

# ColdBooks

<b>Welcome to the ColdBooks Documentation .....</b>	<b>2</b>
<b>Introduction to ColdBooks .....</b>	<b>2</b>
Traditional Integration techniques.....	3
How QuickBooks interacts with web-based applications.....	3
How ColdBooks Works .....	3
<b>Requirements .....</b>	<b>4</b>
ColdFusion Version .....	4
ColdSpring.....	4
Supported QuickBooks Versions.....	4
<b>QuickBooks SDK.....</b>	<b>5</b>
<b>Installation.....</b>	<b>5</b>
Install the QuickBooks Web Connector.....	5
Install ColdBooks .....	6
<b>ColdBooks Administrative Interface Overview .....</b>	<b>7</b>
<b>ColdBooks Logs.....</b>	<b>8</b>
<b>Creating a Connection in ColdBooks .....</b>	<b>10</b>
Connections can be created via CFML.....	15
<b>Using ColdBooks in your Application .....</b>	<b>16</b>
Installing ColdSpring .....	16
Minimal Usage of ColdSpring.....	17
Sending Messages to QuickBooks .....	18
Creating QBXML Requests .....	19
Creating Object-Based Requests.....	24
Checking on the Status of a Request.....	27
Checking Your Logs.....	27
Handling Responses.....	28
Tips for Debugging .....	29
<b>ListId and EditSequence .....</b>	<b>30</b>
<b>Read the QuickBooks SDK Documentation as Well .....</b>	<b>30</b>
<b>Support For ColdBooks .....</b>	<b>30</b>
<b>If you either need hands on assistance or implementation services, Alagad is happy to help you via our paid services! .....</b>	<b>31</b>

## Welcome to the ColdBooks Documentation

Thanks for your interest in ColdBooks. This document provides information on how to get started using ColdBooks to connect your ColdFusion applications to one or more instances of QuickBooks.

Read on to learn more about ColdBooks....

## Introduction to ColdBooks

ColdBooks is a free, open source (under the LGPL), tool for connecting ColdFusion applications to one or more instances of Intuit QuickBooks. This is a non-trivial process that ColdBooks attempts to improve by automating most of the tedious and painful processes involved in QuickBooks integration and providing a simple, generic, API for your usage.

Before you get started, you should be aware that QuickBooks integration does not work the way you may be used to integrating with third party products. There are actually several ways to interact with the traditional desktop versions of QuickBooks. Unfortunately, each of these is limited by QuickBooks core architecture. Specifically, QuickBooks was designed by Intuit to be used by non-technical people who would run the application under their user account when logged in. The rest of the time the application would be shut down and not available.

Furthermore, each of the APIs provided to interact with QuickBooks requires the user's explicit permission to access QuickBooks data. This is handled by first having the QuickBooks file being connected to open in QuickBooks and secondly by showing popup windows in QuickBooks which the user uses to grant permission to the specific application to access QuickBooks's data.

This leads to a problem where, to connect to QuickBooks it must be running in a user process. This ultimately means that server processes (such as services) cannot be configured to access QuickBooks. That's not to say this is *impossible*, but beyond the capabilities of this programmer. As such, the only effective (and Intuit-supported) way to connect to QuickBooks from a web application is to use a tool called the QuickBooks Web Connector (QBWC).

The QBWC is a small application that, like QuickBooks, runs in the user's process while they're logged in. The QBWC sits in the user's system tray and runs while the user is logged in. QuickBooks does not need to always be running, but does need to be running on the initial connection to a new application. (More on this later.)

The net outcome of this is your application is only able to communicate with QuickBooks while the QBWC is running. The QBWC can easily be shut down by the end user. Also, if the user shuts down their desktop or loses their network connection, or any other host of potential problems, your application will not be able to talk to QuickBooks.

## Traditional Integration techniques

Traditional integration techniques take a client-server approach. That is, a client application (like the one you're trying to create) will connect to a server somehow and ask for data.

The most obvious example of this would be connecting to a database. In ColdFusion you establish a Datasource Name (DSN) and use that to read and write data to your SQL database server which is always awaiting requests on the network.

## How QuickBooks interacts with web-based applications

Because of the problems described above, the tried and true client-server approach is not how it works when integrating with QuickBooks. Instead, Intuit has, for whatever reason, taken an approach that this developer will simply refer to as "bass-ackwards".

Remember that though there are other APIs for interacting with QuickBooks, none of them are easily or reliably accessible remotely. As such, the way QuickBooks integration works when using the QBWC is that the QBWC uses the other APIs to talk to QuickBooks for you. However, you cannot talk directly to the QBWC. Instead, the QBWC uses its own configuration to figure out what to talk to.

To clarify this, the QBWC uses a QWC file to create a connection to a web application. The QWC file specifies important information like the name of a connection, its requested permissions, and a URL to a web service exposed by that web application. Then, according to either the default schedule specified in the QWC file, or according to however the end user changes these settings (including disabling the schedule or shutting down the QBWC completely), the QBWC will make occasional calls to the specified web service and then *ask the remote application if it needs anything to be done*.

It's important to emphasize this fact: The QBWC talks to your application and not the other way around. Because of this approach your application must be built in such a way that it satisfies all of the requirements of the QBWC as well as being robust enough to handle situations where the QBWC is not making requests to your application and to handle any errors that may arise.

This is where ColdBooks comes in.

## How ColdBooks Works

ColdBooks is a generic system for creating connections to QuickBooks. It draws inspiration from ColdFusion data sources and how they are configured. It allows multiple connections to multiple instances of QuickBooks. Each of these connections is given a name, similar to DSNs. The connection you create also generates the QWC file that the end user can use to connect the QBWC to the specific connection in ColdBooks.

Once the connection is established you can use the ColdBooks connection to send messages to QuickBooks. Ok, really the QBWC will ask the connection for messages. In reality you're simply enqueue a message and the QBWC later ask for it.

Messages are instructions on what you want QuickBooks to do. These can be almost anything from listing accounts to creating employees to logging time entries to creating invoices. Anything that QuickBooks exposes you can do through these messages to ColdBooks. Messages are ultimately sent to the QBWC in a format called QBXML. As a developer you can elect to either write your own QBXML or use a set of objects to describe what you want to do.

ColdBooks takes the messages and writes them into a database where they are kept until the next connection from the QBWC. When the QBWC connects these messages are sent to the QBWC which then relays it on to QuickBooks for processing. The results of the messages are returned to ColdBooks.

When you send a message you also specify a CFC and a method on the CFC that will handle the response. Responses are either returned as QBXML data or as populated objects. From here it's your responsibility to do with this data as you want.

Don't worry; we'll get into code examples later!

## Requirements

### ColdFusion Version

Sorry to everyone who is on ColdFusion 8 or earlier, ColdBooks currently only supports ColdFusion 9 and later. This is due primarily to the use of script-based CFCs.

Community volunteers are welcome to work with the code to make it compatible with earlier versions of ColdFusion.

### ColdSpring

ColdBooks requires the use of ColdSpring within your application. If you're not a user of ColdSpring, don't worry; ColdSpring is only required to create instances of ColdBooks objects. The Using ColdBooks in your Application section below covers the basic installation and usage of ColdSpring.

