-in will be installed and initialized correctly.
- 4) Continue through the Licensing Information, Destination Select, and Installation Type screens.
- 5) Select "Install" if you wish to install the FileMaker Plug-in and Demo File.
- 6) If prompted, enter your machine credentials to approve the installation.
- 7) Your installation is complete!

Note: Both installers come with an application (.exe or .dmg) to install the plug-in and an Extras folder. In the Extras folder, you will find additional resources such as License, README, FileMaker Demo File, and plug-ins.

3) Installing the Plug-in with the Demo File

Alternatively, you may install the plug-in using the Demo File provided in the Extras folder that came with the bundle from our website. Note: If you have not already, you will need to run the Visual C++ installers that are in the Extras folder in order for the plug-in to function properly.

FileMaker 12 or later:

- 1) Open the FileMaker demo file available in the plug-in bundle (www.productivecomputing.com).
- 2) Select the "Install" button.

4) Troubleshooting Plug-in Installation

When installing the plug-in using the "Install Plug-in" script step, there are certain situations that may cause a 1550 or 1551 error to arise. If such a situation occurs, please refer to the troubleshooting steps involving the most common problems that may cause those errors.

1) Invalid Bitness of FileMaker

- a. In some cases, FileMaker Pro may be attempting to install a plug-in with a different bitness than the FileMaker Pro application. This is most common with Windows plug-ins. The general rule is that the plug-in and FileMaker Pro must be the same bitness.
- b. To resolve this, ensure that the container field holding the plug-in contains the correct bitness of the plug-in. You can verify the plug-in's bitness by checking the file extension: if the extension is .fmx, the plug-in is a 32-bit plug-in; if the extension is .fmx64, the plug-in is a 64-bit plug-in. You can verify the bitness of FileMaker Pro itself by viewing the "About FileMaker Pro" menu option in the Help menu, and clicking the "Info" button to see more information; bitness is found under "Architecture".

2) Missing Dependencies

- a. Every plug-in has dependencies, which are system files present in the machine's operating system that the plug-in requires in order to function. If a plug-in is "installed" into an Extensions folder, but the plug-in does not load or is not visible in the Preferences > Plug-in panel in FileMaker Pro's preferences, it's likely that there are files missing.
- b. To ensure that the appropriate dependencies are installed, please verify that the Visual Studio 2013 C++ Redistributable Package is installed. This can be located by opening Control Panel and checking the Installed Programs list (usually found under "Add/Remove Programs"). Older plug-ins may require the Visual C++ 2008 redistributable package, instead of the 2013 version.
- c. Some plug-ins also have a .NET Framework component that is also required. All such plug-ins of ours will require the .NET Framework 3.5, which can be downloaded from the following link: <https://www.microsoft.com/en-us/download/details.aspx?id=21>

3) Duplicate Plug-in Files

- a. When installing plug-ins, it is possible to have the plug-in located in different folders that are considered "valid" when FileMaker Pro attempts to load plug-ins for use. There is a possibility that having multiple versions of the same plug-in in place in these folders could cause FileMaker Pro to fail to load a newly-installed plug-in during the installation process.
- b. To resolve this, navigate to the different folders listed in the earlier installation steps and ensure that the plug-in is not present there by deleting the plug-in file(s). Once complete, restart FileMaker and attempt the installation again. If you installed the plug-in using a plug-in installer file, if on Windows, run the installer again and choose the "Uninstall" option, or if on Mac, run the "uninstall.tool" file to uninstall the plug-in.

If the three troubleshooting steps above do not resolve the issue, please feel free to reach out to our support team for further assistance.

When installing the plug-in using the "Install Plug-in" script step, there are certain situations that may cause a 1550 or 1551 error to arise. If such a situation occurs, please refer to the troubleshooting steps involving the most common problems that may cause those errors.

5) Registering the Plug-in

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

- 1) Confirm that you have access to the internet and open our FileMaker demo file, which can be found in the "FileMaker Demo File" folder in your original download.
- 2) If you are registering the plug-in in Demo mode, then simply click the "Register" button and do not change any of the fields. Your plug-in should now be running in "DEMO" mode. The mode is always noted on the Setup tab of the FileMaker demo.

FM Books Connector Online

Setup Pull Push Other

Step 1: Install the plug-in

Manually Install:

Plug-in Version: FM Books Connector Online 3.0.0.7
Operating Mode: DEMO

Successful Installation

- 1) Read the Developer's Guide
- 2) Read the Function List
- 3) Familiarize yourself with the interface
- 4) Review the FileMaker demo
- 5) Visit the Video Tutorials
- 6) Refer to the Installation Instructions
- 7) Re-read the Developer's Guide

Step 2: Register the plug-in

License ID:

Result:

Need Help Getting Started?
Visit our [Help Center](#) documentation, support page, or view video tutorials.

Step 3: Initialize QuickBooks Connection

- 3) If you are registering a licensed copy, then simply enter your license number in the "LicenseID" field and select the "Register" button. Ensure you have removed 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 always noted on the Setup tab of the FileMaker demo, or by calling the PCQO_GetOperatingMode function.

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-in 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 PCQO_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 PCQO_GetOperatingMode and PCQO_Version which can assist you when developing an installation and registration process in your FileMaker solution.

How do I hard code the registration process?

You can hard code the registration process inside a simple "Plug-in Checker" script. The "Plug-in Checker" script should be called at the beginning of any script using a plug-in function and uses the PCQO_Register, PCQO_GetOperatingMode and PCQO_Version functions. This eliminates the need to manually register each machine and ensures that the plug-in is installed and properly registered. Below are the basic steps to create a "Plug-in Checker" script.

```
If [ PCQO_Version( "short" ) = "" or PCQO_Version( "short" ) = "?" ]
Show Custom Dialog [ Title: "Warning"; Message: "Plug-in not installed."; Buttons: "OK"
]
If [ PCQO_GetOperatingMode ≠ "LIVE" ]
Set Field [Main::gRegResult; PCQO_Register( "licensing.productivecomputing.com" ; "80" ;
"/PCIReg/pcireg.php" ; "your license ID" )
If [ Main::gRegResult ≠ 0 ]
Show Custom Dialog [ Title: "Registration Error"; Message: "Plug-in Registration
Failed"; Buttons: "OK" ]
```

6) FileMaker 16 Plug-in Script Steps

Newly introduced in FileMaker Pro 16, all plug-ins have been updated to allow a developer to specify plug-in functions as script steps instead of as calculation results. The plug-in script steps function identically to calling a plug-in within a calculation dialog.

In this example, we use the FM Books Connector plug-in's script steps to demonstrate the difference. The same scripting differences would be found for any of the Productive Computing plug-in product lines.

For an example of using plug-in script steps, compare two versions of the same script from the FM Books Connector demo file: Pull Customer__Existing Session.

Script 1 - Pull Customer__Existing Session with calculation ("traditional") plug-in scripting:

```
Set Error Capture [On]
Allow User Abort [Off]
# It is assumed the session is already opened from the previous script calling this
script.
# Query customers in QB (Request)

Set Variable [ $$Result; Value: PCQB_RqNew( "CustomerQuery" ; "" ) ]
Set Variable [ $$Result; Value: PCQB_RqAddFieldWithValue( "ListID" ;
Main::gCust_ListID ) ]
If [ 0 <> PCQB_RqExecute ]
    Exit Script [Text Result:PCQB_SGetStatus]
End If

# Pull customer info into FileMaker (Response)
Set Variable [ $$Result; Value: PCQB_RsOpenFirstRecord ]
Set Field [ main_CUST__Customers::ListID; PCQB_RsGetFirstFieldValue( "ListID" ) ]
Set Field [ main_CUST__Customers::FullName; PCQB_RsGetFirstFieldValue( "FullName" ) ]
Set Field [ main_CUST__Customers::First Name; PCQB_RsGetFirstFieldValue( "FirstName"
) ]
Set Field [ main_CUST__Customers::Last Name; PCQB_RsGetFirstFieldValue( "LastName" ) ]
Set Field [ main_CUST__Customers::Company; PCQB_RsGetFirstFieldValue( "CompanyName" ) ]
Set Field [ main_CUST__Customers::Bill_Address 1; PCQB_RsGetFirstFieldValue(
"BillAddress::Addr1" ) ]
Set Field [ main_CUST__Customers::Bill_Address 2; PCQB_RsGetFirstFieldValue(
"BillAddress::Addr2" ) ]
Set Field [ main_CUST__Customers::Bill_Address 3; PCQB_RsGetFirstFieldValue(
"BillAddress::Addr3" ) ]
Set Field [ main_CUST__Customers::Bill_Address 4; PCQB_RsGetFirstFieldValue(
"BillAddress::Addr4" ) ]
Set Field [ main_CUST__Customers::Bill_City; PCQB_RsGetFirstFieldValue(
"BillAddress::City" ) ]
Set Field [ main_CUST__Customers::Bill_State; PCQB_RsGetFirstFieldValue(
"BillAddress::State" ) ]
Set Field [ main_CUST__Customers::Bill_Postal Code; PCQB_RsGetFirstFieldValue(
"BillAddress::PostalCode" ) ]
Set Field [ main_CUST__Customers::Phone; PCQB_RsGetFirstFieldValue( "Phone" ) ]
Set Field [ main_CUST__Customers::Email; PCQB_RsGetFirstFieldValue( "Email" ) ]

Exit Script [Text Result:0]
```

Script 2 – Pull Customer__Existing Session with plug-in script steps:

```
Set Error Capture [On]
Allow User Abort [Off]
# It is assumed the session is already opened from the previous script calling this
script.
# Query customers in QB (Request)

PCQB_RqNew [Select; Results:$$Result; Request Type:"CustomerQuery"]
PCQB_RqAddFieldWithValue [Select; Results:$$Result; QB Field Name:"ListID"; Field
Value:Main::gCust_ListID]
PCQB_RqExecute [Select; Results:$$Result]
If [$$Result <> 0]
    Exit Script [Text Result:PCQB_SGetStatus]
End If

# Pull customer info into FileMaker (Response)
PCQB_RsOpenFirstRecord [Select; Results:$$Result]
PCQB_RsGetFirstFieldValue [Select; Results:main_CUST__Customers::ListID; Field
Name:"ListID"]
PCQB_RsGetFirstFieldValue [Select; Results:main_CUST__Customers::FullName;
FieldName:"FullName"]
PCQB_RsGetFirstFieldValue [Select; Results:main_CUST__Customers::First Name; Field
Name:" FirstName"]
PCQB_RsGetFirstFieldValue [Select; Results:main_CUST__Customers::Last Name; Field
Name:" LastName"]
PCQB_RsGetFirstFieldValue [Select; Results:main_CUST__Customers::Company; Field
Name:" CompanyName"]
PCQB_RsGetFirstFieldValue [Select; Results:main_CUST__Customers::Bill_Address 1;
Field Name:" BillAddress::Addr1"]
PCQB_RsGetFirstFieldValue [Select; Results:main_CUST__Customers::Bill_Address 2;
Field Name:" BillAddress::Addr2"]
PCQB_RsGetFirstFieldValue [Select; Results:main_CUST__Customers::Bill_Address 3;
Field Name:" BillAddress::Addr3"]
PCQB_RsGetFirstFieldValue [Select; Results:main_CUST__Customers::Bill_Address 4;
Field Name:" BillAddress::Addr4"]
PCQB_RsGetFirstFieldValue [Select; Results:main_CUST__Customers::Bill_City; Field
Name:" BillAddress::City"]
PCQB_RsGetFirstFieldValue [Select; Results:main_CUST__Customers::Bill_State; Field
Name:" BillAddress::State"]
PCQB_RsGetFirstFieldValue [Select; Results:main_CUST__Customers::Bill_Postal Code;
Field Name:" BillAddress::PostalCode"]
PCQB_RsGetFirstFieldValue [Select; Results:main_CUST__Customers::Phone; Field
Name:"Phone"]
PCQB_RsGetFirstFieldValue [Select; Results:main_CUST__Customers::Email; Field
Name:"Email"]

Exit Script [Text Result:0]
```

Using script steps instead of the more traditional methods can make scripting within a solution more direct, as well as help with data entry validation. Some functions accept calculation-style input, while others accept a Boolean "true" or "false" option, and others employ a drop-down list for the developer to choose an option from. As stated earlier, the functionality of the plug-in script step is identical to its functionality as a calculation function; PCQB_RsOpenFirstRecord as a script step will still open the first record in the response, and store the value in the \$\$Result global variable (as seen in Script 2), just the same as the Set Variable script step calls PCQB_RsOpenFirstRecord (which opens the first response record) and stores the result in the \$\$Result variable.

For all Productive Computing, Inc., plug-ins that provide plug-in script step functionality, calculation functions will still be provided for use in development. This is to ensure that scripts already integrated with any of our plug-ins will still be viable and functional, and the developer now has the option to utilize the plug-in script steps at their discretion.

7) Integration Tips and Tricks

Integration of the FM Books Connector Online Edition plug-in can be a challenging and daunting task. When implementing the plug-in with a solution, we have a few tips that may assist a developer in designing proper workflows and connections with QuickBooks Online.

- 1) No two QuickBooks Online companies are exactly alike.

QuickBooks Online company files, like QuickBooks Desktop company files, are as varied as the companies themselves that use them. Some companies make full use of sales orders, estimates, and invoices, ensuring that every transaction is inter-related and fully connected, while others utilize invoices only as receipts for customer purchases and services rendered. Furthermore, two different shipping companies might track their expenses in entirely different manners. Knowing how the intended user's company works makes a significant difference in the complexity of the FM Books Connector Online integration.

- 2) Script for multiple simultaneous user sessions.

When designing the connection process for letting FM Books Connector Online talk to a user's QuickBooks Online company, the best practice is to program for multi-user access; that is, to allow multiple FileMaker users to communicate with the company simultaneously, even if that isn't exactly the case. Such steps could be:

- Always save the session information to a "Global" or "Preferences" table after successfully beginning a session, with "PCQO_SSaveSessionInfo".
- Each FileMaker session, prior to performing the first task that would talk to QuickBooks Online (such as pushing a customer or invoice or pulling items), load any saved session information with "PCQO_SLoadSessionInfo" instead of beginning the session, if possible.
- Only call the "PCQO_EndSession" function when the user explicitly wishes to end the session.
- After any successful call to "PCQO_SLoadSessionInfo", make sure to update the session token field with "PCQO_SSaveSessionInfo". Depending on how frequently your solution loads sessions, the session may have automatically refreshed, so you want to ensure you always have the most up-to-date access token stored for solution use.

- 3) Consider if the user will want to use the plug-in on their FileMaker Server.

Moving some or all of the heavy lifting of the plug-in to FileMaker Server can help improve the performance of any FM Books Connector Online integration and will open up the ability for the user to use mobile or web-based applications of their solution. This is especially useful for service industry customers, for example, that have employees out in the field that need to access the FileMaker solution via FileMaker Go or FM WebDirect.

There are numerous ways to integrate the FM Books Connector Online with a client's solution, and the tips above can help narrow down the scope of any project. If further assistance is needed, Productive Computing, Inc., offers development assistance that can be reached by filling out a Request for Quote (RFQ) or calling our office during normal business hours.

4) Updating from Version 1 to Version 2

The FM Books Connector Online underwent a significant codebase change when updating from version 1 to version 2, which allowed the plug-in to run server-side, on Windows and on Mac, as well as run with FM Cloud. As a result of this update, there may be a need for developers to review their scripting in their system prior to rolling out a live conversion to the newer plug-in build. Here are some key points to consider when updating your environment to version 2 of the plug-in:

1. Authentication on the Mac side is more simplified than the Windows side; the user will not need to copy and paste the authentication string from the web browser into a custom dialog in FileMaker. Instead, a sub-application will launch and control the authentication, and on success the authentication information will be stored internally automatically.
2. When working with ReferenceType objects (such as ParentRef, CustomerRef, or other ___Ref fields), the use of the "name" sub-field alone is insufficient to specify the reference. Developers will need to make sure to include at minimum the "value" sub-field for any ReferenceType object. For example: "CustomerRef::value" alone will work, but specifying just "CustomerRef::name" without also providing "CustomerRef::value" will be insufficient.
3. The internal format of data handled by the plug-in changed from XML-serialized objects to JSON-serialized objects. The functions for retrieving XML from the plug-in are still present, though they return JSON documents instead of XML, and are thus deprecated. Loading pre-generated QuickBooks Online object data only accepts JSON-style documents.

As always, we highly recommend reviewing any integration with our plug-ins when performing a major version update, thus ensuring that any strange behavior is revealed and resolved prior to going "live".