### Supported QuickBooks Versions

ColdBooks currently supports modern desktop versions of QuickBooks for the US, UK and Canada. According to the QuickBooks SDK, the following versions of QuickBooks should be supported by ColdBooks:

- QuickBooks 2009 and Enterprise 9.0
- QuickBooks 2008 and Enterprise 8.0
- QuickBooks 2007 and Enterprise 7.0
- QuickBooks 2006 and Enterprise 6.0
- QuickBooks 2005 and Enterprise 5.0 (R5 and later)
- QuickBooks 2005 (and Enterprise 5.0)
- QuickBooks 2004 (and Enterprise 4.0)
- QuickBooks 2003, Enterprise 3.0 (R7 and above)

- QuickBooks 2003 (and Enterprise 2.0)
- QuickBooks 2002 (R2 and above) (and Enterprise)
- QuickBooks 2002 (R1)
- QuickBooks 2008-2009 Canadian
- QuickBooks 2008-2009 UK
- QuickBooks 2004-2007 Canadian
- QuickBooks 2004-2006 UK Editions
- QuickBooks 2003, Canadian and UK Editions, CA2.0, UK2.0 and all later versions of these through Quick-Books 2006

Each of these version of QuickBooks supports a different version of QBXML. If you want to know the specific version of QBXML supported for your version see “QuickBooks Products and qbXML/QBFC Support” in the QBSDK documentation that comes with the QuickBooks SDK.

ColdBooks does not currently support QuickBooks POS or Merchant Services. ColdBooks does not currently support the QuickBooks Online Edition. Support for these may be added in the future. (If you decide to add support for these, please coordinate with Alagad so these features can be added into the core product.)

## QuickBooks SDK

If you haven't already downloaded the QuickBooks SDK, you should. You can get it from <http://developer.intuit.com>. You'll need to create an account to log in. Once you're logged in download the QuickBooks SDK. At the time of writing this could be found here: [https://member.developer.intuit.com/myidn/technical\\_resources/download.aspx?type=qbsdk&v=80&d=1](https://member.developer.intuit.com/myidn/technical_resources/download.aspx?type=qbsdk&v=80&d=1)

The SDK is mostly useful for the Unified Onscreen Reference (UOR) which is included. The UOR provides documentation on all of the various versions of QBXML and its syntax.

## Installation

Installation of ColdBooks is reasonably simple. First...

### Install the QuickBooks Web Connector

To use ColdBooks you will need to install the QuickBooks Web Connector on the PC that has QuickBooks installed.

At the time of writing this, the latest QBWC can be downloaded from: <http://marketplace.intuit.com/webconnector/>.

This can be a royal pain to find in Intuit's websites. If needed use Google and search for “download quickbooks web connector”.

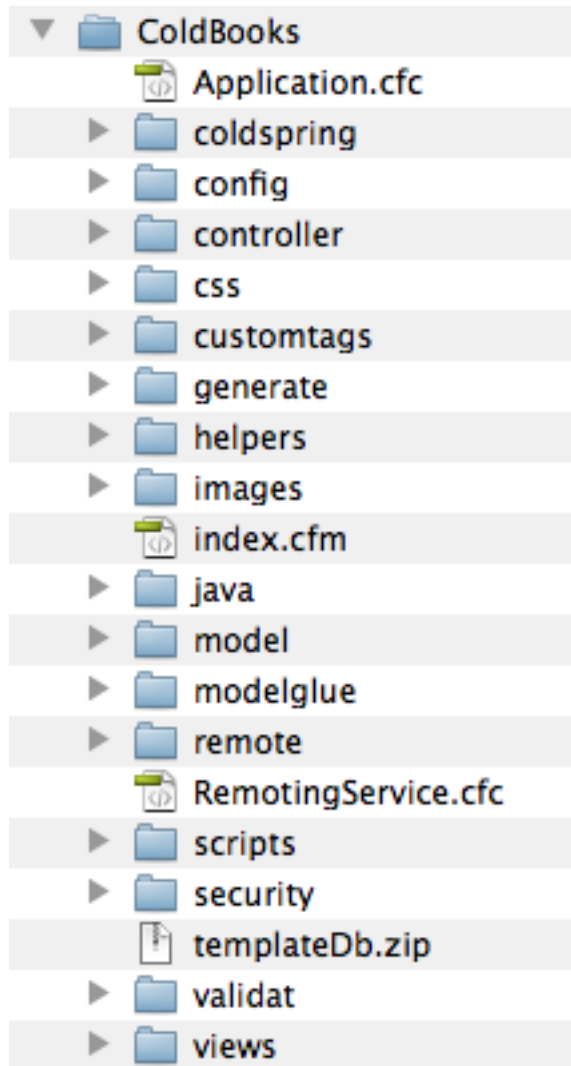
The default installation of the QBWC should work fine.

## Install ColdBooks

To use ColdBooks you do need to install it. ColdBooks integrates into the ColdFusion administrator and therefore needs to be installed in specific location.

Start by either downloading ColdBooks from the Alagad.com website or from the SVN repository at <http://svn.alagad.com/ColdBooks/trunk/>. (Currently no releases are tagged in SVN so just use the Trunk.)

Find the ColdBooks directory. This directory will have the following contents:

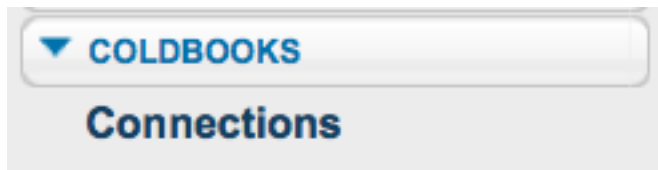


Copy the ColdBooks directory underneath your /CFIDE/administrator directory. This directory can be in many different locations on your server depending on how you installed ColdFusion. We trust that you can find this folder.

Next, you'll need to find and edit the custommenu.xml file underneath your /CFIDE/administrator directory. Before the closing menu tag in this document, you will need to add the following code:

```
<submenu label="ColdBooks">
    <menuitem href="ColdBooks/index.cfm?event=Connections"
        target="content">Connections</menuitem>
</submenu>
```

Upon restarting ColdFusion, when you log into the ColdFusion administrator you should see the following link in the left navigation:



The Connections link is your gateway to connecting to QuickBooks. When clicked, it will show the ColdBooks Administrative Interface.





## ColdBooks Administrative Interface Overview

After logging into the ColdFusion Administrator and clicking on the ColdBooks Connections link, you will see the following interface:

**ColdBooks Connections**

ColdBooks allows you to establish connections to QuickBooks via the QuickBooks Web Connector.

[Register New QuickBooks Connection](#)

QuickBooks Connections					
Actions	Connection Name	Web Service URL	Pending Messages	Errorred Messages	Last Connection On
   	<a href="#">Example Connection</a>	http://localhost/CFIDE/administrator/ColdBooks/remote/qbwc.cfc	0	1	Feb 2, 2010 at 2:20 PM

This page is a listing of all the connections established by ColdBooks and provides tools to create and manage connections.

The Register New QuickBooks Connection button is used to create a new ColdBooks connection (this can also be done in code).

In the list of connections you will see a row for each connection. Most of the columns are fairly self explanatory but we'll go through them anyhow:

**Actions** – This column has four buttons you can use to perform common actions. These are (left to right): Edit the Connection, View Logs, Download the QWC file, and Delete the Connection.

**Connection Name** – The name of the connection. (I bet you guessed that!)

**Web Service URL** – This is the URL that the QBWC will use to connect to ColdBooks. You may need this to insure that file is publicly accessible.

**Pending Messages** – This shows the number of messages that are queued up and waiting to be sent to QuickBooks.

**Errored Messages** – This shows the number of messages that were sent to QuickBooks and either errored on the QuickBooks side or in the call-back handling CFC. Of course, there's a small chance this might show errors directly in ColdBooks, but that's somewhat unlikely. This column is also a link to see the log.

**Lost Connection On** – This is the date and time that the QBWC last connected to this ColdBooks connection. This is important. If you have a client complaining that their web application isn't showing their most up to date data from QuickBooks, you can look here and see when their QBWC last connected.

## ColdBooks Logs

By clicking on either of the links to view the logs you can easily browse through a history of messages sent to QuickBooks and response returned. How you configure your connection determines what sort of logging information remains in ColdBook's database. The options are to retain all logs, retain only errored logs, or retain no logs. (More information about this setting is in the Creating a Connection in ColdBooks section of the documentation.

This is what the ColdBooks log browser looks like:

### ColdBooks Connection Log

Created	Modified	Callback CFC	Callback Function	Callback Format	Request XML	Response XML	Error Text
January, 31 2010 1	January, 31 2010 1	/Users/dhughes/Sit	handleAccountQue	xml	<AccountQueryRq	<?xml version="1.0 {code=, ExtendedIr	
January, 31 2010 1	January, 31 2010 1	/Users/dhughes/Sit	handleEmployeeQ	object	<EmployeeQueryR	<?xml version="1.0	
February, 10 2010	February, 10 2010	/Users/dhughes/Sit	handleAccountQue	xml	<AccountQueryRq		
February, 10 2010	February, 10 2010	/Users/dhughes/Sit	handleEmployeeQ	object	<EmployeeQueryR		
February, 10 2010	February, 10 2010	/Users/dhughes/Sit	handleEmployeeQ	object	<EmployeeQueryR		

Page 1 of 1

The following columns are represented:

**Created** – This is the date that the request was enqueued.

**Modified** – This is the date the request was last modified. In practice, if different from the created date, this is really the date the request was sent to QuickBooks and either succeeded or errored.



**Callback CFC** – This is the absolute path to a CFC that ColdBooks will notify when a response to the request is received.

**Callback Function** – This is the function on the Callback CFC that will be called when a response to the request is received.

**Callback Format** – This is the format that the returned data will be sent to the Callback function. The values can be XML or Object.

**Request XML** – This is the XML of the request being sent to QuickBooks. Note that you can send requests in either Object or XML format. Object-formatted requests are always translated to XML and stored in the database.

**Response XML** – This is the response XML (if any) received from QuickBooks.

**Error Text** – This is the text of any error that occurs. The format of this value is not predictable. If the error occurred in QuickBooks this will be a string with a QuickBooks-determined format. If the error occurred in the Callback function it will be a ColdFusion error formatted to text.

If you click on a row in the logs you can see details about the request (most importantly the complete XML request and response).

The screenshot shows the ColdBooks Manager interface. At the top, there's a browser window with the URL `http://scratch/CFIDE/administrator/ColdBooks/index.cfm?event=ViewL`. Below the browser, the main content area is titled "ColdBooks Connection Log". It contains a table with columns: Created, Modified, Callback CFC, Callback Function, Callback Format, Request XML, Response XML, and Error Text. The first row is highlighted in blue and shows a log entry from January 31, 2010, at 3:05 PM, for the callback function `handleAccountQue` in XML format. Below the table, there's a "Log Detail" section. It shows the "Created" date as Jan 31, 2010 at 3:05 PM, and the "Last Modified" date as Jan 31, 2010 at 3:06 PM. The "Callback CFC" is `/Users/dhughes/Sites/Scratch/testColdBooks/callback.cfc`. The "Callback Function & Type" is `handleAccountQueryXml(xml)`. The "Request XML" is shown as a collapsed node: `<-AccountQueryRq requestID="{3BE41933-98AF-B899-4246-0003ED9DF48F}">`. The "Response XML" is shown as an expanded node: `<-AccountQueryRs requestID="{3BE41933-98AF-B899-4246-0003ED9DF48F}" statusCode="0" statusMessage="Status OK" status`. Below the response XML, there's a collapsed node for `<-AccountRet>`, which is expanded to show `<-ListID>` with the value `80000035-1174096139`. At the bottom, there's a search bar with the text "Find: web" and buttons for "Next", "Previous", "Highlight all", "Match case", and "Phrase not found".

Created	Modified	Callback CFC	Callback Function	Callback Format	Request XML	Response XML	Error Text
January, 31 2010 1	January, 31 2010 1	/Users/dhughes/Sit	handleAccountQue	xml	<AccountQueryRq	<?xml version="1.0 {code=, Extended	
January, 31 2010 1	January, 31 2010 1	/Users/dhughes/Sit	handleEmployeeQt	object	<EmployeeQueryR	<?xml version="1.0	
February, 10 2010	February, 10 2010	/Users/dhughes/Sit	handleAccountQue	xml	<AccountQueryRq		
February, 10 2010	February, 10 2010	/Users/dhughes/Sit	handleEmployeeQt	object	<EmployeeQueryR		
February, 10 2010	February, 10 2010	/Users/dhughes/Sit	handleEmployeeQt	object	<EmployeeQueryR		

**ColdBooks Connection Log**

**Log Detail**

**Created**  
Jan 31, 2010 at 3:05 PM

**Last Modified**  
Jan 31, 2010 at 3:06 PM

**Callback CFC**  
/Users/dhughes/Sites/Scratch/testColdBooks/callback.cfc

**Callback Function & Type**  
handleAccountQueryXml(xml)

**Request XML**  

```
<-AccountQueryRq requestID="{3BE41933-98AF-B899-4246-0003ED9DF48F}">
</AccountQueryRq>
```

**Response XML**  

```
<-AccountQueryRs requestID="{3BE41933-98AF-B899-4246-0003ED9DF48F}" statusCode="0" statusMessage="Status OK" status
<-AccountRet>
  <-ListID>
    80000035-1174096139
  </ListID>
```

Find: web Next Previous Highlight all Match case Phrase not found

Done 120%

The screenshot above shows the log detail. Note that the XML of the request and response are nicely formatted and allow you to use the -/+ signs to collapse and expand nodes in the XML. Also, the full text of any error is shown at the bottom of the log detail.

## Creating a Connection in ColdBooks

Creating a connection in ColdBooks is very similar to creating a ColdFusion DSN. You simply need to click the button to Register a new QuickBooks Connection. This will show the following form:

## Add / Edit QuickBooks Connection

Use the form below to create a new connection to QuickBooks using ColdBooks.