8) Authorization - QuickBooks Online Connection and Establishing a Session

Developer's Note

As of December 17th, 2019, Intuit's QuickBooks Online service now uses OAuth 2.0 in order to facilitate authentication for third-party applications such as the FM Books Connector Online plug-in. For our users, the authentication process will be the same as it has been before; the plug-in will launch authentication with PCQO_BeginSession, the user will be prompted to log into their Intuit account and authorize the plug-in to communicate with their QuickBooks Online company, Windows users will copy their session ticket from the website into FileMaker to pass to PCQO_Authorize, and finally the plug-in will be authenticated. The major key difference is that while OAuth 1.0 sessions last for up to 6 months, OAuth 2.0 sessions last for only 1 hour, but also include a higher level of security for user connections. The FM Books Connector Online plug-in will automatically ensure that each session is refreshed while connected as the plug-in works, so long-term connections (e.g. connections lasting longer than one hour) will continue to remain connected.

If your deployment version of FM Books Connector Online is older than version 3.0.0.0, you must upgrade your deployment to the latest version before December 17th, 2019. After that date, any attempts to connect with OAuth 1.0 will be declined by Intuit.

Authorization

FM Books Connector Online uses a new session management style called "OAuth 2.0". There are no installers required to use OAuth 2.0. See the Demo file "Begin Session" script for an example of the authorization process.

Windows Plug-in Authentication

When calling "PCQO_BeginSession", the default browser of the client machine will open up and navigate to Intuit's OAuth server, prompting the user to log into the QuickBooks Online company that the plug-in will connect to. Upon successful authorization, the user will be redirected to a page containing the unique session ticket string. This session ticket is used in a call to "PCQO_Authorize" to complete the process and ensure that the plug-in will use the authorized session information when exchanging data with QuickBooks Online company. Calling "PCQO_EndSession" will terminate the authorized session, and beginning a new session will require acquiring a new session ticket.

Mac Plug-in Authentication

Starting in Version 2 of the FM Books Connector Online plug-in, the authentication process has been updated to be more streamlined, by the steps below:

1) Begin the Session

- a. To begin the authentication process, the solution will need to call the PCQO_BeginSession(optSandbox) function, passing either "true" if connecting to a Sandbox QuickBooks Online company, or "false" if connecting to a Live QuickBooks Online company.

2) Authenticate the User

- a. The plug-in will create and display a modal dialog window upon beginning the session, bringing the user to the Intuit connection page, where they will log in with their access credentials, optionally choose a company to connect to (if they have access to more than one company with those credentials), and authorize the FM Books Connector Online to exchange data with the company.

3) Validate the connection

- a. Once the user clicks the "Authorize" button, the window will close and the plug-in will validate the connection information by querying for the company's information. If the query is invalid (such is the case when the authorization process fails), the plug-in will report an error and the authentication process will need to be started again.

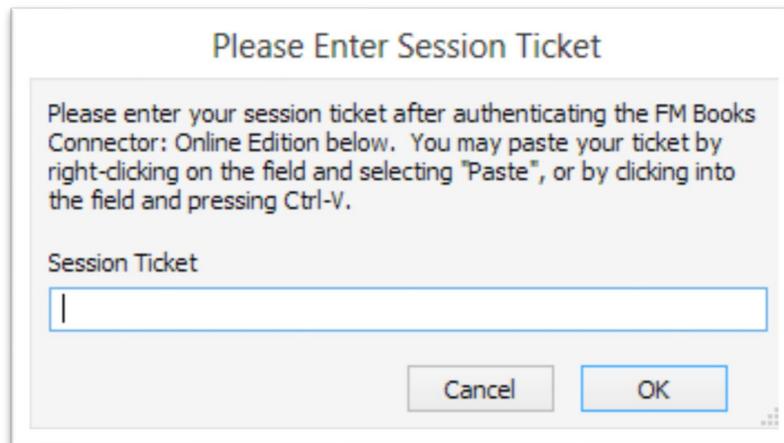
This differs from the Windows version in that the plug-in no longer launches an internet browser window and relies on a subsequent call to PCQO_Authorize to complete the authentication process.

NOTE: Intuit currently restricts the number of active sessions between a third-party application and a QuickBooks Online company to one session at a time. However, as of version 1.0.1.0 you will be able to share that single connection across multiple machines with custom scripting. Please see section Multi-User Session Management for more details. For further information on this restriction, please refer to the Known Issues section of this document.

Begin a QuickBooks Online Session for Windows using the Demo File

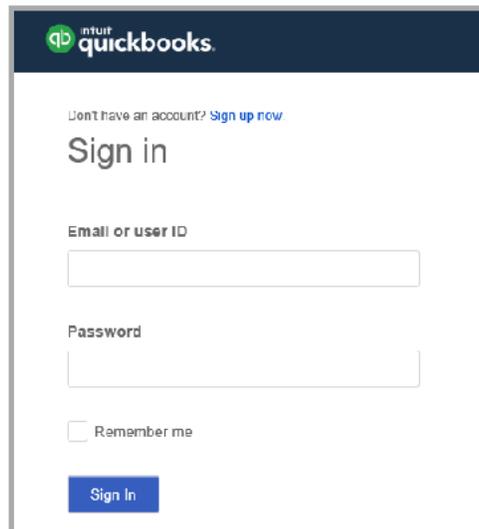
To begin a session with QuickBooks Online Demo file for Windows, go to Step 2 of the plug-in screen and click Begin Session.

The "Please Enter Session Ticket" window will open while FM Books Connector Online is connecting to QuickBooks Online. Do not close out of this window and do not enter any information yet.



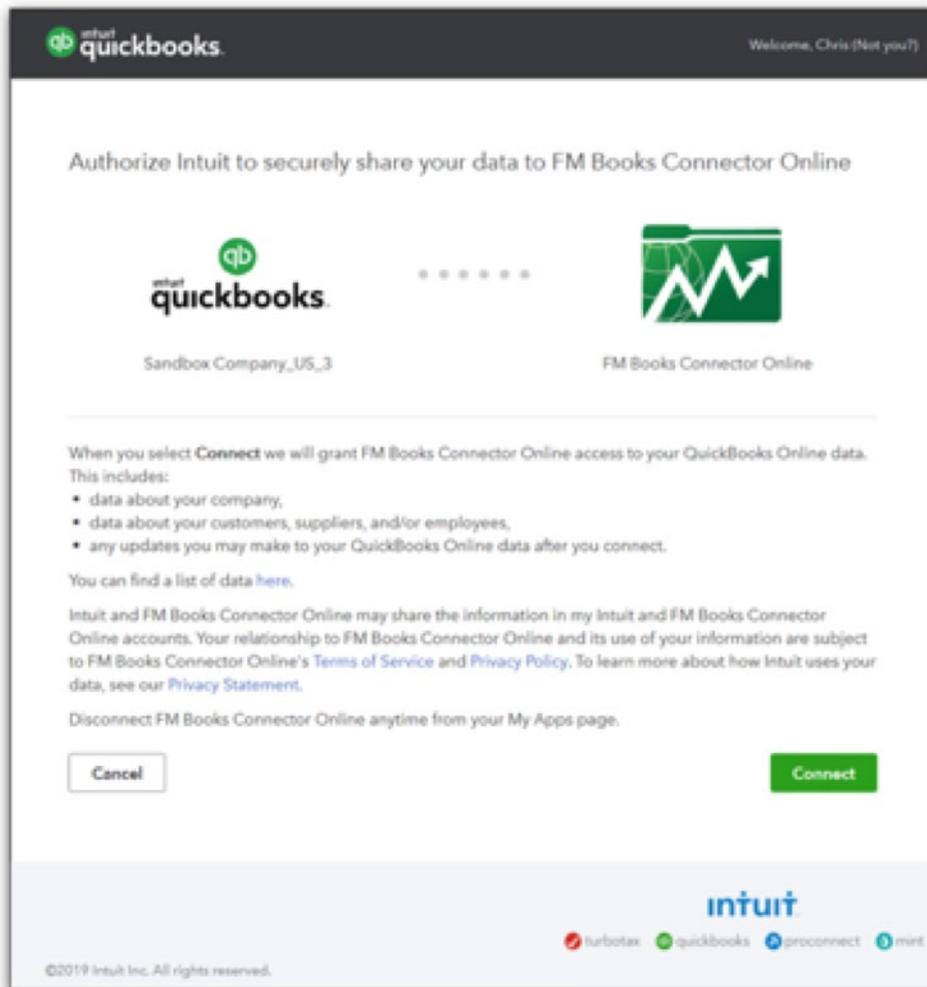
The dialog box has a title bar that says "Please Enter Session Ticket". Inside, there is a text area with the following text: "Please enter your session ticket after authenticating the FM Books Connector: Online Edition below. You may paste your ticket by right-clicking on the field and selecting 'Paste', or by clicking into the field and pressing Ctrl-V." Below this text is a text input field labeled "Session Ticket" with a vertical cursor on the left. At the bottom right of the dialog are two buttons: "Cancel" and "OK".

An internet browser window will open to connect with your QuickBooks Online company account. Enter your QuickBooks Online user ID name and password.

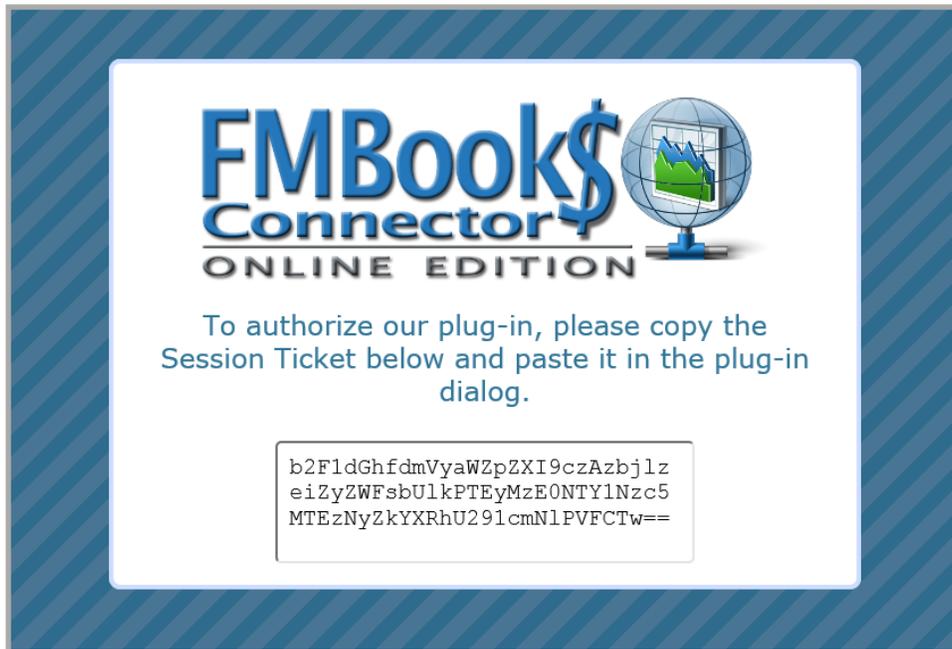


The sign-in page features the Intuit QuickBooks logo at the top left. Below the logo is a link: "Don't have an account? [Sign up now](#)". The main heading is "Sign in". There are two input fields: "Email or user ID" and "Password". Below the password field is a checkbox labeled "Remember me". At the bottom is a blue "Sign In" button.

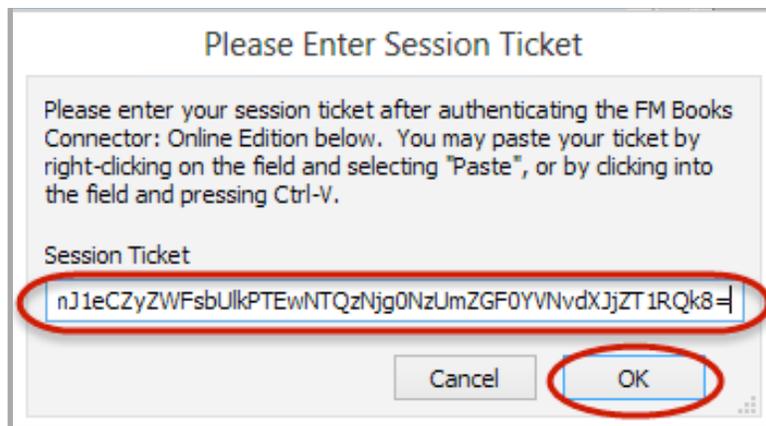
If your login credentials are accepted, the browser will display "FM Books Online would like to access your Intuit company data." This step is required to establish a link between your QuickBooks Online company account and FM Books Connector Online. Click "Authorize" to proceed.



Once authorized, the browser will display a Session Ticket as shown below. Copy the Session Ticket string. If a dialog is displayed stating that an application is trying to close the window, simply press "OK." The Session Ticket dialog should be open in the foreground of your browser. Please note connection speeds may vary when communicating with the Intuit QuickBooks Online servers.



Paste your Session Ticket into the "Please Enter Session Ticket" window and select "OK" as shown below.



If successful, you will receive an "Authentication successful" window. Click "OK". The Demo file will display Result = "0" and the session ticket will be visible on the screen.

You should now have an active session with QuickBooks Online.

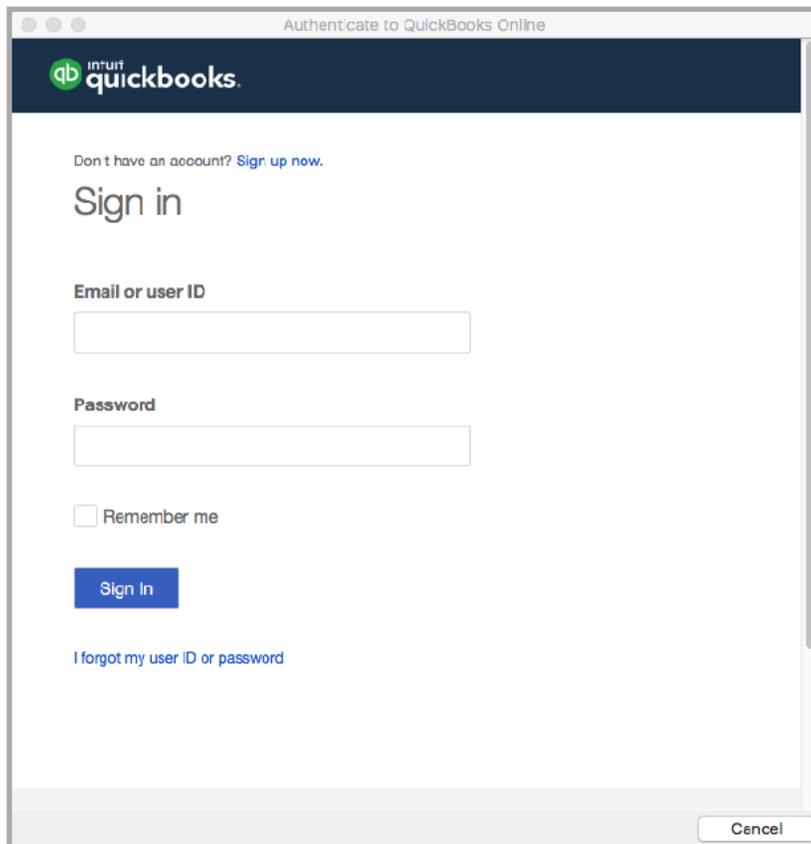
Begin a QuickBooks Online Session for Mac using the Demo File

To begin a session with QuickBooks Online Demo file for Mac, go to Step 2 of the plug-in screen and click Begin Session.

Step 2: Quick Books Online Connection Test

[Begin Session](#) Session Mode: Live Sandbox Result: A result of 0 = Success

The plug-in will then display an Intuit login window where you will log in with your access credentials.



The screenshot shows a browser window titled "Authenticate to QuickBooks Online". The page features the Intuit QuickBooks logo at the top. Below the logo, there is a link for "Don't have an account? Sign up now." followed by the "Sign in" heading. The form includes two input fields: "Email or user ID" and "Password". Below these fields is a "Remember me" checkbox. A blue "Sign In" button is positioned below the checkbox. At the bottom of the form, there is a link for "I forgot my user ID or password". A "Cancel" button is located in the bottom right corner of the window.

You should now have an active session with QuickBooks Online.

Multi-User Session Management

This section discusses how to manage multi-user situations with the FM Books Connector Online edition. This is a fairly technical section that developers can and should use when deploying the FM Books Connector Online plug-in in environments in which more than one physical client machine will need to communicate with QuickBooks Online.

As of version 1.0.1.0, the FM Books Connector Online plug-in now supports multi-user session management, or "session sharing". This will allow a single machine to initialize the communication session with QuickBooks Online using the FM Books Connector Online, and then share that connection with other machines using the FM Books Connector Online that are connected to the same database.

When the FM Books Connector Online plug-in authorizes a session ticket received from Intuit, it saves the session state internally, allowing the plug-in to make authorized calls to QuickBooks online to add customers, pull items, edit invoices, and more. This session state can be exported from the plug-in into a FileMaker field, which can then be read by another copy of the plug-in and set as the second plug-in's session state. This will allow the second plug-in to make further requests on the same session, simultaneously with the first plug-in.

The following script demonstrates how to save the session information to a text field located in a table called "Preferences".

```
# Assume that the session has already begun, and the user has been authenticated..
#
# Get the active session information
Set Variable [ $sessionInfo, Value: PCQO_SSaveSessionInfo ]
If ( $sessionInfo = "!!ERROR!!" )
    # Handle error...
Else
    # Set the session field with the active QBO session information
    Set Field [ Preferences::QBOSessionInfo, Value: $sessionInfo ]
End If
#
```

When loading the active session into the FM Books Connector plug-in (such as from another machine, or after closing and reopening FileMaker), the following script demonstrates the steps necessary to restore the QuickBooks Online session.

Please note that if the PCQO_EndSession command is called at any point while an active session is in use by the plug-in, that session WILL be terminated, and any other plug-ins using the same session will receive errors when trying to communicate with QuickBooks Online.

```
# Assume that the fields and tables referenced below are the same as the previous sample
script
#
# Load the session information from the Preferences table into the plug-in
Set Variable [ $result, Value: PCQO_SLoadSessionInfo( Preferences::QBOSessionInfo ) ]
If ( $result = "!!ERROR!!" )
    # Handle error...
Else
    # Report success, set connection state flags, etc.
End If
#
```

A good practice is to restrict access to the PCQO_BeginSession and PCQO_EndSession functions to admin-level users and allow end users to connect and disconnect as they need with a combination of PCQO_SLoadSessionInfo and connection status flags within your solution.

Due to the recent change to authorizing to Intuit with OAuth 2.0, it is possible that the FM Books Connector Online plug-in will automatically refresh its session during the course of its workflow, especially if the plug-in is expected to communicate with QuickBooks Online for longer than one hour. If a session token expires during communication, it may become refreshed during either a call to PCQO_RqExecute or PCQO_SLoadSessionInfo. We advise that developers include a procedure to store the session information if it changes by immediately calling PCQO_SSaveSessionInfo in the event of an automatic refresh. For an example, developers can check to see if the current session info string in FileMaker matches what the plug-in is using by comparing it to the result of PCQO_SSaveSessionInfo, and update the FileMaker field if the two are different. This will ensure that any users loading that session information will be using the latest, up-to-date session, and will mitigate any authentication errors from Intuit.

9) Server-Side Deployment

As of version 2.0.1.0, the FM Books Connector Online plug-in is able to run on FileMaker Server within the FileMaker Server Scripting Engine or Web Publishing Engine. This section will go into detail on how to install the plug-in, how the plug-in handles registration, session management, and functionality.

Installation

The server plug-in will need to be installed on the server machine within the FileMaker Server “Extensions” folder. See the paths below for the plug-in installation locations on both Windows and Mac servers (assuming a default installation path):

Windows

Database Server:

C:\Program Files\FileMaker\FileMaker Server\Database Server\Extensions

Web Direct & Custom Web Publishing:

C:\Program Files\FileMaker\FileMaker Server\Web Publishing\publishingengine\cwpc\Plugins

Mac

Database Server:

/Library/FileMaker Server/Database Server/Extensions

Web Direct & Custom Web Publishing:

/Library/FileMaker Server/Web Publishing/publishing-engine/cwpc/Plugins

The Demo file available from our website provides a one-click installation method for installing the FM Books Connector Online Server-side plug-in into a host FileMaker Server’s Extensions folder; however, it will only install it into the Database Server location above. The demo file must be hosted on FileMaker Server in order to install the plug-in into the Server’s Extensions folder. In order to install the server-side plug-in for use in Web Direct or Custom Web Publishing, the plug-in can be installed by opening the hosted server-side demo file using Web Direct and using the Web Direct Setup tab, or it can be installed manually.

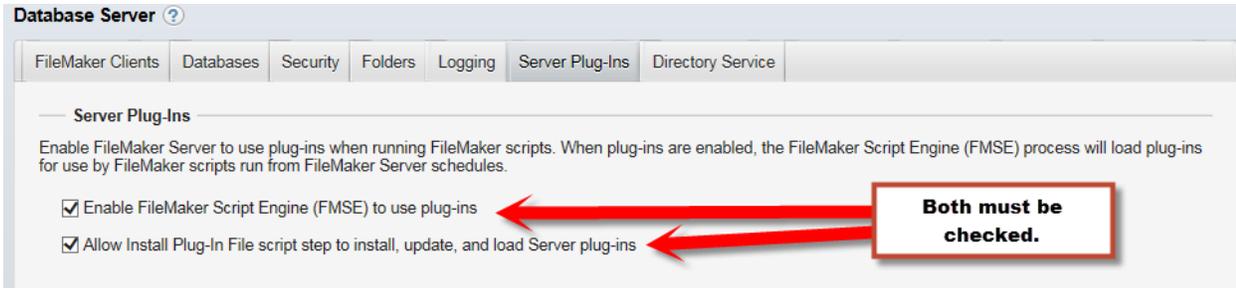
Any manual installation of the plug-in will require the server or web publishing engine to be restarted in order for the plug-in to load.

An important note: In order to allow FileMaker Server to install and use a server-side plug-in from a hosted database, both options in the Database Server > Server Plug-ins section in the Admin Console must be checked. See screenshot below.

Please refer to the link below for further information on managing server-side plug-ins on FileMaker Server.

https://www.filemaker.com/help/15/fms/en/index.html#page/fms/fmsh_plugins.14.1.html

Database Server Plug-in Options:



Web Direct Plug-in Options:



Deployment Requirements

In order to utilize the FM Books Connector Online server-side plug-in, the plug-in must be installed at the location above on the FileMaker Server machine, and the solution that has scripts designed to utilize the plug-in must be hosted on that FileMaker Server machine. This is easily done by following the instructions for uploading a solution to FileMaker Server, which can be reviewed in the FileMaker Server User's Guide, which comes with every version of FileMaker Server.

Registration

Like the client version, the server version of FM Books Connector Online must be registered in order to function. In order for the plug-in to function on the server machine, a call to `PCQO_Register` with valid license information must be called at the beginning of any scripted process. This process should be hard-coded and reference either passed script parameters from a calling script or pre-populated fields within the FileMaker solution. Please see the "Recommended Script Structure" section below for an example of how registration can be utilized.

Authentication with QuickBooks Online

The FM Books Connector Online server-side plug-in cannot initiate or authenticate a new session with Intuit QuickBooks Online on its own. This is in part due to the restriction for FileMaker server-side plug-ins that prevents the use of external dialog windows or pop-ups. In order to allow the FM Books Connector Online server-side plug-in to communicate with QuickBooks Online while respecting the FileMaker server-side plug-in restrictions, the information of a session must be passed to the server-side plug-in during any scripted process.

A client-side version of FM Books Connector Online must proceed with beginning and authenticating a session with QuickBooks Online. Once this is successful, the solution must call "PCQO_SSaveSessionInfo" and store the resulting encrypted session information string within a field in the hosted FileMaker solution. When the server-side plug-in is then called to perform a task (such as a user accessing the solution via FileMaker Go and seeking to push a set of invoices to QuickBooks Online), the scripted process must load that stored session information string by calling "PCQO_SLoadSessionInfo" and using the session info field's value. If the result of the call is 0, the connection has been established and the server-side plug-in is able to communicate freely with QuickBooks for the duration of the scripted process.

Recommended Script Structure

In order to best illustrate the concepts of a script process, handling for registration, and handling for QuickBooks Online authentication, we have prepared a sample script example pulled from the server-side plug-in demo file. This example is the "Pull All Customer Names" process that pulls all customer names from a QuickBooks Online company and stores them in the hosted FileMaker solution's Customers table. For ease of example, this script process involves a "Perform Script On Server" script step that calls the Pull All Customers Names script and waits for completion.

Script 1: Pull All Customer Names [Server]

```
# Ensure the solution is hosted
If [IsEmpty( Get(HostIPAddress) )]
    Show Custom Dialog ["File Not Hosted"; "This solution must be hosted to use this
feature."]
    Halt Script
End If

If [$$$IsWebDirect]
    # $$$IsWebDirect is a true or false flag that is set to true if the file is
    # being accessed via Web Direct. This would be set in the file's opening script.

    # Perform the script for Web Direct. Web Direct uses "client"-style scripting
    # just as well as Perform Script On Server.
    Perform Script ["SVR - Pull All Customers Names"]
Else
    # Perform the script on the server
    Perform Script on Server [Wait for completion; "SVR - Pull All Customers Names"]
End If

# Get the result and check for error
If [Get(ScriptResult) <> 0]
    Show Custom Dialog ["Pull Failed"; "Failed to pull customer names: " &
Get(ScriptResult)]
End If
```

Script 2: SVR – Pull All Customers Names

```
# Ensure the plug-in is registered. In this demo, we use subscripts for better readability.
Perform Script ["SVR - Register Plug-in"]
If [Get(ScriptResult) <> 0]
    # There was an error with registration. Return the result back to the client.
    Exit Script [Text Result:Get(ScriptResult)]
End If

# Ensure the plug-in has an authorized connection.
Perform Script ["SVR - Load Session"]
If [Get(ScriptResult) <> 0]
    # There was an error with loading the session. Return the result back to the client.
    Exit Script [Text Result:Get(ScriptResult)]
End If

# Clear any existing customer records
Go to Layout ["Customer List" (Customers)]
Show All Records
Delete All Records [With dialog:Off]

# Create the request (we want all customers in the QuickBooks Online company)
Set Variable [$$Result; Value: PCQO_RqNew( "Query" ; "Customer" )]

# Execute the request
Set Variable [$$Result; Value: PCQO_RqExecute]
If [$$Result <> 0]
    # There was an error during execution. Return the result back to the client.
    Exit Script [Text Result:PCQO_SGetStatus]
End If

# Process the response and create records in FileMaker
Set Variable [$$Result; Value: PCQO_RsOpenFirstRecord]
Set Variable [ $Count; Value: 0 ]
Loop
    Exit Loop If [$$Result = "!!ERROR!!" or $$Result = "End" or $$Result = "?" or $Count
= 100 // Only pull the first 100 customers]
    New Record/Request
    Set Variable [ $Count; Value: $Count + 1 ]
    Set Field [Customers::QB_List_ID; PCQO_RsGetFirstFieldValue( "Id" )]
    Set Field [Customers::Customer_Name; PCQO_RsGetFirstFieldValue( "FullyQualifiedName"
)]
    Set Field [Customers::Total_Balance; PCQO_RsGetFirstFieldValue( "BalanceWithJobs" )]
    Set Variable [$$Result; Value: PCQO_RsOpenNextRecord]
End Loop

# At the end of the script, since we would have exited if there was an issue, return 0.
Exit Script [Text Result:0]
```

Script 3: SVR – Register Plug-in

```
# Register the plug-in using the license information stored in the solution
Set Variable [ $result; Value: PCQO_Register( Main::RegistrationServer;
Main::RegistrationPort; Main::RegistrationPage; Main::RegistrationLicenseID ) ]
If [ $result <> 0 ]
    Exit Script [Text Result:$result]
Else
    Exit Script [Text Result:0]
End If
```

Script 4: SVR – Load Session

```
# Check if the session information is stored
If [IsEmpty( Main::Session Info String )]
    Exit Script [Text Result:"No session information is saved."]
End If

# Load the QBO session using the session information stored in the solution
Set Variable [$result; Value: PCQO_SLoadSessionInfo( Main::Session Info String )]
If [$result <> 0]
    Exit Script [Text Result:PCQO_SGetStatus]
Else
    # The session may have been automatically refreshed, so update the session
    string field with the latest session data.
    Set Field [ Main::Session Info String ; Value: PCQO_SSaveSessionInfo ]
    Exit Script [Text Result:0]
End If
```

Script #1 is considered the "Driver" script. This kind of script is usually tied to a button control or some other interactive object, and its purpose is to ensure that the server-side script has what it needs from the client, and then calls the "Worker" script using Perform Script on Server. In a Web Direct environment, the "Driver" script can simply call the "Worker" script directly, or it can call Perform Script on Server if the developer decides to make the server scripting engine's plugin perform the work.

Script #2 is considered the "Worker" script. This script is the entire scope of the script session, and is the "scripted process" mentioned in the earlier sections. At the beginning of the Worker script, a call to register the server-side plug-in (Script #3) and load a stored QuickBooks Online session (Script #4) must be made successfully before any further progress can be made. The actual work scripting is identical between the client and server styles, with the notable exception being that there can be no dialog windows used when running in the server session.

Once the Worker script has completed all of its script steps, control will return back to the Driver script, and the server-side plug-in will release the registration and session information that it has, and return to the original "neutral" state it was in before the call to the Worker script. You can call multiple Worker scripts *sequentially*, and each call will register the plug-in, load the session information, and perform its task before returning to the Driver script.

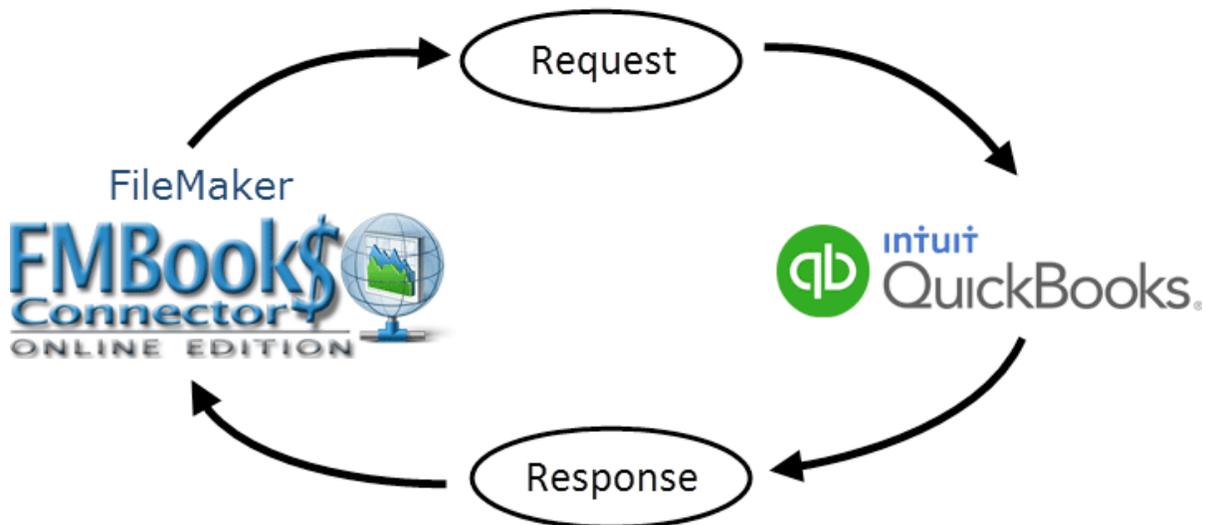
It is important to note that allowing multiple *simultaneous* Worker script sessions to run, while supported, is not advised. There are conditions that may allow information to be overridden by different sessions, which can lead to unexpected behavior in the solution. It is therefore the responsibility of the developer to ensure that any calls to use a server-side plug-in do so in a way that prevents this kind of problem as best as possible.

If you need assistance with developing a server-side deployment of the FM Books Connector Online, please feel free to reach out to Productive Computing, Inc., for developer assistance. We can be reached between 8:00 AM and 5:00 PM Pacific Standard Time at our phone number (760) 510-1200, or via our support email at support@productivecomputing.com.

10) Talking to QuickBooks Online

It is easiest to picture the information exchanged between the plug-in and QuickBooks Online 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 below.