Connection Name	<input type="text"/>
Description	<input type="text"/>
Password	<input type="password"/>
Support URL	<input type="text"/> <small>Example: <a href="http://www.example.com/path/to/support/url">http://www.example.com/path/to/support/url</a>. It's important to note that the web service URL must be on the same host name as the Support URL. For this reason, the path to the web service will be generated for you. IP Addresses will not work. The only way to have your web service URL not run under SSL is if you use localhost.</small>
Web Service URL	<input type="text"/>
Read Only	<input type="radio"/> No <input type="radio"/> Yes
Personal Data	<input type="button" value="Not Needed"/>
Scheduled Interval	Run every <input type="text"/> <input type="button" value="v"/> <small>Leave blank for no schedule</small>
Connection ID	<input type="text" value="3E50486B-A69A-801A-C4FCDDEB037BF296"/> <small>The Connection ID uniquely identifies <i>this</i> connection to quickbooks for whatever purposes you make. If you ever delete this connection you'll want to recreate it with the same Connection ID. In general, just leave this value to the generated value.</small>
Log Retention	<input type="radio"/> Retail all logs. (May include sensitive or personal information from QuickBooks.) <input type="radio"/> Retain only errored requests. (May include sensitive or personal information when CFC callbacks throw errors.) <input type="radio"/> Retail no logs. (Will not include sensitive or personal information from QuickBooks nor any other useful debugging information.) <small>ColdBooks stores requests in a database until they are fulfilled. For record keeping or debugging purposes you can keep a log of requests and their responses or error messages. However, certain requests may include sensitive or personal information such as social security numbers, account numbers, and more. You should consider how you want to store that information.</small>
Log Truncation	Truncate logs after <input type="text"/> days. <small>This indicates how many days logs will be kept before being deleted. A value of 0 means no logs will be kept (errors or otherwise).</small>

There is already a lot of useful documentation on this form, but we'll add more detail. Note that by creating the connection you are not actually *establishing* the connection to QuickBooks. Once you're done creating the connection you will need to download the QWC file and load that into ColdBooks.

**Connection Name** – This is the name of the connection. This serves the same purpose in ColdBooks that a DSN does for querying data from a database. Specifically, it is the name used when identifying a connection in the ColdBooks API.

**Description** – This is free form text describing the connection.

**Password** – When you download the QWC file and load it into the QBWC you will need to type in a password to the web service. This is where you specify that password.

**Support URL** – This is a URL to a page within your application. It could be the home page or an actual support URL. This is required by the QBWC and is shown within the QBWC user interface. The QBWC requires that the web service be exposed underneath this URL's domain. So, for this reason, this URL is used to generate the web service URL. There are a few important rules related to this:

All QBWC web service URLs must be provided over SSL. The only exception to this is when accessing the localhost. IE: for local development.

You cannot use an IP address in the URL and host names (if possible) are impractical.

This means that if you are developing on a different computer from the one that has QuickBooks on it (for example, if you're developing on a Mac) you will either need to configure SSL for your website or configure a proxy on the computer with QuickBooks that forwards request to <http://localhost> to your development server.

As of this writing using self-signed SSL certificates has not been tested, but is expected *not* to work.

In a nutshell, it's easiest if QuickBooks and ColdFusion are on the same computer while in development. If not, you will have to jump through the hoops of either setting up SSL on your development server or configuring an HTTP Proxy against localhost on the QuickBooks computer that forwards to your development server. The following is an example configuration of such a forwarding proxy in Apache:

```
ProxyRequests Off
```

```
<Proxy *>  
Order deny,allow  
Allow from all  
</Proxy>
```

```
ProxyPass / http://scratch/  
ProxyPassReverse / http://scratch/
```

This required enabling mod\_proxy in the apache httpd.conf file.

**Web Service URL** – As described above, this is generated based on the value provided in the support URL. The value is simply the host name in the support url prefixed with "https://" (unless the hostname is localhost) followed by "/CFIDE/administrator/ColdBooks/remote/qbwc.cfc". Because this file is underneath the

CFIDE directory all you need to do is insure that your application has /CFIDE configured as an alias or virtual directory in your web server.

**Read Only** – Indicates if the Connection will be used to only read data or to read and write data. If you indicate that the connection is read only and then try to write data QuickBooks will report errors.

**Personal Data** – The selected value indicates if the connection requires access to personal data from QuickBooks. This personal data may include information like social security numbers and more. It's recommended to set this as restrictive as you can. The options are:

**Not Needed** – This indicates that the connection does not need personal information.

**Optional** – Indicates that the connection may use personal information but that it's optional.

**Required** – Indicates that the connection requires access to personal information.

Note that the end user of QuickBooks who installs the QWC file has the option to deny this level of access to the connection.

**Scheduled Interval** – This indicates how often the QBWC will connect to ColdBooks to check for requests. You have a few options on how you can configure this setting. By leaving the "run every" field blank you are electing to have no default schedule. You may also provide a positive numeric value for the run every field and select an interval from the drop down menu (seconds or minutes).

In theory, you should be able to configure the QBWC to run every 1 seconds for near real time access to QuickBooks. Unfortunately, in practice there is a bug in the latest versions of the QBWC which prevent this from working. Furthermore, the end user can always override this setting or disable running the connection on a schedule.

You should consider the schedule to be a suggestion to the end user and not cast in stone.

**Connection ID** – This is a unique ID that QuickBooks will identify with this specific connection. This value is generated for you and in almost every situation you will want to leave this alone. The only case you may want to change this value is if you want to upgrade or change the settings for a specific connection in the QBWC.

**Log Retention** – This setting controls how ColdBooks retains logs. ColdBooks stores all requests in a database until they are fulfilled by the QBWC. For record keeping or debugging purposes you can keep a log of requests and their responses or error messages. However, certain requests may include sensitive or personal information such as social security numbers, account numbers, and more. You should consider how you want to store that information and set the log retention setting accordingly. The options are:


**Retain all logs** – This configures ColdBooks to keep all logs for all requests and response and errors. The only time a log entry will be deleted is if it's older than the log truncation setting allows for. This is risky on a public web server that will be working with personal information. There's a chance that information like social security numbers, etc, may end up in the logs.

**Retain only errored requests** – This setting tells ColdBooks to delete any logs where there were no errors. This is less risky than the retain all logs setting but may still have some personal information in the responses retained when the Callback CFC errors.