FileMaker and QuickBooks Online Information Exchange



No matter what the request is, whether it is to add contacts to a QuickBooks Online 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:

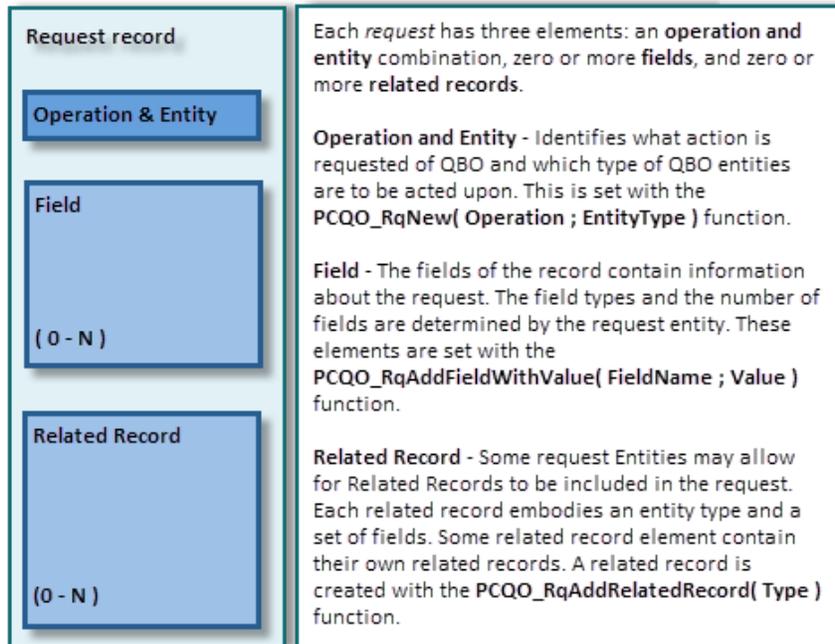
- a) The plug-in always initiates any conversation with a request.
- b) QuickBooks Online always responds to a properly posed request.
- c) The conversation is always finished after QuickBooks Online responds to the request. The request and response conversation is the foundation for exchanging information between FileMaker and QuickBooks Online. It is imperative to remember that this is the form of the conversation no matter what the plug-in is requesting of QuickBooks Online. It will be especially useful to remember this when it comes time to create scripts in your FileMaker solution for exchanging data with QuickBooks Online.
- d) We now know how the plug-in talks to QuickBooks Online. 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 into the explicit 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 Online, and how to read QuickBooks Online's response.

Request and Response Explained

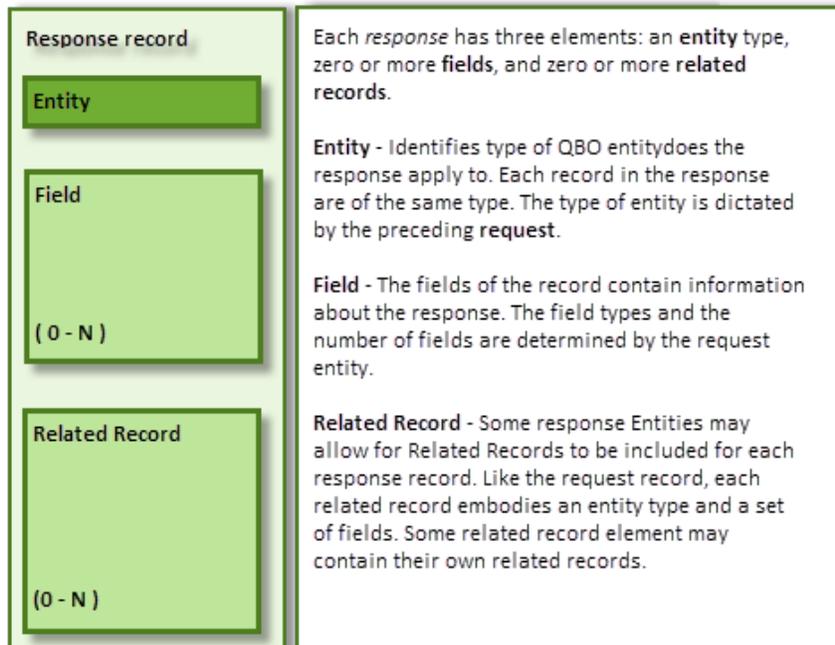
Requests are created and responses are read using external functions. Different requests and responses have different fields defined for them. Furthermore, requests and responses may have related items.

Please refer to the diagrams below for an explanation of the three different elements for both a request and response.

Request Model



Response Model



Create a Request

Each request has both an Operation and an Entity type. The operation defines what action to take and the entity type defines what type of record will the action act upon. For example, a request message with an operation of "Create" and an entity of "Customer" will tell QuickBooks Online to create a new customer record in the QuickBooks Online file. The available operations and entity types for a request are listed in the ESR and are left out of this document for the sake of brevity.

Some sample request operation and entity type combinations and their definitions follow:

"Update" and "Customer": the request wants to update/modify an existing Customer record in QuickBooks Online.

"Query" and "Invoice": the request wants to find Invoice records in QuickBooks Online.

"Create" and "PurchaseOrder": the request wants to add a Purchase Order record to QuickBooks Online.

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 entity types are also listed in the ESR.

Two functions are used to create a basic request:

```
PCQO_RqNew( Operation ; EntityType )
PCQO_RqAddFieldWithValue( FieldName ; Value )
```

The first function is used to create a request with the desired Operation and Entity Type, and the second is used to populate the fields of the request.

A simple request to add a Customer follows:

```
PCQO_RqNew( "Create" ; "Customer" )
PCQO_RqAddFieldWithValue( "DisplayName" ; "Bob Jones" )
PCQO_RqAddFieldWithValue( "BillAddr::Line1" ; "123 Any Street" )
PCQO_RqAddFieldWithValue( "BillAddr::City" ; "Any Town" )
PCQO_RqAddFieldWithValue( "BillAddr::CountrySubDivisionCode" ; "Any State" )
PCQO_RqAddFieldWithValue( "BillAddr::PostalCode" ; "11111" )
PCQO_RqAddFieldWithValue( "PrimaryEmailAddr" ; "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:

```
PCQO_RqAddRelatedRecord( Type )
PCQO_RqCloseRelatedRecord
```

A sample Create Invoice request demonstrates the usage of these two functions.

PCQO_RqNew("Create" ; "Invoice")

PCQO_RqAddFieldWithValue("CustomerRef::name" ; "Bob Jones")

PCQO_RqAddFieldWithValue("TxnDate" ; "2006/01/01")

PCQO_RqAddFieldWithValue("DocNumber" ; "123456789")

PCQO_RqAddFieldWithValue("BillAddr::Line1" ; "123 Any Street")

PCQO_RqAddFieldWithValue("BillAddr::City" ; "Any Town")

PCQO_RqAddFieldWithValue("BillAddr::CountrySubDivisionCode" ; "Any State")

PCQO_RqAddFieldWithValue("BillAddr::PostalCode" ; "11111")

PCQO_RqAddRelatedRecord("Line")

PCQO_RqAddFieldWithValue("Amount" ; "79.95")

PCQO_RqAddFieldWithValue("DetailType" ; "SalesItemLineDetail")

PCQO_RqAddRelatedRecord("SalesItemLineDetail")

PCQO_RqAddFieldWithValue("ItemRef::name" ; "Widgit")

PCQO_RqAddFieldWithValue("Qty" ; "20")

PCQO_RqCloseRelatedRecord

PCQO_RqCloseRelatedRecord

PCQO_RqAddRelatedRecord("Line")

PCQO_RqAddFieldWithValue("Amount" ; "2.95")

PCQO_RqAddFieldWithValue("DetailType" ; "SalesItemLineDetail")

PCQO_RqAddRelatedRecord("SalesItemLineDetail")

PCQO_RqAddFieldWithValue("ItemRef::name" ; "Gadget")

PCQO_RqAddFieldWithValue("Qty" ; "3")

PCQO_RqCloseRelatedRecord

PCQO_RqCloseRelatedRecord

You will notice that the PCQO_RqAddFieldWithValue function operates in the context of the current record. When the PCQO_RqAddRelatedRecord function is called, the context shifts from the parent record (the main request) to the related record, in this case the Line record (and again the SalesItemLineDetail record). Subsequent calls to PCQO_RqAddFieldWithValue will add field values to the related record until the PCQO_RqCloseRelatedRecord function is called and the context is shifted back to the parent record. For each call to PCQO_RqAddRelatedRecord, there needs to be a corresponding call to PCQO_RqCloseRelatedRecord to ensure that the request is formed properly.

Of particular note is that adding related records of the type "Line" or other records similar to Line (such as "TaxLine"), the related records have a "Header" section that holds the basic information of the Line, including its detail type, and the "Detail" section that holds specific information about the line's detail type. The available fields and detail names, as well as structure of the Lines related records, are located in the ESR for reference.

Post the Request

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

PCQO_BeginSession

PCQO_RqExecute

PCQO_EndSession

PCQO_BeginSession is used to begin and secure a session with a QuickBooks Online company. Once a session has begun and the user has successfully authorized the plug-in to communicate with a desired QuickBooks Online company, the plug-in may make calls to exchange information with that company freely. Because of the nature of an authorized session, it is advised to implement a method that best suits your solution, whether to begin the session in the morning and end it at the end of the day, or to utilize the PCQO_SSaveSessionInfo and PCQO_SLoadSessionInfo functions to persist the session across multiple days. For more information, please see the "Multi-User Session Management".

Requests posted during PCQO_RqExecute will be subject to validation by QuickBooks Online, and an error will be returned for an insufficient, invalid or malformed request.

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 'Create Customer' request) QuickBooks Online will respond with a 'Customer' record that the plug-in will hold in memory. Query requests such as a 'Query Invoice' 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:

```
PCQO_RsOpenFirstRecord  
PCQO_RsOpenNextRecord  
PCQO_RsOpenFirstRelatedRecord( Type )  
PCQO_RsOpenNextRelatedRecord  
PCQO_RsCloseRelatedRecord  
PCQO_RsGetFirstFieldValue( FieldName )
```

The first two functions are used to iterate through all the records in the response. `PCQO_RsOpenFirstRecord` opens the first record in the response and `PCQO_RsOpenNextRecord` is used to open successive records in the response. `PCQO_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 `PCQO_RsGetFirstFieldValue(FieldName)` function. The name of the desired field is passed with the function and the contents of the field are returned. See the ESR 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 `PCQO_RsOpenFirstRelatedRecord(Type)` and `PCQO_RsOpenNextRelatedRecord` functions. Once the related record is opened, its contents can be read with the `PCQO_RsGetFirstFieldValue` function.

An example of reading a response to a 'Query Invoice' request follows:

```
#Open the First record in the response
SetField[ someField ; PCQO_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 account
  SetField [someField ; PCQO_RsGetFirstFieldValue( "CustomerRef::name" ) ]
  SetField [someField ; PCQO_RsGetFirstFieldValue( "ARAccountRef::name" ) ]
  # Get any and all related transactions to the current invoice
  SetField [someField ; PCQO_RsOpenFirstRelatedRecord( "LinkedTxn" ) ]
  Loop
    Exit Loop If [ (someField < 0) or (someField = "END") ]
    #Gets the information from the related transaction
    SetField [someField ; PCQO_RsGetFirstFieldValue( "TxnId" ) ]
    SetField [someField ; PCQO_RsGetFirstFieldValue( "TxnType" ) ]
    #Opens the next related transaction
    SetField [someField ; PCQO_RsOpenNextRelatedRecord ]
  End Loop
  #Close the related record to return to the main record
  SetField [someField ; PCQO_RsCloseRelatedRecord ]
  #Opens the invoice line item related records
  SetField [someField ; PCQO_RsOpenFirstRelatedRecord( "Line" ) ]
  Loop
    Exit Loop If [ (someField < 0) or (someField = "End") ]
    #Gets the information from the line item header
    SetField [someField ; PCQO_RsGetFirstFieldValue( "Amount" ) ]
    SetField [detailTypeField ; PCQO_RsGetFirstFieldValue( "DetailType" ) ]
    SetField [someField ; PCQO_RsOpenFirstRelatedRecord( detailTypeField ) ]
    #Gets the information from the line item detail
    SetField [someField ; PCQO_RsGetFirstFieldValue( "ItemRef::name" ) ]
    SetField [someField ; PCQO_RsCloseRelatedRecord ]
    #Opens the next line item
    SetField [someField ; PCQO_RsOpenNextRelatedRecord ]
  End Loop
  #Close the related record to return to the main record
  SetField [someField ; PCQO_RsCloseRelatedRecord ]
  #Open the next record in the response
  SetField [someField ; PCQO_RsOpenNextRecord ]
End Loop
```

As with building a request using "Line" related records, processing a response with "Line" related records (or "TaxLine", "PaymentLine", or other "Line"-type records) will require first opening the Line's Header section, then the Line's Detail section, to pull all relevant information from the Line related record.

Send Invoice as PDF

QuickBooks Online offers the ability to send an invoice as a PDF attached to an email to the invoice's linked Customer. The FM Books Connector Online can similarly perform this task using two different methods: the "Long" method and the "Short" method.

The "Long" method involves creating a new request to send the invoice, populating the invoice ID to be sent and the email address to send the email to, and then executing the request. The following pseudoscript demonstrates a quick Long method call to email an invoice to a client:

```
Set Variable [ $result; Value:PCQO_RqNew( "Send" ; "Invoice" ) ]
Set Variable [ $result; Value:PCQO_RqAddFieldWithValue( "Id" ; Invoice::QBID ) ]
Set Variable [ $result; Value:PCQO_RqAddFieldWithValue( "Email" ; Invoice::ContactEmail ) ]
```

The "Short" method is a much quicker and simpler process, performing a Send Invoice request in the span of a single function call. Naturally, the plug-in must be registered and authenticated to QuickBooks Online before the function call can be made. The following pseudoscript demonstrates the ease of calling the Short function:

```
Set Variable [ $result; Value:PCQO_SSendInvoice( Invoice::QBID ; Invoice::ContactEmail ) ]
```

The developer must decide whether to use the Long or Short methods. Both have been included for ease of availability.

Custom Fields

Custom fields are accessed using CustomField objects. If a QB object (invoice, customer, etc.) supports custom fields then the list of available fields in the Response for the object will contain one or more CustomField objects. Accessing the CustomField object in a response is the same as accessing any other related record of the response. Setting values for CustomField objects is accomplished with adding the CustomField object as a related record. One must be familiar with creating requests using the ESR to successfully modify CustomField objects.

This CustomField object is available in the following QBO List Entities:

- PaymentMethod
- Term

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

- Bill
- CreditMemo
- Estimate
- Invoice
- Line (maximum of 3 CustomField objects)
- PurchaseOrder
- SalesReceipt
- TimeActivity

Since custom fields are more advanced and can be quite complex, we are available for hire to assist with this development. You may also reference custom fields in the DataTypes section of the ESR for further details about getting and setting information for CustomFields, and reference the listed entities above for the particular ways to handle CustomField objects for those entities.

Minor Versions

QuickBooks Online makes use of a feature called “minor versions”, which are revisions to the Accounting API engine as a result of bug fixes and enhancements. As a general rule, the FM Books Connector Online plug-in uses the latest Minor Version value as of the release date. Naturally, these minor versions can quickly fall behind the Quickbooks Online versions as Intuit performs updates during their normal development cycle.

Introduced in version 2.0.5.0, the FM Books Connector Online plug-in will now allow the FileMaker developer to access and update the internal minor version value held by the plug-in. Two new functions were added to provide this service: PCQO_SGetMinorVersion, and PCQO_SSetMinorVersion(MinorVersion).

PCQO_SGetMinorVersion returns the current minor version of the plug-in.

PCQO_SSetMinorVersion will set the current minor version of the plug-in with the provided minor version number value.

It is important to note that minor versions only persist within a FileMaker session; once FileMaker closes and reopens, the plug-in will default back to the original base minor version, which is hard coded into the plug-in at release time.

For more information on minor versions and what each minor version added or changed, please reference the following link:

<https://developer.intuit.com/app/developer/qbo/docs/learn/explore-the-quickbooks-online-api/minor-versions>

Batch Request and Response Handling

Introduced in version 2.0.5.0, the FM Books Connector Online plug-in will allow the support of Batch scripting, letting developers bundle many different requests together within a batch queue and executing them all at once. These requests can be any of the four “CRUD” request operations: Create, Read (“Query”), Update, or Delete. Batch requests are processed in the same order as they are received, and their responses are stored in the same order: the first batch request will result in the first batch response, the second request will result in the second response, and so forth. Batch responses can be successful or errors, as appropriate for the situation.

When handling batches, the following new functions are used. Please refer to the Functions Guide for more details about these functions:

- PCQO_RqAddRequestToBatch
- PCQO_RsOpenFirstBatchRecord
- PCQO_RsOpenNextBatchRecord
- PCQO_SClearBatchQueues(Type)

The general workflow of a batch handling script can be described as such:

- 1) Create a request (Create, Query, Update, or Delete)
- 2) Populate the request as needed
- 3) Add the request to the batch
- 4) Repeat steps 1-3 as often as required
- 5) Execute
- 6) Open the first batch response record
- 7) Open the first response record (if expecting more than one records, such as as the result of a query)
- 8) Process the response record
- 9) Open the next response record; repeat 7-9 as often as necessary
- 10) Open the next batch response record
- 11) Repeat steps 6-10 as often as necessary
- 12) Clear the request and response queues
- 13) End of Workflow

For an example of a batch process, below is a sample script demonstrating the use of performing a mix of adding and updating a set of Customer records.

```
Allow User Abort [Off]
Set Error Capture [On]

# Verify the plug-in is ready to function
Perform Script ["Plug-in Checker"]

# Start batch loop; we assume we are in the context of Customers
Go to Record/Request [First]
Loop
  If [not IsEmpty( Customers::QBOId )]
    Set Variable [$result; Value: PCQO_RqNew( "Update" ; "Customer" )]
    Set Variable [$result; Value: PCQO_RqAddFieldWithValue( "Id" ;
Customers::QBOId )]
    Set Variable [$result; Value: PCQO_RqAddFieldWithValue( "SyncToken" ;
Customers::SyncToken )]
  Else
    Set Variable [$result; Value: PCQO_RqNew( "Create" ; "Customer" )]
    Set Variable [$result; Value: PCQO_RqAddFieldWithValue( "DisplayName" ;
Customers::DisplayName )]
  End If

  Set Variable [$result; Value: PCQO_RqAddFieldWithValue( "CompanyName" ;
Customers::CompanyName )]
  Set Variable [$result; Value: PCQO_RqAddFieldWithValue( "GivenName" ;
Customers::FirstName )]
  Set Variable [$result; Value: PCQO_RqAddFieldWithValue( "FamilyName" ;
Customers::LastName )]

  # Add the request to the request batch
  Set Variable [$result; Value: PCQO_RqAddRequestToBatch]

  Go to Record/Request [Next; Exit After Last:On]
End Loop

# Batch is complete, so execute the batch requests (We assume a session is active)
Set Variable [$result; Value: PCQO_RqExecute]

# Loop over the batch responses
Set Variable [$result; Value: PCQO_RsOpenFirstBatchRecord]

Loop
  Exit Loop If [$result = "END" or $result = "!!ERROR!!"]
```

```

# When loading a response record from the batch, the status results
# of that response are also loaded, so PCQO_SGetStatus will reflect
# whether the request was successful or not for each response.
If [PatternCount( PCQO_SGetStatus ) ; "Success" ] > 0]
    # Process successful response
    Set Variable [$res; Value: PCQO_RsOpenFirstRecord]

    Set Variable [$qbID; Value: PCQO_RsGetFirstFieldValue( "Id" )]
    # Find Criteria: Find all where Customers::QBOId == $qbID
    Perform Find [Restore]

    # Since these are all single-record responses to Add or Mod requests,
    # we can simply store the QB list information and move on to the next
    # record.
    Set Field [Customers::QBOId; PCQO_RsGetFirstFieldValue( "Id" )]
    Set Field [Customers::SyncToken; PCQO_RsGetFirstFieldValue( "SyncToken" )]
Else
    # Process error response
    Set Field [Customers::QBError; PCQO_SGetStatus]
End If

Set Variable [$result; Value: PCQO_RsOpenNextBatchRecord]
End Loop

# After all the batching has been done, we need to "clean up" the batch queues,
# so that the requests and responses are cleared out and won't be included in the
# next batch.
Set Variable [$result; Value: PCQO_SClearBatchQueues( "Both" )]

```

As demonstrated in the script above, the batch handling parts wrap around standard request/response script, where each request is added to the batch, and each response is pulled from the batch. It is also important to point out that as each batch response record is accessed, the internal status values for the response record are updated to reflect the success or failure status of the record itself. In other words, if a successful batch response record is accessed, PCQO_SGetStatus will return a "Success" or "0" status indicator; if a failed batch response record is accessed, however, PCQO_SGetStatus will return the error that QuickBooks Online assigned to that particular record, such as a validation fault, a missing value, or simply a notice that no records are found.

Working with Attachable Files

Some situations can arise where a business user would need to attach a document to a transaction record, such as a scanned image of a receipt or a signed PDF document for a work order. As of version 2.0.5.0, the FM Books Connector Online plug-in is now able to upload and download these documents, or "Attachable Files", to and from QuickBooks Online.

When working with attachable files, the PCQO_SUploadAttachableFile function will upload the attachable file to QuickBooks Online for a specified transaction entity from a file path or container field, while PCQO_SDownloadAttachableFile will download the specified attachable file to a set file path or to a container field. Attachable files can be provided either through container fields as uncompressed embedded binary data (as opposed to compressed embedded binary data or files stored as a reference), or as a file path reference to the file located on a mapped storage device that the local machine can access.

During the course of uploading an attachment file, the plug-in will load the file reference or file data into memory, construct a packet of metadata containing details of the attachable file (as per the Attachable entity reference in the Accounting API), and finally upload the file to QuickBooks Online. Attachable files must have a valid entity type and entity ID to be uploaded. These entity types can be transaction entities such as Invoices and Bills, and the ID is a valid QuickBooks Online ID corresponding with the desired entity. In QuickBooks Online, you can view any attached files by reviewing the entity details page (e.g. viewing the Invoice details page if you attached a file to an invoice) and locating the attached file in a portal at the bottom left. See screenshot below for an example of the location.

Invoice #764
⚙️ ? ✕

BALANCE DUE

\$150.00

Send later
 Co/Bcc

Last Delivery: Sent by email to chris.turner@productivecomputing.com at Sep 12, 10:59 am Pacific Daylight Time

Billing address

Nina Landon
 Emerging Company
 Emerging Company
 ATTN: Nina Landon
 123 Centered Street

Terms

Invoice date

Due date

Location

Shipping address

Ship via

Shipping date

Tracking no.

Crew #

Sales Rep

Subtotal \$150.00

Taxable subtotal \$0.00

Discount percent \$0.00

Shipping \$0.00

Total **\$150.00**

Balance due **\$150.00**

#	PRODUCT/SERVICE	SKU	DESCRIPTION	QTY	RATE	AMOUNT	TAX	CLASS
1	Concrete		Description of the first line item goes here (More description here)	1	150	150.00		
2								

Subtotal \$150.00

Message displayed on invoice

This is a test customer memo

Message displayed on statement

This is a test statement memo

Attachments Maximum size: 20MB

Attach to email LICENSE.txt (2) ✕
Drag/Drop files here or click the icon

[Show existing](#)

[Privacy](#)

Attachable files can also be downloaded from QuickBooks Online. The first step in downloading a file would be to pull down the attachable file's reference, which can be accomplished with a standard query on the Attachable entity. If you know what the transaction ID that the attachable file has been linked to is, you can provide that in a query string parameter as follows:

```
PCQO_RqAddFieldWithValue( "QueryString" ; "Select * From Attachable Where AttachableRef.EntityRef.value =
"& $Id & ""and AttachableRef.EntityRef.type = ""& $entityType & """)
```

The above query string will return all details of every attachable object that belongs to the \$entityType with an ID of \$Id. If \$entityType is "Invoice" and \$id is "200", then it will return all attachments belonging to Invoice with the QBO ID of 200.

For examples of attaching files to transactions in QuickBooks Online, please see the script below that will attach a container field's contents, whether stored by reference or as embedded binary data, to a newly-created Invoice record.

```
Allow User Abort [Off]
Set Error Capture [On]

# Verify the plug-in is ready to function
Perform Script ["Plug-in Checker"]

# Start batch loop; we assume we are in the context of Invoices
Set Variable [$result; Value: PCQO_RqNew( "Create" ; "Invoice" )]
Set Variable [$result; Value: PCQO_RqAddFieldWithValue( "CustomerRef::value" ;
invoice_Customer::QBOId )]

# Populate additional invoice information here...
# ...
# ...

# Execute the request to add the invoice
Set Variable [$result; Value: PCQO_RqExecute]

# Process the response
If [$result = 0]
    Set Variable [$res; Value: PCQO_RsOpenFirstRecord]

    Set Field [Invoice::QBOId; PCQO_RsGetFirstFieldValue( "Id" )]

    # Invoice was created, so now we attach the file.
    If [GetContainerAttribute( Invoice::AttachableFile ; "storageType" ) = "File
Reference"]
        # File is stored as reference, so pass file path through.
        Set Variable [$path; Value: GetValue( Invoice::AttachableFile ; 2 )]
        # Format $path to a valid system-formatted value, either for Mac or
Windows...

        Set Variable [$result; Value: PCQO_SUploadAttachableFile( "Invoice" ;
Invoice::QBOId ; $path ; False ; GetContainerAttribute(
Invoice::AttachableFile ; "filename" ) )]
    Else
        # File is attached as embedded data, so pass that through.
        Set Variable [$result; Value: PCQO_SUploadAttachableFile( "Invoice" ;
Invoice::QBOId ; Invoice::AttachableFile ; True ; GetContainerAttribute(
Invoice::AttachableFile ; "filename" ) )]
    End If
```

```

    If [$result = 0]
        # File attached successfully
    Else
        # Failed to attach file
    End If
Else
    # Process error response
    Set Field [Invoice::QBError; PCQO_SGetStatus]
End If

```

In the above script example, we assume there is only one container field called "AttachableFile" in the Invoice table that contains either a reference or an embedded file, and this file is passed on through to QuickBooks Online via the PCQO_SUploadAttachableFile function. It could be further expanded to support multiple files; in those cases, the developer would be prudent to set up a portal of related attachable files connecting to the invoice and iterating over the set to attach each one to the invoice.

If users perform data entry within QuickBooks itself, it is possible that an attachable file can be linked to more than one entity. If this is the case, then the developer must be able to adjust their scripting to handle pulling down attachables that reference multiple transactions. The plug-in will still only upload attachments on a one-for-one basis, however, so any attachable file uploaded by the plug-in will only link to a single transaction entity.

Attachable Notes

Not all attachable objects are files, however. Some attachable objects in QuickBooks Online are simply notes, or large text blobs that describe information about the transaction as defined by the user. For these, the developer would not need to use the PCQO_SUploadAttachableFile function to create the attachable note; instead, the standard request-response process would be suitable. Take the following pseudoscript for example:

```

PCQO_RqNew( "Create" ; "Attachable" )
PCQO_RqAddFieldWithValue( "Note" ; "This is an attached note. It can have up to 2000
characters in it." )
PCQO_RqAddRelatedRecord( "AttachableRef" )
PCQO_RqAddFieldWithValue( "EntityRef::value" ; "95" )
PCQO_RqAddFieldWithValue( "EntityRef::type" ; "Invoice" )
PCQO_RqCloseRelatedRecord
PCQO_RqExecute

```

This pseudoscript will create a note with the specified text and associate it to the invoice with an internal ID of 95. Querying for attachable notes follows the same recommendations as querying for attachable files, with the key distinction being that attachable notes do not contain a "FileName" field value, and attachable files do not contain a "Notes" field value.

Additional Fields

The following fields are not defined in the ESR but are acceptable fields for the PCQO_RqAddFieldWithValue function for query operations.

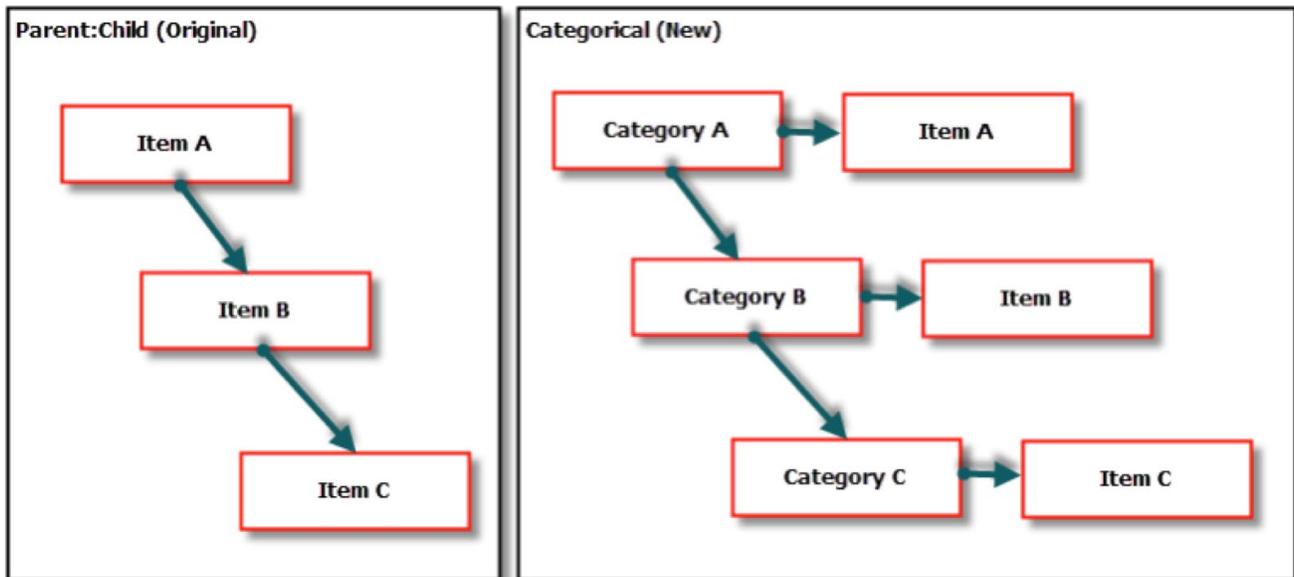
Module	Method	Field Name	Acceptable Values/Notes
Query	Set	MaxResults	Any integer value greater than 0. Default value is the number of the corresponding entity in QuickBooks Online, or 500 entities, whichever is less.
Query	Set	StartPosition	Any integer value greater than 0. Default value is 1.
Query	Set	QueryString	Any SQL Select statement that follows Intuit's SQL Syntax rules. The SQL Select syntax can be found at the following link: https://help.developer.intuit.com/s/article/SQL-Queries
Query	Get	CountResult	Returns the number of entities counted by a "count(*)" query operation
Update	Set	SparseUpdate	Determines whether the update will be a "Complete" or "Sparse" update. True or False. Default value is true.

Item Hierarchy Change

Intuit recently updated the handling of items in new and existing QuickBooks Online companies that alters how items are categorized, splitting Items into true items and Item Categories.

In a nutshell, categories are descriptive entries describing what an item is part of, such as "Running Shoes", "Sales Tax", or "Labor Services". True items are the actual inventory, non-inventory, service, or other items that are still in place. The major difference between the new setup for item hierarchies and the old setup is that a true item cannot be the parent of another true item; only a category can be the parent.

To illustrate this, please reference the following graphic:



In the left setup ("Parent:Child"), items can be direct parents of each other. If you wanted to designate a reference to Item B, you would use the Item FullName of "Item A:Item B".

In the right setup ("Categorical"), only the categories can be parents of each other, and also can be parents of the items; there are no direct item-to-item relationships. If you wanted to designate a reference to Item B, you would use the Item FullName of "Category A:Category B:Item B".

Of important note is that Categories cannot be used as items, for example in an invoice line item. They can be used as part of the full name for a true item but specifying them as the true item will return an error from QuickBooks Online.

Further information can be found in detail at

<https://developer.intuit.com/app/developer/qbo/docs/workflows/item-hierarchies-using-categories>.

Change Data Capture (CDC)

In FM Books Connector Online version 3.0.1.0 (Windows) / 3.0.1.1 (Mac), the entity known as "Change Data Capture" is now available for Query operations. This entity contains a change log, or a JSON document containing a list of all changed entities, for a set of entities from a given point in time. Entities returned in this JSON document are full objects, containing every piece of information about the entity in question. If multiple entities were specified in the Query request, the response document will split the returned entities grouped by their type, in order of last updated time within the group. More information on Change Data Capture operations and their contents can be found in the Accounting API reference.

The only acceptable fields for a Change Data Capture request are "EntityList" and "ChangedSince". EntityList should be a comma-separated list of one or more QBO entities (e.g. "customer,invoice,payments") that match the name of the Entity as you would specify for any other request. ChangedSince should be a standard timestamp of the point in time after which the changes should be logged. Date fields are also accepted; they will default to midnight of the date specified (e.g. specifying "12/31/2021" will default to "12/31/2021 12:00:00 AM").

Below is a sample script of how to perform a Change Data Capture request. **Due to the potential grouping and complex nature of the JSON response document, the FM Books Connector online plug-in will not process the response document via the Rs functions; if you wish to programmatically process the document, we highly recommend utilizing the native FileMaker JSON functions available in FileMaker 18 and on.**

```
# Assuming we have run the Plug-in Checker and verified we are connected to a Company..

# Initiate the request
Set Variable [$result; Value: PCQO_RqNew( "Query" ; "ChangeDataCapture" )]

# Specify the EntityList and ChangedSince parameters
# For this demonstration, we assume the $entityList is a comma-separated list
# consisting of "customer,invoice" and the $changedSince date is January 1st, 2022.
# This request should return a list of all changed Customers or Invoices that were
# created or altered since January 1st, 2022.