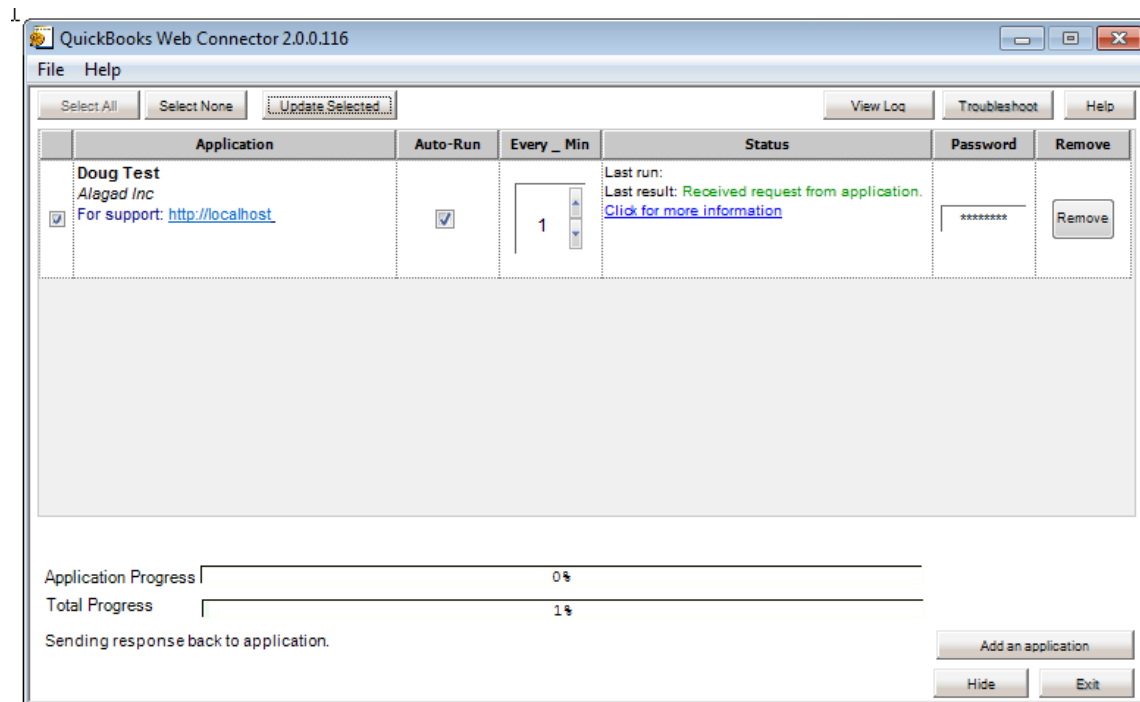
**Retain no logs** – This is the best setting for a production environment but least useful for debugging. In this configuration no logs are kept only the requests are recorded and they are deleted once sent to the QBWC whether they fail or not.

**Log Truncation** – This setting controls the number of days that logs are kept before being truncated. This setting overrides all other settings. After the configured number of days are passed any logs (errored or not) will be deleted.

When you are done configuring your connection, click the Submit button to create the Connection. You will see the new connection in the connections list.

The last step in establishing the connection is to click the download QWC file button (  ). This will download the QWC file that you can load into the QBWC tool by simply double clicking on it and following the resulting instructions in the dialogues.

Once you have loaded the QWC file into the QBWC you can select by checking the checkbox and run the connection by clicking the Update Selected button.



If you see any errors reported you can look at the QBWC's logs to trouble shoot the error. (Warning – here be dragons!)

### Connections can be created via CFML

You can also create a connection programmatically. Unfortunately the function to do this isn't very straightforward as it was designed to work with a collection of form data.

Before you can run this code you will probably want to read the next section of the documentation about Using ColdBooks in your Application.

Here is an example of how to create the connection and dump out the contents of the generated QWC file. Note that this example assumes that you a ColdSpring bean factory and have loaded ColdBooks into your application as documented in "Using ColdBooks in your Application":

```
<!-- the connection service is used to create connections and get them -->
<cfset ConnectionService = application.cs.getBean("ColdBooksConnectionService")
/>
```

```
<!-- create the connection -->
<cfset result = ConnectionService.saveConnection({
    name="My New Connection",
    description="My connection's description",
    password="foobar123",
    supportUrl="http://localhost",
    webserviceUrl="http://localhost/CFIDE/administrator/ColdBooks/remote/QBWC.
```

```

cfc",
    isReadOnly=false,
    personalDataPref="required",
    schedulerInterval=5,
    schedulerUnit="minutes",
    connectionId=createUUID(),
    logRetention="all",
    logTruncation=60
}) />

<!-- if there are errors dump them --->
<cfif result._errorCount>
    <cdump var="#result#" />
<cfelse>
    <!-- no errors, get the connection --->
    <cfset connection = ConnectionService.getConnectionByName("My New
Connection") />

    <!-- get the QWC xml --->
    <cfset qwcxml = connection.getQwsXml() />

    <!-- dump the qwcxml --->
    <cdump var="#qwcxml#" />
</cfif>

```

## Using ColdBooks in your Application

ColdBooks requires the usage of ColdSpring, a free and open source Inversion of Control framework for ColdFusion. A full explanation of ColdSpring is beyond the scope of this document. However, in its simplest form, ColdSpring is a framework that knows how to create objects correctly for use by you. Instead of you being required to know how to create the three objects that the ColdBooks ConnectionService requires (*and all the objects that they depend on (and all the objects that they depend on (etc...))*), you simply ask ColdSpring for the ColdBooks ConnectionService and it comes to you ready to use.

We will cover the minimum ColdSpring knowledge required to use ColdBooks. For more information you may want to visit <http://www.coldspringframework.org> and the almighty Google.

### Installing ColdSpring

ColdBooks uses ColdSpring under the covers and ships with a copy of the framework. For the lazy developer who wants the quickest path to getting started with ColdBooks, you can simply use this version of ColdSpring within your application. There are two ways to do this:

- Create a ColdFusion mapping named ColdSpring pointing to the /CFIDE/administrator/ColdBooks/coldspring directory.



- Copy the /CFIDE/administrator/ColdBooks/coldspring directory underneath the webroot of your application.

Alternatively, those with more experience can get a copy of ColdSpring from the framework's website or CVS repository and install it however you so wish.

### Minimal Usage of ColdSpring

No matter how you install ColdSpring, once it's installed you will need to use it within your application. To do this you will need to create a basic ColdSpring configuration file and create an instance of the framework using the configuration file.

Assuming you don't already use ColdSpring, you can simply create a file named ColdSpring.xml in your webroot. The file should have these contents:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">

<beans default-autowire="byName">

    <import resource="/CFIDE/administrator/ColdBooks/config/ColdSpring.ColdBooks.c
    />

</beans>
```

### READ THE ABOVE SECTION AND NOTE THE "import" TAG USAGE!

At the risk of being redundant, you need to add this tag into your ColdSpring.xml:

```
<import
resource="/CFIDE/administrator/ColdBooks/config/ColdSpring.ColdBooks.core.xml"
/>
```

This configuration file simply tells your instance of ColdSpring to import the ColdBooks configuration from the path specified. If you end up defining any other objects in your system using ColdSpring you can simply add them into your ColdSpring.xml file as you normally would.

Once you've created your configuration file you will need to create an instance of ColdSpring that uses this configuration file. This is quite simple. The following lines of code can go in your Application.cfm or Application.cfc file:

```
<cif NOT StructKeyExists(application, "cs")>
```

```

<!--// define and load a new instance of ColdSpring //-->
<cfset application.cs = createObject( "component" ,
"coldspring.beans.DefaultXmlBeanFactory" ).init() />

<cfset application.cs.loadBeansFromXmlFile(
expandPath("Coldspring.xml"), true ) />
</cfif>

```

This code creates a variable in the application scope called “cs” that is an instance of the ColdSpring “BeanFactory”. This is the object you will use to create ColdBooks objects. Because we don’t want to (and don’t need to) create a new instance of the bean factory on every request, we’ve simply wrapped it in a cfif that checks to see if the object already exists. As you work with your application you may run into situations where you need to reload the bean factory. We’re leaving it up to you to be creative in that situation.

### Sending Messages to QuickBooks

Now, after installing, configuring, and creating an instance of ColdSpring you can at long last use the ColdSpring BeanFactory to create instances of ColdBooks objects. In this section we’ll show you how to use the ConnectionService to get a specific Connection and then use that Connection to send messages to ColdBooks.

For this example, use the ColdBooks administration interface and create a new connection named “Example Connection”. Then import the generated QWC file into your QBWC install. After the first run of the new connection in the QBWC, you should see the connection in the ColdBooks administration interface has 0 messages, 0 errors, and a last connection time.

Now that we have a connection, let’s get a handle on it. The first thing we need to do is use the ColdSpring BeanFactory (application.cs in our examples) to create an instance of the ConnectionService:

```

<!-- the connection service is used to create connections and get them --->
<cfset ConnectionService = application.cs.getBean("ColdBooksConnectionService")
/>

```

As you can see, getting the ConnectionService is not terribly difficult. We just call getBean() on the ColdSpring BeanFactory and pass in the name “ColdBooksConnectionService”.

Next, we’ll need to use the ConnectionService to get our new connection.