Set Variable [$result; Value: PCQO_RqAddfieldWithValue( "EntityList" ; $entityList )]
Set Variable [$result; Value: PCQO_RqAddfieldWithValue( "ChangedSince" ; $changedSince)]

# Perform the request
Set Variable [$result; Value: PCQO_RqExecute]
If [$result <> 0]
    # Error handling...
Else
    # Create a new record in a "ChangeLog" table, populate it with the changed data
    # JSON document, as well as the list of entities requested and what date the
    # changes were made since.

    New Record/Request
    Set Field [ChangeLog::Log; JSONFormatElements ( PCQO_SGetJSON( "Response" ) )]
    Set Field [ChangeLog::ChangedEntities; $entityList]
    Set Field [ChangeLog::ChangedSince; $changedSince]
End If
```

JSON Handling

The FM Books Connector Online plug-in now stores its internal data for Requests and Responses using the JSON data format (further information on the JSON format can be found here: <http://www.w3schools.com/json>). This format is more user-friendly than XML and has the added benefit of being the sample data format as seen in the Accounting API Reference on Intuit's QuickBooks Online site. The plug-in is still able to load XML documents into the Request or Response object, however, this is provided for the benefit of FM Books Connector Online developers who are more familiar with XML.

New functions have been defined for the FM Books Connector Online plug-in below:

- PCQO_RqUseXML(XML)

- Loads the XML document "XML" into the plug-in, converting it to an appropriate JSON document for use by the plug-in. The document is stored as a Request. If the XML is malformed or contains some error, the function will return "!!ERROR!!"; otherwise, it will return "0" on success.

- PCQO_RqUseJSON(JSON)

- Loads the JSON document "JSON" into the plug-in. This document is stored as a Request. If the JSON is malformed or contains some error, the function will return "!!ERROR!!"; otherwise, it will return "0" on success.

- PCQO_RsUseXML(XML)

- Loads the XML document "XML" into the plug-in, converting it to an appropriate JSON document for use by the plug-in. The document is stored as a Response. If the XML is malformed or contains some error, the function will return "!!ERROR!!"; otherwise, it will return "0" on success.

- PCQO_RsUseJSON(JSON)

- Loads the JSON document "JSON" into the plug-in. This document is stored as a Response. If the JSON is malformed or contains some error, the function will return "!!ERROR!!"; otherwise, it will return "0" on success.

- PCQO_SGetJSON(Type)

- Returns the Request or Response object, specified by "Type", as a JSON string, if the object exists.

- PCQO_SGetXML(Type)

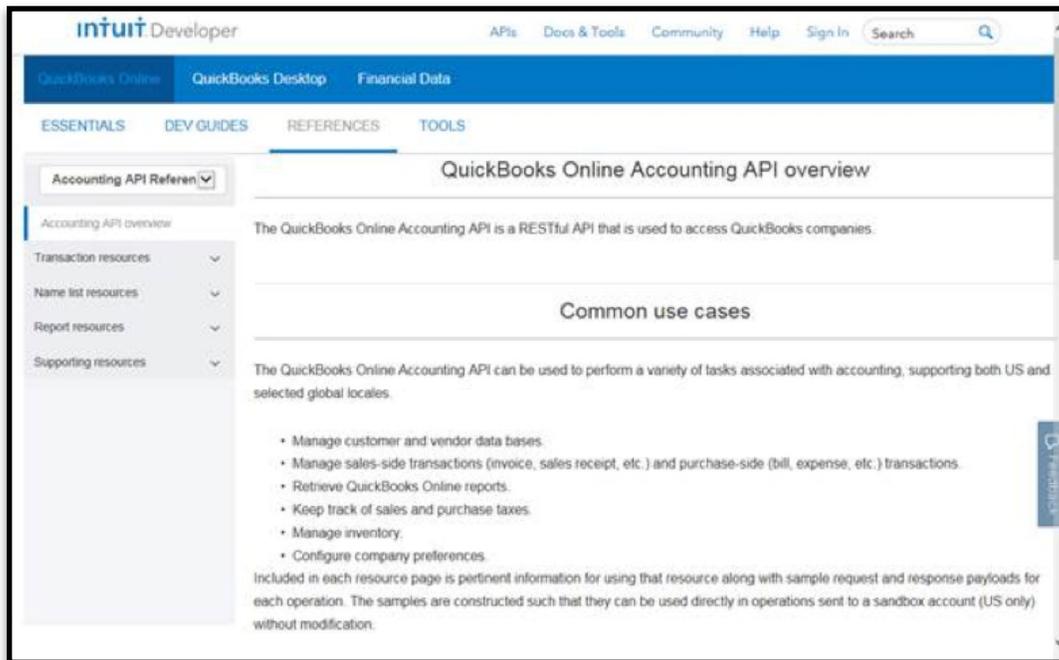
- This function is deprecated and will return a JSON string. Please use "PCQO_SGetJSON" instead.

11) QuickBooks Online Accounting API Reference

Now that you thoroughly understand how FileMaker and QuickBooks Online “talk” to each other by making requests and responses, we are ready to introduce Intuit’s Accounting API Reference guide. The Accounting API Reference contains a complete list of all available fields/filters, detailed descriptions of the request entity types, errors, and specifies field constraints and requirements. This will be crucial during development.

The Accounting API can be found at the following link:

<https://developer.intuit.com/docs/api/accounting>



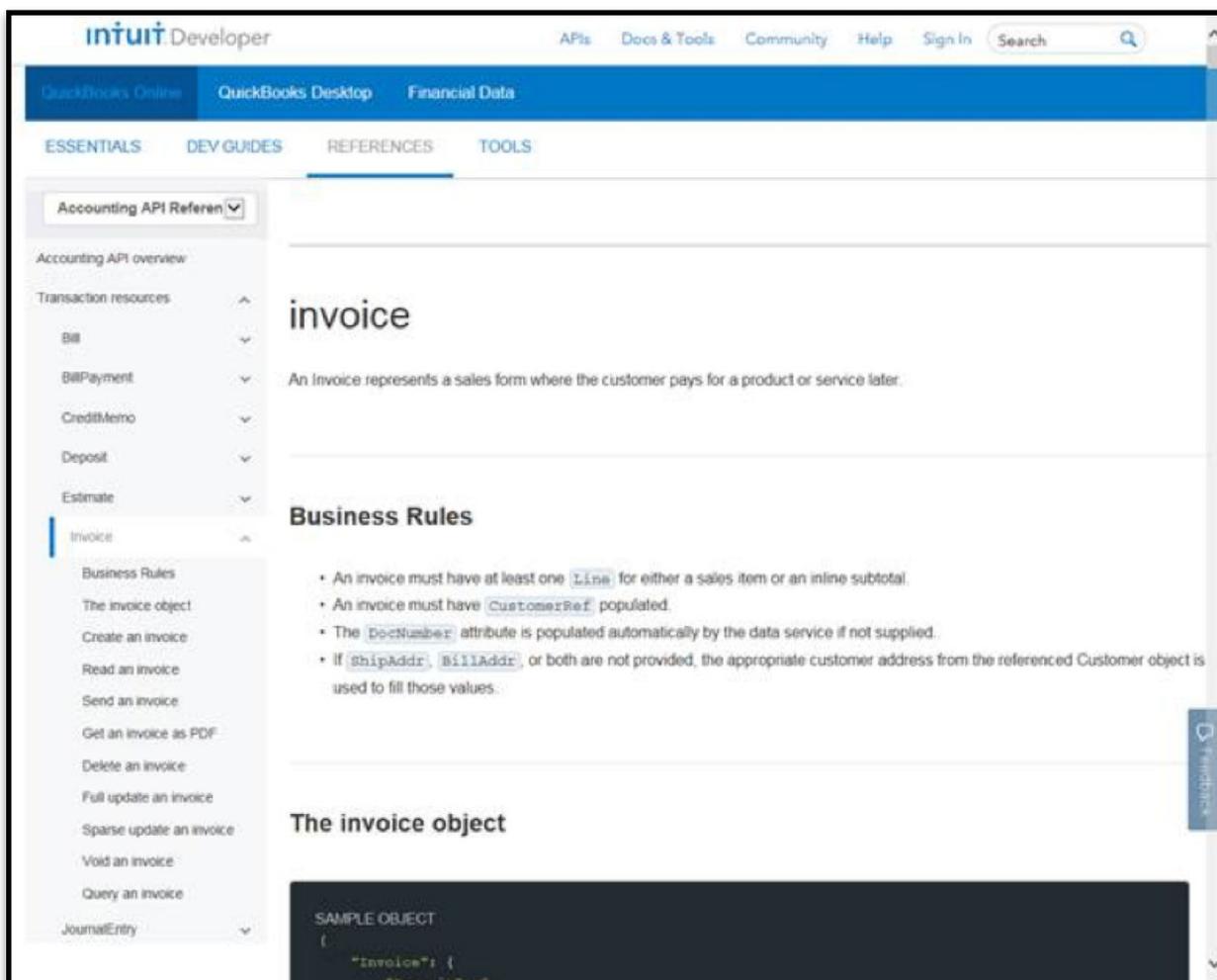
Using the Accounting API Reference

The Accounting API Reference provides details for all supported entities in QuickBooks Online. Details of an entity provide a description, supported operations, any applicable business rules and attributes of the entity. Included in the resources pages are examples of the supported operations of that entity written in JSON requests and responses.

NOTE: The FM Books Connector Online plug-in uses JSON data format for requests and responses, accessible by using the PCQO_Rq and PCQO_Rs functions. When building requests and processing responses, use the entity attributes (fields) by name in all function calls to PCQO_RqAddFieldWithValue and PCQO_RsGetFirstFieldValue.

Since you already understand what Requests and Responses are, select the appropriate resource to begin. The resource defines the class of records we want to work with and which action we want applied to those records. In our example below, we select the resource "Invoice" to view its business rules, supported operations, and attributes.

Accounting API Reference Example:



The screenshot shows the Intuit Developer website's Accounting API Reference page for the 'invoice' resource. The page is titled 'invoice' and includes a description: 'An invoice represents a sales form where the customer pays for a product or service later.' Below the description, there is a section for 'Business Rules' with four bullet points: 'An invoice must have at least one Line for either a sales item or an inline subtotal.', 'An invoice must have CustomerRef populated.', 'The DocNumber attribute is populated automatically by the data service if not supplied.', and 'If ShipAddr, BillAddr, or both are not provided, the appropriate customer address from the referenced Customer object is used to fill those values.' The page also features a section for 'The invoice object' which includes a 'SAMPLE OBJECT' in JSON format. The left sidebar contains a navigation menu with categories like 'Accounting API overview', 'Transaction resources', and 'Invoice', with 'Invoice' currently selected. The top navigation bar includes links for 'APIs', 'Docs & Tools', 'Community', 'Help', 'Sign In', and a search bar.

Applicable operations are listed under the expanded Resources > (Type) dropdown on the left (in the above screenshot, under the "Invoice" section that is highlighted with the blue bar marker). These operations are "Create" ("Create an invoice"), "Read" ("Read an invoice"), "Update" ("Full update an invoice" or "Sparse update an invoice"), "Delete" ("Delete an invoice"), or "Query" ("Query an invoice"). Not all operations are supported; for example, "Send an invoice", "Get an invoice as PDF", and "Void an invoice" in the example above are not fully supported by the FM Books Connector Online plug-in.

When updating a record, the options are to either fully update the record ("Full update an invoice"), which will completely replace the matching record in QuickBooks Online with the record provided by the plug-in, or sparse-update the record ("Sparse update an invoice"), which will only replace data in the record in QuickBooks Online with what fields are defined in the request from the plug-in. The "sparse" tag can be set by using "PCQO_RqAddFieldWithValue("sparse" ; "true" or "false"), and only truly fits during an "Update" operation.

Business Rules define special conditions for a given entity that must be met when posing requests to QuickBooks Online. Query operations, in general, do not need to consider business rules in their request, as they are merely posing queries for entities matching their criteria.

Accounting API Reference – Entity Attributes Example:

The screenshot shows the Intuit Developer Accounting API Reference page for the Invoice entity. The page is titled "intuit Developer" and has a navigation bar with "APIs", "Docs & Tools", "Community", "Help", "Sign In", and a search bar. Below the navigation bar, there are tabs for "QuickBooks Online", "QuickBooks Desktop", and "Financial Data". The main content area is divided into "ESSENTIALS", "DEV GUIDES", "REFERENCES", and "TOOLS". The "REFERENCES" tab is selected, and the "Accounting API Reference" dropdown is open, showing a list of entities including "Invoice". The "Invoice" entity is selected, and its attributes are displayed:

- Id:** String, filterable, sortable
required for update: Unique identifier for this object. Sort order is ASC by default.
- SyncToken:** String
required for update: Version number of the object. It is used to lock an object for use by one app at a time. As soon as an application modifies an object, its `SyncToken` is incremented. Attempts to modify an object specifying an older `SyncToken` fails. Only the latest version of the object is maintained by QuickBooks Online.
- MetaData:** ModificationMetaData
Descriptive information about the object. The MetaData values are set by Data Services and are read only for all applications.
↳ Child attributes
- CustomField [0..3]:** CustomField
Custom field or data extension.
↳ Child attributes
- DocNumber:** String, maximum of 21 chars, filterable, sortable
Reference number for the transaction. If not explicitly provided at create time, this field is populated based on the setting of `Preferences:CustomTxnNumber` as follows:
 - If `Preferences:CustomTxnNumber` is true a custom value can be provided. If no value is supplied, the resulting DocNumber is null.
 - If `Preferences:CustomTxnNumber` is false, resulting DocNumber is system generated by incrementing the last number by 1.
 If `Preferences:CustomTxnNumber` is false and a value is supplied, that value is stored even if it is a duplicate. Recommended best practice: check the setting of

Attributes are the "fields" of an entity, and have the following properties:

- Name of the attribute ("Id")
- Caveat for the attribute ("required for update")
- Number of appearances for the attribute ("CustomField [0..3]" means that the CustomField attribute can appear at least 0 times and as many as 3 times in a request/response)
- Type of attribute ("String", meaning text, or "CustomField", being a complex object type)
- Features of the attribute ("filterable, sortable")
 - o Filterable means the attribute can be used in the "WHERE" clause of a query to constrain what entities are returned for the query
 - o Sortable means the attribute can be used in the "ORDER BY" clause of a query to sort how the entities are returned for the query
- Description of the attribute ("Unique identifier for this object. Sort order is ASC by default.")
 - o Also can contain an expansion of child attributes, if the attribute is a complex type, or caveats/considerations/acceptable values for the attribute.

Accounting API Reference – Line Handling

The screenshot shows the Intuit Developer Accounting API Reference page. The main content area displays a JSON response for an invoice. A red box highlights the first line item, which is divided into two parts: the 'Line Header' and the 'Line Detail'. The 'Line Header' includes fields like 'Id', 'LineNum', 'Description', 'Amount', and 'DetailType'. The 'Line Detail' includes fields like 'ItemRef', 'UnitPrice', 'Qty', and 'TaxCodeRef'. Red arrows point from labels 'Line Header' and 'Line Detail' to their respective sections in the JSON.