```

<!-- Get a particular connection --->
<cfset Connection = ConnectionService.getConnectionByName("Example Connection")
/>

```

Again, as you can see, getting the connection is not terribly difficult. We simply use the `getConnectionByName()` function on the `ConnectionService` and pass in the name of our connection.

The `Connection` object has two functions on it for sending messages to ColdSpring. These functions are:

- `sendXmlRequest()` – This function sends a QBXML formatted message to QuickBooks and specifies a callback object and function and a specific format to receive response data in.
- `sendRequest()` – This function sends a message to QuickBooks using a set of objects and specifies a callback object and function and a specific format to receive response data in.

Note that the primary difference between `sendXmlRequest()` and `sendRequest()` are the fact that one accepts data in an XML format and the other accepts data in Objects. The same is true for CFCs that accept callback data; they can accept data in XML or as Objects.

We'll start with the `sendXmlRequest()` function. This function accepts the following arguments:

**XML** – This is a QBXML XML formatted request.

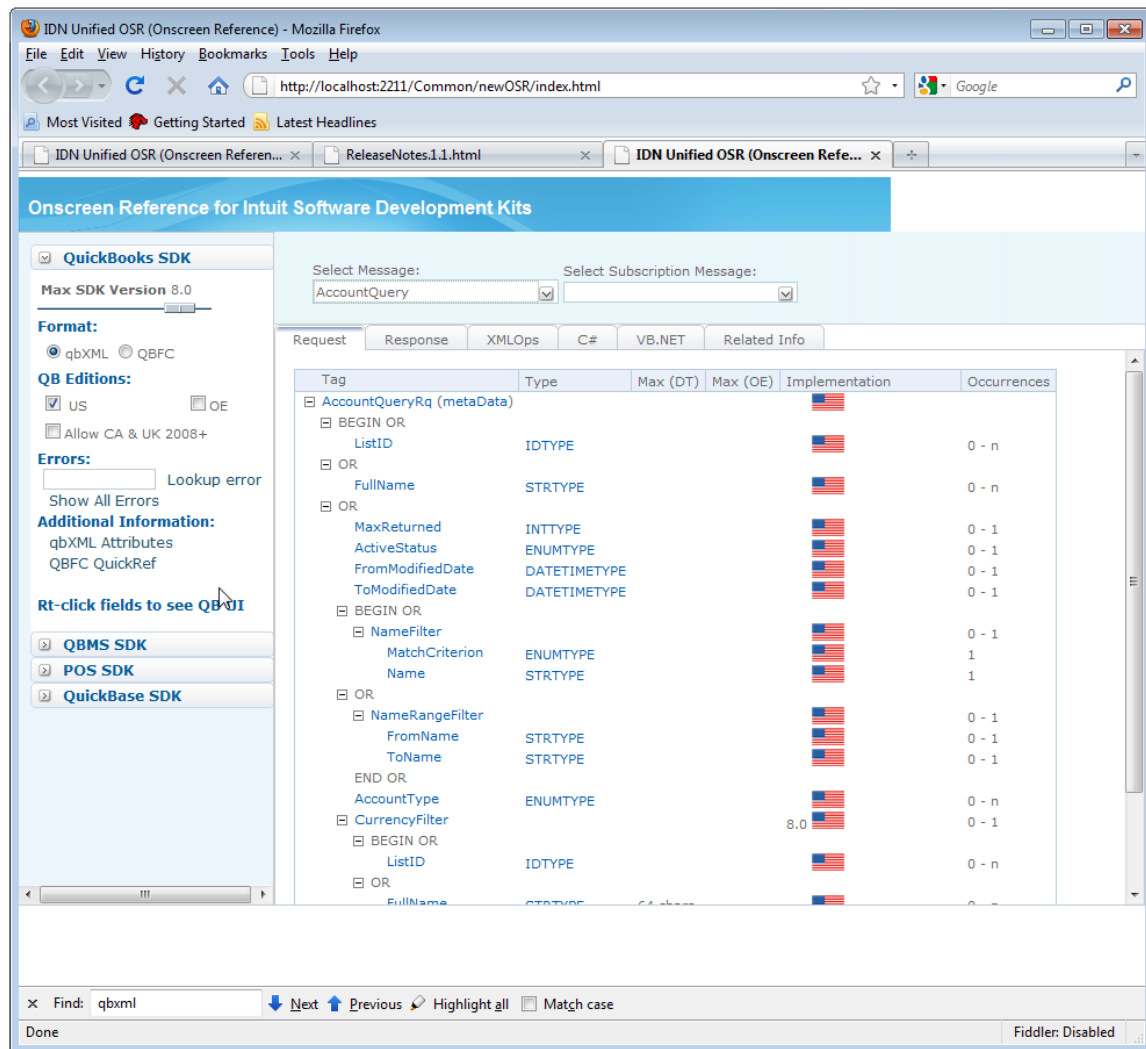
**callbackCFC** – This is the absolute path (not relative!) to a CFC that will be invoked when data is returned from QuickBooks.

**callbackFunction** – This is the name of the function on the callback CFC that will be invoked when data is returned from Quickbooks.

**returnFormat** – This is the format in which data will be returned from QuickBooks and passed into the callback function on the callback CFC. More on this in later parts of the documentation...

### Creating QBXML Requests

Since we're working with the `sendXmlRequest()` function you will need to create the XML data to send to QuickBooks. You may be wondering how to know what to write into the XML. To determine this you can use Intuit's Unified Onscreen Reference that is shipped with the QuickBooks SDK that you downloaded earlier. Here is a screenshot of this tool:



In this tool you will want to select the QBXML format and select a specific max version of the SDK you want to use. Also, insure that the country edition of your QuickBooks is correctly selected. Note that different versions of QuickBooks support different versions of QBXML.

If you're unsure what you need here, you can dump the connection object after the first connection is made and see the values in use. Here's an example of doing this:

```
<cfdump var="#Connection#" />
```

This will show the following dump output:

component <u>ColdBooks.model.entity.QbConnection</u> extends <u>ColdBooks.model.entity.Entity</u>		
PROPERTIES	<i>ColdBooksJavaLoader</i>	
	<i>ColdBooksMessageFactory</i>	
	<i>ColdBooksMessageDao</i>	
	<i>ColdBooksTranslator</i>	
	id	16
	name	Example Connection
	description	This is a new test for QB
	passwordHash	52E805342C7A75FE2028C430FD9616B5
	supportUrl	http://localhost
	webserviceUrl	http://localhost/CFIDE/administrator/ColdBooks/remote/qbwc.cfc
	fileId	[empty string]
	isReadOnly	0
	connectionId	3BCCD02A-DDE3-B03A-82EAA8ECFF34B4FC
	logRetention	all
	logTruncation	30
	personalDataPref	required
	schedulerInterval	1
	schedulerUnit	minutes
	lastConnectionDateTime	2010-02-10 16:10:14.68
	companyFile	C:\Users\Doug Hughes\Documents\Alagad Inc.QBW
	country	US
	qbXmlMajorVersion	7
	qbXmlMinorVersion	0
	createdDate	2010-01-31 15:03:47.123
	modifiedDate	2010-02-10 16:10:14.68
METHODS		

Note that three of the properties are Country, QbXmlMajorVersion and QbXmlMinorVersion. So, for our examples, I'll be setting the Unified Onscreen Reference to provide me documentation on the US edition of QBXML version 7.0.

Once I've selected my version and edition I can use the "Select Message" drop down to select a specific message I want to send to QuickBooks. For this documentation I will send a simple request to list vendors. If you look through the list of messages you will see a message for almost everything you can do in QuickBooks.

For this example, I'm using one of the sample company files provided with QuickBooks. And I'll be getting a list of Accounts using AccountQueryRq. I want to list every Account that contains the letter "B". Using the Unified Onscreen Reference, I can find the documentation on this XML as well as an example document. Based on this documentation, I've produced this XML:

```
<AccountQueryRq>
  <NameFilter>
    <MatchCriterion>Contains</MatchCriterion>
    <Name>B</Name>
  </NameFilter>
</AccountQueryRq>
```

Now that I've produced this XML I should validate it using tools provided in the QuickBooks SDK. I will save this into a text file named "test message.xml". And wrap it in some boilerplate XML that only matters for validation:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<?qbxml version="7.0" ?>
<QBXML>
  <QBXMLMsgsRq onError="stopOnError">

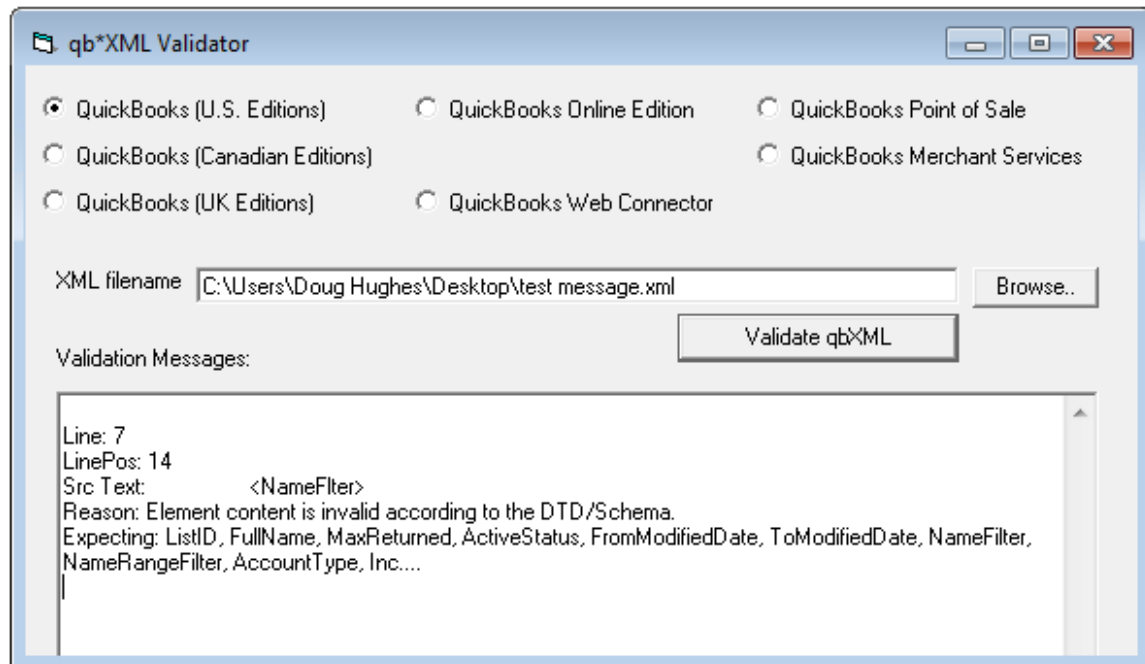
    <!-- your QBXML -->
    <AccountQueryRq>
      <NameFilter>
        <MatchCriterion>Contains</MatchCriterion>
        <Name>B</Name>
      </NameFilter>
    </AccountQueryRq>

  </QBXMLMsgsRq>
</QBXML>
```

There are only two parts of the XML above you need to worry about. The first is the QBXML that we wrote above and the version value in the ?qbxml processing directive. This version should be set to the API version you're supporting. In the case of this example, we're supporting version 7.0.

Once the file is saved I can load it into the XML Validator program that ships with the QuickBooks SDK and can be found in your start menu easily enough. There are actually a few different versions of this tool to choose from. I tend to choose the simple XML Validator tool because it's, well, simple.

In the XML Validator I simply load the file and make sure that I'm testing against the correct edition of QuickBooks, for me QuickBooks (U.S. Edition). When I click the Validate qbXML button. For the XML I wrote above I'll see this output:



As you can see, the validation message indicates I have a problem in my XML. After reading this I noticed that I have a typo in my XML. Instead of “NameFilter” I have “NameFlter” which is invalid. I’ll make my correction, revalidate it and I’ll see this output:



Excellent, now I know my XML is valid. I’ll grab the important part of the QBXML now and write some code to send the message to my connection:

```
<!-- create my AccountQueryRq xml -->  
<cfset xml = '<AccountQueryRq>
```

```

        <NameFilter>
            <MatchCriterion>Contains</MatchCriterion>
            <Name>B</Name>
        </NameFilter>
    </AccountQueryRq>' />

<!-- send the mesasge to QuickBooks -->
<cfset messageId = Connection.sendXmlRequest(xml, expandPath("callback.cfc"),
"handleAccountQueryXml", "xml") />

<cfoutput>#messageId#</cfoutput><br />

```

Just to reiterate the point, in my XML I do not need to provide any processing directives or any of the boilerplate qbxml or QBXMLMsgsRq tags. ColdBooks handles this automatically for me.

In the code above we pass the XML we wrote into the sendXmlRequest() function and specify the full path to a cfc, callback.cfc, that ColdBooks will notify when the QBWC finally gets around to running this request. The specific function that will be called is handleAccountQueryXml and the data returned from QuickBooks will be returned in XML format. (Note that in the Unified Onscreen Reference I can see the documentation on the format of the XML returned by clicking on the Response tab for a specific request.

### Creating Object-Based Requests

If you're not a fan of XML there is an alternative method you can use to send requests to QuickBooks using Objects.

As a user of ColdBooks it's not terribly important that you understand this, but the QuickBooks SDK includes a set of XSD documents that define the structure of XML for each edition and version of QuickBooks. When creating ColdBooks this XSD information was used to generate literally thousands of Java objects that map to the XML structure. You can make use of these Java objects to describe your request.