```
    "Line": {
      "Id": "1",
      "LineNum": 1,
      "Description": "Rock Fountain",
      "Amount": 275.0,
      "DetailType": "SalesItemLineDetail",
      "SalesItemLineDetail": {
        "ItemRef": {
          "value": "75",
          "name": "Rock Fountain"
        },
        "UnitPrice": 275,
        "Qty": 1,
        "TaxCodeRef": {
          "value": "TAX"
        }
      }
    }, {
      "Id": "2",
      "LineNum": 2,
      "Description": "Fountain Pump",
      "Amount": 12.75,
      "DetailType": "SalesItemLineDetail",
      "SalesItemLineDetail": {
        "ItemRef": {
          "value": "11",
          "name": "Fountain Pump"
        }
      }
    }
  ]
}
```

Special consideration should be taken for the "Line" data types, found in Transaction entities. Line types are broken up into two portions: the "Header", which holds generic information about the Line, and the "Line Detail", which holds specific information that relates to the line as defined by the header's DetailType attribute. Please see the example script for processing a "Query Invoice" request to see how Invoice Lines are handled.

Accounting API Reference – Line Header

Line [0..n]: Line
 required | Individual line items of a transaction. Valid `Line` types include:
 Sales item line:

▼ Child attributes

Id: String
 required for update | The id of the line item. Its use in requests is as follows:

- If `Id` is greater than zero and exists for the company, the request is considered an update operation for a line item.
- If no `Id` is provided, the `Id` provided is less than or equal to zero, or the `Id` provided is greater than zero and does not exist for the company then the request is considered a create operation for a line item.

Available in all objects that use lines and support the update operation.

LineNum: Decimal, positive integer
 Specifies the position of the line in the collection of transaction lines.

Description: String, max 4000 chars
 Free form text description of the line item that appears in the printed record.

Amount: Decimal, max 15 digits in 10.5 format
 required | The amount of the line item.

DetailType: LineDetailTypeEnum
 required | Set to `SalesItemLineDetail` for this type of line.

SalesItemLineDetail: SalesItemLineDetail

▼ Child attributes

ItemRef: ReferenceType
 Reference to an Item object.

Some data types have their own attributes just like the standard entities and as they are not stand-alone entities in their own right, they should be handled as related records to a main entity (such as an Invoice). In the example above, a Line's Header section has the attributes "Id", "LineNum", "Description", "Amount", and "DetailType". Line Details are provided along with a corresponding DetailType. In the example above, the DetailType field's value is "SalesItemLineDetail" for a Sales item line.

12) Supported and Unsupported areas using QBO

There are certain areas that QuickBooks Online does not have or provide support for third-party applications. The most obvious of these areas are listed below:

- Payroll
- Templates

Available features and functionality will vary depending on your QuickBooks Online service plan subscription.

See list below as an example (QuickBooks plans and features may change over time):

Functionality	Simple Start	Essentials	Plus
Track your income and expenses	x	x	x
Send unlimited estimates and invoices	x	x	x
Download transactions from your bank and credit card accounts ¹	x	x	x
Print checks and record transactions	x	x	x
Import data from Excel ² or QuickBooks desktop ³	x	x	x
Back up your data online automatically ⁴	x	x	x
Same security and encryption as banks ⁵	x	x	x
Access your data from a tablet or smartphone ⁶	x	x	x
Invite up to two accountants to access your data ⁷	x	x	x
Integrate with available applications ⁸	x	x	x
Set up invoices to automatically bill on a recurring schedule		x	x
Manage and pay bills from vendors		x	x
Enter bills and schedule payments for later		x	x
Compare your sales and profitability with industry trends		x	x
Control what your users can access		x	x
Create and send purchase orders			x
Track inventory			x
Prepare and print 1099s			x
Give employees and subcontractors limited access to enter time worked			x
Track billable hours by customer			x
Create budgets to estimate future income and expenses			x
Categorize your income and expenses using class tracking			x
Track sales and profitability for each of your locations			x
Number of people who can simultaneously use QuickBooks Online	1	3	5
Number of built-in business reports	20+	40+	65+

13) Entity Query Support

Each entity has its own "Query" support, specified by the Query operation. By default, when receiving an entity query request, QuickBooks Online will return no more than 500 matching entities. When setting up query requests using the FM Books Connector Online, however, this number of entities to be returned can be specified by defining the "MaxResults" field in the request. If this is omitted, then the request will return as many entities as there are in the QBO company, or 500, whichever is less.

Some example entities that can be queried are:

- Customer
- Invoice
- Payment
- CompanyInfo
- Purchase

Example:

If we wish to query for all invoices, but only wish to receive 10, then we can structure a query as:

```
Set Variable [$result, Value:PCQO_RqNew( "Query" ; "Invoice" )]  
Set Variable [$result, Value:PCQO_RqAddFieldWithValue( "MaxResults" ; "10" )]  
Set Variable [$result, Value:PCQO_RqExecute]
```

The use of the MaxResults field also ties into iterator support. Using the field name of "StartPosition", a subset of entities that match a query's response can be accessed, based on what position they reside in within the total query results.

Example:

Assuming that the previous example returned more than 10 Invoice entities, accessing the next ten Invoice entities can be structured as:

```
Set Variable [$result, Value:PCQO_RqNew( "Query" ; "Invoice" )]  
Set Variable [$result, Value:PCQO_RqAddFieldWithValue( "MaxResults" ; "10" )]  
Set Variable [$result, Value:PCQO_RqAddFieldWithValue( "StartPosition" ; "11" )]  
Set Variable [$result, Value:PCQO_RqExecute]
```

Simple Query Language

Instead of posing a query using the individual field/attribute names as qualifiers for the query, setting the field "QueryString" will allow a SQL-style query string to be considered as the query for an entity. This string follows the Intuit format of SQL Select statements, defined at the link below.

<https://help.developer.intuit.com/s/article/SQL-Queries>

The supporting SQL objects "MaxResults" and "StartPosition" are automatically included in the query if they are not defined within the QueryString value.

Sample QueryString values:

```
SELECT * FROM Invoice WHERE Id = '155'
```

Returns all attributes for an Invoice where the Id is 155

```
SELECT GivenName, FamilyName, CompanyName FROM Customer STARTPOSITION 1 MAXRESULTS 20
```

Returns the GivenName, FamilyName, and CompanyName for the first 20 Customer entities

```
SELECT count(*) FROM Purchase WHERE PaymentType = 'Check' AND TotalAmt > '1000.0'
```

Returns the number of Purchases that are checks that cost more than \$1000.00

Counting Entities

In the above example we used a "count" operator in a query string. As of version 1.0.1.0, if your solution needs to know how many of a certain entity (Customer, Invoice, Item, etc.) exist in QuickBooks Online, there are two ways to accomplish this goal.

The first method involves using the `PCQO_SCountEntities(EntityType)` function. This will return a count of all entities of a given `EntityType` in QuickBooks Online. Note that it cannot be limited by selectors; it will count every record of the corresponding type and return that value.

```
# Set up the request
Set Variable [ $result, Value: PCQO_RqNew( "Query" ; "Item" ) ]
# Specify the query string - We want to know how many active items we have in QuickBooks
Online
Set Variable [ $result, Value: PCQO_RqAddFieldWithValue( "QueryString" ; "SELECT count(*)
FROM Item WHERE Active = true" ]
# Execute the request
Set Variable [ $result, Value: PCQO_RqExecute ]
#
# Error handling...
# ...
#
# Get the count
Set Variable [ $activeItemCount, Value: PCQO_RsGetFirstFieldValue( "CountResult" ) ]
```

The second method allows the use of selectors or filters. When specifying a query for an entity, using the "count(*)" operator in the query string will ensure that the system will acquire the count of all entities that meet the query's selection criteria. See the example script below for a demonstration of this.

The "CountResult" field is a field specific only to making a query that consists of the "count(*)" operator. Calling this field in any other query request will return an error. When using the "count(*)" operator in a query string, please ensure that the only selection criteria you specify is the count(*) operator. You cannot choose additional fields along with the count(*) operator (such as "SELECT count(*), Id, FullyQualified Name FROM ...").

14) Sample Scripts and Queries

We have prepared a number of sample scripts and queries in order to better illustrate the different ways the FM Books Connector Online can be used to integrate with QuickBooks Online.

Query for the Id and Name of any inventory items that contain the text “Package” anywhere in the SKU

When querying for items from QuickBooks Online, the developer can structure the query by adding different fields to the query request, or by specifying a query string. In this example, we’ll use a query string so that we can make use of the power of “fuzzy searches” allowed by Intuit’s SQL support.

```
# Perform the plug-in checker to ensure the plug-in is installed and registered
Perform Script ["Plug-in Checker"]
If [Get(ScriptResult) <> 0]
    # Plug-in is not installed or registered
    Show Custom Dialog ["Failed Plug-in Checker"; "Plug-in is not installed or
registered"]
    Exit Script [Text Result:]
End If

# Structure the inventory item query
Set Variable [$result; Value:PCQO_RqNew( "Query" ; "Item" )]
Set Variable [$queryString; Value:"Select Id, Name from Item Where Type =
`Inventory `And SKU like `%Package%`"]
Set Variable [$result; Value:PCQO_RqAddFieldWithValue( "QueryString" ; $queryString
)]

# Execute the request (This sample script assumes the session is already active)
Set Variable [$result; Value:PCQO_RqExecute]

If [$result <> 0]
    # Handle Error
    Show Custom Dialog ["Execution Error"; PCQO_SGetStatus]
    Exit Script [Text Result:]
End If

# Result is successful, process the response
Set Variable [$result; Value:PCQO_RsOpenFirstRecord]
Loop
    Exit Loop If [$result = "!!ERROR!!" or $result = "END" or $result = "?"]

    # Perform a subscript to handle the adding or updating of the record in
FileMaker
    Perform Script ["_Add/Update Item ( QBOId )";
Parameter:PCQO_RsGetFirstFieldValue( "Id" )]

    Set Variable [$result; Value:PCQO_RsOpenNextRecord]
End Loop
```

Pull and add active customers, 10 customers at a time

In some cases, the amount of records pulled in by the FM Books Connector Online plug-in is too vast; large quantities of data will slow down execution time, so using a technique called "pagination" can help limit the set of records, allowing the script to page through small sets of records and making the process more seamless.

```
# Perform the plug-in checker to ensure the plug-in is installed and registered
Perform Script ["Plug-in Checker"]
If [Get(ScriptResult) <> 0]
    # Plug-in is not installed or registered
    Show Custom Dialog ["Failed Plug-in Checker"; "Plug-in is not installed or
registered"]
    Exit Script [Text Result:]
End If

# Set up Pagination Variable(s)
Set Variable [$startPosition; Value:1]
Set Variable [$maxResults; Value:10]

# Gather the maximum number of records our query would find in QuickBooks Online
# This is optional; performing this step makes the looping logic easier later in
the
# script.
Set Variable [$result; Value:PCQO_RqNew( "Query" ; "Customer")]
Set Variable [$result; Value:PCQO_RqAddFieldWithValue( "QueryString" ; "Select
count(*) From Customer Where Active = true")]
Set Variable [$result; Value:PCQO_RqExecute]

If [$result <> 0]
    # Handle Error
    Show Custom Dialog ["Execution Error"; PCQO_SGetStatus]
    Exit Script [Text Result:]
Else
    # Get the count result
    Set Variable [$maxCustomers; Value:PCQO_RsGetFirstFieldValue("CountResult")]
End If

# We're performing a loop within a loop; main loop is over the entirety of the
request
# (we want all Active customers), inner loop iterates only over 10 records at a
time
Loop

    # Structure the query ("Select all active customers, 10 at a time")
    Set Variable [$result; Value:PCQO_RqNew( "Query" ; "Customer" )]
    Set Variable [$result; Value:PCQO_RqAddFieldWithValue( "Active" ; "true" )]

    # Add the Start Position, which always advances with each pass; each pass
should be
    # equal to $maxResults + 1 + 10 per pass (1, 11, 21, 31, etc.)
    Set Variable [$result; Value:PCQO_RqAddFieldWithValue( "StartPosition" ;
$startPosition )]

    # Add the Max Results, which is always 10 in this example; we only want 10
records
    # per pass.
    Set Variable [$result; Value:PCQO_RqAddFieldWithValue( "MaxResults" ;
$maxResults )]

    # Execute the request (This sample script assumes the session is already
active)
    Set Variable [$result; Value:PCQO_RqExecute]

    If [$result <> 0]
        # Handle Error
        Show Custom Dialog ["Execution Error"; PCQO_SGetStatus]
```

```

    Exit Script [Text Result:]
End If

# Result is successful, process the response
Set Variable [$result; Value:PCQO_RsOpenFirstRecord]
Loop
    Exit Loop If [$result = "!!ERROR!!" or $result = "END" or $result = "?"]

    # Perform a subscript to handle the adding or updating of the record in
    FileMaker
    Perform Script ["_Add/Update Customer( QBOId )"];
    Parameter:PCQO_RsGetFirstFieldValue( "Id" )
    Set Variable [$startPosition; Value:$startPosition + 1]

    Set Variable [$result; Value:PCQO_RsOpenNextRecord]
End Loop

# Exit the loop if we've gathered the maximum number of customers
Exit Loop If [$startPosition >= $maxCustomers]

End Loop

```

Add a company currency entry for Japanese Yen (JPY), and set a customer's preferred currency to Japanese Yen

If the customer's company is set up to use multiple currency, the plug-in can add and update currency entries and link those currencies to customers that prefer to pay in other monetary forms. As a note, the default currency, or "Home Currency", is only configurable within the QuickBooks Online web interface; the plug-in is unable to change what the Home Currency is, or establish a currency as the home currency. Below is an example of creating a new company currency entry and setting a customer's preferred currency to it.

```

# Perform the plug-in checker to ensure the plug-in is installed and registered
Perform Script ["Plug-in Checker"]
If [Get(ScriptResult) <> 0]
    # Plug-in is not installed or registered
    Show Custom Dialog ["Failed Plug-in Checker"; "Plug-in is not installed or
    registered"]
    Exit Script [Text Result:]
End If

# Create the Company Currency add request
Set Variable [$result; Value:PCQO_RqNew( "Create" ; "CompanyCurrency" )]
Set Variable [$result; Value:PCQO_RqAddFieldWithValue( "Code" ; "JPY" )]
Set Variable [$result; Value:PCQO_RqAddFieldWithValue( "Name" ; "Japanese Yen" )]
Set Variable [$result; Value:PCQO_RqAddFieldWithValue( "Active" ; "true" )]

# Execute the request
Set Variable [$result; Value:PCQO_RqExecute]

If [$result <> 0]
    # Handle Error
    Show Custom Dialog ["Execution Error"; PCQO_SGetStatus]
    Exit Script [Text Result:]
End If

# Take the code ("JPY") and use it to sparse update a customer record (assuming the
# current context is on the customer)
Set Variable [$result; Value:PCQO_RqNew( "Update" ; "Customer" )]
Set Variable [$result; Value:PCQO_RqAddFieldWithValue( "SparseUpdate" ; "true" )]
Set Variable [$result; Value:PCQO_RqAddFieldWithValue( "Id" ; Customer::QBO_ID )]
Set Variable [$result; Value:PCQO_RqAddFieldWithValue( "SyncToken" ;
Customer::SyncToken )]

```

```

# Add the CurrencyRef element to assign the new preferred currency
Set Variable [$result; Value:PCQO_RqAddFieldWithValue( "CurrencyRef::value" ; "JPY"
)]

# Execute the request (This sample script assumes the session is already active)
Set Variable [$result; Value:PCQO_RqExecute]

If [$result <> 0]
  # Handle Error
  Show Custom Dialog ["Execution Error"; PCQO_SGetStatus]
  Exit Script [Text Result:]
End If

# Result is successful, store the new SyncToken
Set Variable [$result; Value:PCQO_RsOpenFirstRecord]
Set Field [Customer::SyncToken; Value:PCQO_RsGetFirstFieldValue( "SyncToken" )]

```

After the process runs, any invoices created for this customer will have its currency type coerced to Japanese Yen. For more information on managing multiple currencies, please refer to the Intuit Accounting API's "Managing multiple currencies" section at the following web address:

<https://developer.intuit.com/app/developer/qbo/docs/workflows/manage-multiple-currencies>

Apply custom fields to an invoice

A useful feature of QuickBooks is the ability to work with additional field data that doesn't fit the default fields available for certain transaction or name list entities. These "custom fields" are configured and set up in QuickBooks Online's online interface, and are accessible to add or update on transaction and name list entities via the FM Books Connector Online. If the developer wishes to note any available custom fields in a customer's solution, custom field information can be found by making a query to the Preferences entity and reading the CustomField list from the "SalesFormsPrefs" related element.

Below is an example of setting two custom fields named "Crew #" and "Sales Rep" for an invoice.

```
# Perform the plug-in checker to ensure the plug-in is installed and registered
Perform Script ["Plug-in Checker"]
If [Get(ScriptResult) <> 0]
    # Plug-in is not installed or registered
    Show Custom Dialog ["Failed Plug-in Checker"; "Plug-in is not installed or
registered"]
    Exit Script [Text Result:]
End If

# Create the base Invoice create request (For simplicity's sake, we're using local
# variables for field data)
Set Variable [$result; Value:PCQO_RqNew( "Create" ; "Invoice" )]
Set Variable [$result; Value:PCQO_RqAddFieldWithValue( "CustomerRef::value" ;
$customerID)]
Set Variable [$result; Value:PCQO_RqAddFieldWithValue( "TxnDate" ; $txnDate )]
Set Variable [$result; Value:PCQO_RqAddFieldWithValue( "DocNumber" ; $docNumber )]
Set Variable [$result; Value:PCQO_RqAddFieldWithValue( "DueDate" ; $txnDate + 30 )]
# Add Line information
Set Variable [$result; Value:PCQO_RqAddRelatedRecord( "Line" )]
Set Variable [$result; Value:PCQO_RqAddFieldWithValue( "Description" ; $description
)]
Set Variable [$result; Value:PCQO_RqAddFieldWithValue( "DetailType" ;
"SalesItemLineDetail" )]
Set Variable [$result; Value:PCQO_RqAddFieldWithValue( "Amount" ; $lineAmount )]
Set Variable [$result; Value:PCQO_RqAddRelatedRecord( "SalesItemLineDetail" )]
Set Variable [$result; Value:PCQO_RqAddFieldWithValue( "ItemRef::value" ; $itemID
)]
Set Variable [$result; Value:PCQO_RqAddFieldWithValue( "Qty" ; $qty )]
Set Variable [$result; Value:PCQO_RqAddFieldWithValue( "UnitPrice" ; $rate )]
Set Variable [$result; Value:PCQO_RqAddFieldWithValue( "TaxCodeRef::value" ; "NON"
)]
Set Variable [$result; Value:PCQO_RqCloseRelatedRecord // SalesItemLineDetail]
Set Variable [$result; Value:PCQO_RqCloseRelatedRecord // Line]

# Add the Custom Field Data
Set Variable [$result; Value:PCQO_RqAddRelatedRecord( "CustomField" )]
# For DefinitionId, this is the unique identifier belonging to the "Sales Order
Number"
# custom field. In this example, we'll assume the DefinitionId is "1".
Set Variable [$result; Value:PCQO_RqAddFieldWithValue( "DefinitionId" ; "1" )]
Set Variable [$result; Value:PCQO_RqAddFieldWithValue( "Name" ; "Crew #" )]
Set Variable [$result; Value:PCQO_RqAddFieldWithValue( "Type" ; "StringValue" )]
Set Variable [$result; Value:PCQO_RqAddFieldWithValue( "StringValue" ; "12345" )]
Set Variable [$result; Value:PCQO_RqCloseRelatedRecord // CustomField]

Set Variable [$result; Value:PCQO_RqAddRelatedRecord( "CustomField" )]
# For DefinitionId, this is the unique identifier belonging to the "Sales Rep"
# custom field. In this example, we'll assume the DefinitionId is "2".
Set Variable [$result; Value:PCQO_RqAddFieldWithValue( "DefinitionId" ; "2" )]
Set Variable [$result; Value:PCQO_RqAddFieldWithValue( "Name" ; "Sales Rep" )]
Set Variable [$result; Value:PCQO_RqAddFieldWithValue( "Type" ; "StringValue" )]
Set Variable [$result; Value:PCQO_RqAddFieldWithValue( "StringValue" ; "John Smith"
)]
Set Variable [$result; Value:PCQO_RqCloseRelatedRecord // CustomField]
```

```

# Execute the request
Set Variable [$result; Value:PCQO_RqExecute]

If [$result <> 0]
  # Handle Error
  Show Custom Dialog ["Execution Error"; PCQO_SGetStatus]
  Exit Script [Text Result:]
End If

# Result is successful, store the response data
Set Variable [$result; Value:PCQO_RsOpenFirstRecord]
Set Field [Main::gInvoice_TxnID; Value:PCQO_RsGetFirstFieldValue( "Id" )]
Show Custom Dialog ["Push Successful"; "Invoice has been pushed successfully to
QuickBooks Online."]

```

Please see below for an example on accessing the Custom Field definitions from the Preferences section:

```

# Perform the plug-in checker to ensure the plug-in is installed and registered
Perform Script ["Plug-in Checker"]
If [Get(ScriptResult) <> 0]
  # Plug-in is not installed or registered
  Show Custom Dialog ["Failed Plug-in Checker"; "Plug-in is not installed or
registered"]
  Exit Script [Text Result:]
End If

# Create the Preferences query request
Set Variable [$result; Value:PCQO_RqNew( "Query" ; "Preferences" )]

# Execute the request
Set Variable [$result; Value:PCQO_RqExecute]

If [$result <> 0]
  # Handle Error
  Show Custom Dialog ["Execution Error"; PCQO_SGetStatus]
  Exit Script [Text Result:]
End If

```

The resulting JSON response will contain all preference data, according to the Preferences object definition in the Accounting API documentation. An example of the response JSON can look like this:

```

[[
  "SalesFormsPrefs":
  {
    "ETransactionPaymentEnabled": false,
    "CustomField": [
      {
        "CustomField":[
          {
            "Name": "SalesFormsPrefs.UseSalesCustom3",
            "Type": "BooleanType",
            "BooleanValue": false
          },
          {
            "Name": "SalesFormsPrefs.UseSalesCustom2",
            "Type": "BooleanType",
            "BooleanValue": true
          },
          {
            "Name": "SalesFormsPrefs.UseSalesCustom1",
            "Type": "BooleanType",
            "BooleanValue": true
          }
        ]
      }
    ]
  },

```

```

{
  "CustomField": [
    {
      "Name": "SalesFormsPrefs.SalesCustomName1",
      "StringValue": "Crew #",
      "Type": "StringType"
    },
    {
      "Name": "SalesFormsPrefs.SalesCustomName2",
      "StringValue": "Sales Rep",
      "Type": "StringType"
    }
  ]
},
"IPNSupportEnabled": false,
"ETransactionAttachPDF": false,
"AllowDiscount": true,
...
...
...
}
}]

```

An important note to make is that the DefinitionId used in the transaction is derived from the numerical value at the end of the CustomField::Name element in the Preferences response. Additionally, there are multiple different kinds of custom fields available for different types of entities, and all such custom fields can be found in the Preferences query response. For further information on how to create, manage, and utilize custom fields, please refer to the Intuit Account API documentation at the link below, and the "Preferences" endpoint in the primary Accounting API overview:

<https://developer.intuit.com/app/developer/qbo/docs/workflows/create-custom-fields>

15) Error Handling

We find that most developers run into issues due to a lack of error trapping. Please ensure that you properly trap for errors in your solutions.

Typically the plug-in functions will return a 0 for a success. However, any of the plug-in functions may encounter an error during processing and will return the !!ERROR!! string. When an !!ERROR!! occurs during processing, immediately call the PCQO_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.

This makes it simple to check for errors. If a plug-in function does not return a 0 or returns !!ERROR!!, then immediately after call PCQO_SGetStatus function for a detailed description of what the exact error was. Here are a few samples of how you can check for errors.

Example 1:

```
Set Variable [ $result = PCQO_SomePluginFunction( "a" ; "b" ) ]
If [ $result = !!ERROR!! ]
Show Custom Dialog [ "An error occurred: " & PCQO_SGetStatus ]
End If
```

Example 2:

```
Set Variable [ $result = PCQO_SomePluginFunction( "a" ; "b" ) ]
If [ $result ≠ 0 ]
Show Custom Dialog [ "An error occurred: " & PCQO_SGetStatus ]
End If
```

It is good practice to ALWAYS trap for errors immediately after the PCQO_BeginSession, PCQO_Authorize and PCQO_RqExecute functions.

Please find a list of return codes and descriptions below for your reference:

Error Number	Error Text
1	Informational warning
-3	Invalid parameter value
-990	General Exception
-991	Validation Exception
-1000	Session Error
-1001	Session not authenticated or begun
-500	Invalid API operation defined
-501	Invalid entity defined
-502	Invalid field name specified
-503	Field not valid for current operation
-504	No object loaded that can be modified or accessed
-505	Invalid entity for execution
-506	Related record not found
-507	Invalid type of related record
-508	Invalid call order
-509	No related record to close
-510	Response is null

Error Number -990

While processing a request, the plug-in may encounter an error that it is not expecting, and as a result, it will report an error with a code of "-990". The -990 code is a general exception code that usually means that something happened internally within the plug-in, and not necessarily as a direct result of communicating with QuickBooks Online. Potential causes of a -990 error would be a function being called out of order, a special character being used that isn't allowed (such as "<", "&", and in some cases the smart quotes " and '), or some unknown exception error that was not anticipated by the plug-in.

When troubleshooting a -990 error, the best solution would be to analyze what information is being sent into the plug-in to be sent to QuickBooks Online. Most causes of -990 are data errors; a letter being used instead of a number, an invalid character, a function being called out of order, etc. There have been enhancements made to the handling of internal processes within the plug-in to help mitigate -990 errors, but the plug-in cannot account for every possible combination of factors. If you need further assistance in troubleshooting a -990 error, please reach out to our support team (see the Contact Us section).

16) Tips

Tax Tips

Tax varies depends on the version of QuickBooks Online you are using and requires knowledge of how QuickBooks Online handles tax. Since we do not provide QuickBooks training, we assume that you already have knowledge of how your QuickBooks Online 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.

Global Tax Model

Intuit describes a “global tax model” that QuickBooks Online adheres to, as well as specific instructions on how the global tax model fits in with QuickBooks Online entities. Please reference this when setting tax information for applicable QuickBooks Online entities. The link to the global tax model details is below.

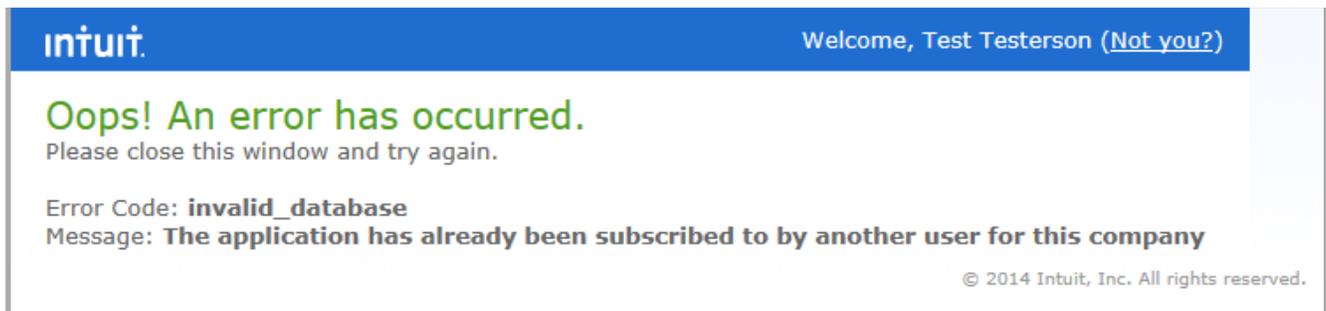
<https://developer.intuit.com/app/developer/qbo/docs/workflows/manage-sales-tax-for-non-us-locales>

17) Known Issues

Maximum Number of Active Sessions with QBO

There is currently a limit on the number of active sessions that a third-party application can have with a given QuickBooks Online company file. There can be no more than **ONE** active session from an application with a company file. This is a restriction put in place by Intuit.

If a session is authorized with the FM Books Connector Online plug-in for a company file, communication can take place between FileMaker and QuickBooks Online via that machine's connection. If a second machine wishes to connect to the same QuickBooks Online company through a different user account, they will receive the following error message in response:



If the second machine connects to QuickBooks Online through the same user account as the first, Intuit will disconnect the first machine and allow the second machine to be the new conduit through which to communicate with QuickBooks Online.

NOTE: As of version 1.0.1.0 you will be able to share that single connection across multiple machines with custom scripting. Please see section Multi-User Session Management for more details.

Microsoft EDGE Browser

At this time, QuickBooks Online does not support EDGE. This is an issue with QuickBooks Online, not a deficiency with the plug-in. When connecting to QuickBooks Online with the FM Books Connector Online Edition, be sure to use an alternative browser.

The plug-in will attempt to connect to QuickBooks Online using the default web browser of the system. If the default web browser is Microsoft EDGE, it is recommended to switch the default browser by following the instructions below:

- 1) Open the Start menu and choose "Settings"
- 2) Choose the "System" option in the Settings window
- 3) Select "Default apps" from the list on the left
- 4) Scroll down until you see "Web Browser"; click the app "Microsoft EDGE" to choose a new app
- 5) Select your browser of choice from the list provided. If you do not see your desired browser, you must download and install the browser from the browser developer's site.

Supported browsers for Windows:

- Internet Explorer, version 10 or later (32 bit only)
- Mozilla Firefox, recent versions (QuickBooks blocks anything older than 2 releases past)
- Google Chrome, recent versions (QuickBooks blocks anything older than 2 releases past)
- Safari for Windows version 6.1 or later

For more information, please click this link: <https://qbo.support/microsoft-edge-browser-known-issues-in-quickbooks-online-4/>.

Intuit Throttling Policy

In 2016, Intuit introduced a restriction to communication with QuickBooks Online companies. Each authorized connection is permitted 500 requests per connection per minute. If performing a large number of requests (such as updating a large quantity of customers or posting a large number of invoices), it is best to make use of the Pause Script script step in order to prevent the plug-in from reaching Intuit's throttling limit. If the throttling limit is reached, the request that would break that limit will return an error message similar to the one below (displayed in XML):

```
<RestResponse xmlns="http://www.intuit.com/sb/cdm/v3">
  <Error RequestId="12345">
    <RequestName>ErrorRequest
    <ProcessedTime>2015-12-31T10:10:01+00:00
    <ErrorCode>3001
    <ErrorDesc>message=This client has made too many consecutive requests over too short a
period of time. Please wait a short amount of time before attempting to submit again;
errorCode=003001; statusCode=403; source=Throttling Policy
  </Error>
</RestResponse>
```

Installer Prerequisites

In some cases, installing the plug-in and its package contents on Windows may encounter an issue with recognizing the plug-in or its prerequisites. In these cases, the likely culprit is a missing dependency for the plug-in. Usually, performing an uninstall/reinstall of the plug-in while running the processes as Administrator can resolve it. If that does not succeed, the following steps can be taken in order to resolve the problem:

- 1) Download the Visual C++ Redistributable Package at the following site:
<https://support.microsoft.com/en-us/kb/2977003>
 - a. Note: Depending on the plug-in, you will need either the Visual Studio 2008 or Visual Studio 2013 packages, in either x86 (32-bit) or x64 (64-bit) types.
- 2) Run the downloaded installer to completion
- 3) Run the installer for the plug-in

This should resolve the problem. If there is still an issue, or if you need assistance troubleshooting this issue, please feel free to reach out to our support team by email at support@productivecomputing.com, or by phone at (760) 510-1200 Monday through Friday from 8:00am to 5:00pm Pacific time.

Known Issue: “invalid_client” Error Result

When connecting to a company, it is possible to receive an “invalid_client” authentication error when loading a session string from FileMaker. The common cause of this error is that the session information string has been loaded on another machine and has since been refreshed, but that machine has not saved its updated session to FileMaker. To resolve this, please ensure that any workflow that loads session information will also immediately save their session information to FileMaker, as it may change due to being automatically refreshed. Sessions may also be saved after issuing calls to `PCQO_RqExecute`, as execution may also result in an automatic refresh. This error is most common in server-side deployments. Beginning a session anew will also resolve the “invalid_client” error; this will also invalidate any existing sessions that may be stored, so use it if any other resolution attempts fail.

Known Issue (Mac server-side):

PCQO_SLoadSessionInfo “Failed to Decode Authentication String” Error

We have noticed on Mac server-side deployments on macOS 10.15 Catalina that the attempt to load the session during a PSOS script call may result in an error that the plug-in could not decode the authentication string. In testing, we have found success in working around this by calling the PCQO_SLoadSessionInfo function a second time. See below for an example of an updated authentication workflow run on server-side:

```
# First authentication attempt
Set Variable [ $result ; Value: PCQO_SLoadSessionInfo( Main::Session Info String ) ]
If ( $result = 0 )
    # First authentication attempt succeeded, so all is good; save the session in
    case it refreshed and report success.
    Set Field [ Main::Session Info String; Value: PCQO_SSaveSessionInfo ]
    Exit Script [ 0 ]
Else
    # First authentication attempt failed, so run it again in case the error is the
    “Failed to Decode Authentication String” error
    Set Variable [ $result ; Value: PCQO_SLoadSessionInfo( Main::Session Info String
    ) ]
    If ( $result = 0 )
        # Second authentication attempt successful, so save the session in case it
        refreshed and report success.
        Set Field [ Main::Session Info String; Value:
        PCQO_SSaveSessionInfo ]
        Exit Script [ 0 ]
    Else
        # Second authentication attempt failed; error might be something else. We
        recommend beginning a new session at this point.
        Exit Script [ “ERROR - ” & PCQO_SGetStatus ]
    End If
End If
```

III. Contact Us

Successful integration of a FileMaker plug-in requires the creation of integration scripts within your FileMaker solution. A working knowledge of FileMaker Pro, especially in the areas of scripting and calculations is necessary. If you need additional support for scripting, customization or setup (excluding registration) after reviewing the videos, documentation, FileMaker demo and sample scripts, then please contact us via the avenues listed below.

Phone: 760-510-1200

Email: support@productivecomputing.com

Forum: www.productivecomputing.com/forum

Please note assisting you with implementing this plug-in (excluding registration) is billable at our standard hourly rate. We bill on a time and materials basis billing only for the time in minutes it takes to assist you. We will be happy to create your integration scripts for you and can provide you with a free estimate if you fill out a Request For Quote (RFQ) at www.productivecomputing.com/rfq. We are ready to assist and look forward to hearing from you!