The first thing you need to know is how to create the objects you want to use. The ColdBooks Connection object has a getQbObjectFactory() function which returns an object specific to the edition and version of QuickBooks in use. This object has hundreds of functions on it to create different types of messages that can be sent to QuickBooks. As an example, if you look at the Unified Onscreen Reference you will see that the US edition, 7.0 version, of QBXML provides four messages for Accounts:

- AccountAddRq
- AccountModRq
- AccountQueryRq
- AccountTaxLineInfoQueryRq

For each of these messages there is a corresponding function on the QbObjectFactory:



- createAccountAddRqType()
- createAccountModRqType()
- createAccountQueryRqType()
- createAccountTaxLineInfoQueryRqType()

Note that this is simply the name of the message prepended with “create” and postpended with “Type”. You can follow this pattern to create any type of request object.

Once you have the request object it has a collection of functions that map up to properties defined in the Unified Onscreen Reference.

For example, we’ll work to create the same request as we did in XML. The first thing we’ll do is get an instance the AccountAddRq type using the createAccountAddRqType() function. By dumping it can can see what functions are on this object.

```
<!-- get the QbObjectFactory -->
<cfset AccountQueryRq =
Connection.getQbObjectFactory().createAccountQueryRqType() />

<cfdump var="#AccountQueryRq#" />
```

This code will dump the following output:

object of com.alagad.ColdBooks.US.v70.AccountQueryRqType		
Class Name	com.alagad.ColdBooks.US.v70.AccountQueryRqType	
Methods	Method	Return Type
	getAccountType()	java.util.List
	getActiveStatus()	java.lang.String
	getFromModifiedDate()	java.lang.String
	getFullName()	java.util.List
	getIncludeRetElement()	java.util.List
	getListID()	java.util.List
	getMaxReturned()	java.math.BigInteger
	getMetaData()	java.lang.String
	getNameFilter()	com.alagad.ColdBooks.US.v70.NameFilter
	getNameRangeFilter()	com.alagad.ColdBooks.US.v70.NameRangeFilter
	getOwnerID()	java.util.List
	getRequestID()	java.lang.String
	getToModifiedDate()	java.lang.String
	setActiveStatus(java.lang.String)	void
	setFromModifiedDate(java.lang.String)	void
	setMaxReturned(java.math.BigInteger)	void
	setMetaData(java.lang.String)	void
	setNameFilter(com.alagad.ColdBooks.US.v70.NameFilter)	void
	setNameRangeFilter(com.alagad.ColdBooks.US.v70.NameRangeFilter)	void
	setRequestID(java.lang.String)	void
	setToModifiedDate(java.lang.String)	void

Looking at the methods exposed, you can see that there is a functions to set and get the name filter. We can also see the type of object the set function expects (NameFilter). Using the QbObjectFactory we'll create instances of this object, then populate it and set it into our AccountQueryRqType object:

```

<!-- get the QbObjectFactory -->
<cfset QbObjectFactory = Connection.getQbObjectFactory() />

<!-- Creaet our Account Query request -->
<cfset AccountQueryRq = QbObjectFactory.createAccountQueryRqType() />

<!-- create a name filter -->
<cfset NameFilter = QbObjectFactory.createNameFilter() />
<cfset NameFilter.setMatchCriterion("Contains") />
<cfset NameFilter.setName("B") />

<!-- set the name filter into our Account Query -->
<cfset AccountQueryRq.setNameFilter(NameFilter) />

<!-- finally, send the request -->
<cfset messageId = Connection.sendRequest(AccountQueryRq,

```

```
expandPath("callback.cfc"), "handleAccountQueryObject", "object") />

<cfoutput>#messageId#</cfoutput><br />
```

The code above produces the exact same effect as the XML above, but to some programmers may be more expressive or easier to write than the XML. (They both have trade offs.

Note that in the `sendRequest()` function we pass in the `AccountQueryRq` object. I also specify the same call back CFC, but a different method and return type. This is entirely optional. I could return XML or Object data no matter how I send the request to QuickBooks.

### Checking on the Status of a Request

The `sendXmlRequest()` and `sendRequest()` functions returns a unique ID given to the message which you can store or ignore. If you store the ID you can use the `getRequestStatus()` function on the Connection object to check on the status of a message like so:

```
<!-- check on the status of my message -->
<cfoutput>#Connection.getRequestStatus("692224B2-E9C8-4E99-
937E3335626D5710")#</cfoutput>
```

The `getRequestStatus()` function will return one of four values:

**pending** – the message has not yet been sent to QuickBooks




**quickbooks error** – the message was sent to QuickBooks and QuickBooks reported an error.

**coldfusion error** – the message was sent to QuickBooks and a response was returned, however there was an error on the ColdFusion side of the process.

**complete** – the message has been sent to QuickBooks and a response was returned and handled successfully.

### Checking Your Logs

Now that we have a couple requests logged we can check on their status in the ColdBooks administration interface. In our admin interface we see the following:

QuickBooks Connections					
Actions	Connection Name	Web Service URL	Pending Messages	Errored Messages	Last Connection On
  	<a href="#">Example Connection</a>	http://localhost/CFIDE/administrator/ColdBooks/remote/qbwc.cfc	2	0	Feb 11, 2010 at 11:09 AM

This shows that there are two pending messages and no errors recorded. If you click to view the logs you will see the two pending requests (at least, depending on the number of requests you've recorded and your log retention settings).

If you click on a row in the table of logs you can see the details of the request:

## ColdBooks Connection Log

Created	Modified	Callback CFC	Callback Function	Callback Format	Request XML	Response XML	Error Text
February, 11 2010	February, 11 2010	/Users/dhughes/Sit	handleAccountQue	object	<AccountQueryRq		
February, 11 2010	February, 11 2010	/Users/dhughes/Sit	handleAccountQue	object	<AccountQueryRq		

Page 1 of 1

### Log Detail

#### Created

Feb 11, 2010 at 11:21 AM

#### Last Modified

Feb 11, 2010 at 11:21 AM

#### Callback CFC

/Users/dhughes/Sites/Scratch/testColdBooks/callback.cfc

#### Callback Function & Type

handleAccountQueryObject(object)

#### Request XML

```
<AccountQueryRq requestID="{6A5B5F4B-A1BE-CE17-6BDC-5B119D8F5D4C}">
  <NameFilter>
    <MatchCriterion>
      Contains
    </MatchCriterion>
    <Name>
      B
    </Name>
  </NameFilter>
</AccountQueryRq>
```

#### Response XML

[Empty String]

#### Error

[Empty String]

Looking at this image you can see that this is the Object-based request (because we used a different callback function name). You can also see that ColdBooks has translated the objects into XML format.

As your request is handled by QuickBooks you will see the Response XML and Error values updated with the results.

## Handling Responses

When QuickBooks finally connects to ColdBooks, ColdBooks will send all of the requests that are queued up for processing. QuickBooks will run the requests and either return errors or data. Errors are logged but your callback CFC is not notified.

As it's been stated several times above, responses from QuickBooks are sent to a callback handler CFC. This CFC must have the method indicated when the request was sent and it must accept data in the format specified. Here is a example callback CFC which implements the `handleAccountQueryObject()` and `handleAccountQueryXml()` functions used in the examples above:

```

component{

    public function handleAccountQueryXml(xml, messageId){
        writelog("Received XML for #messageId#");

        // your business logic goes here...

        writeDump(xml, "console");
    }

    public function handleAccountQueryObject(object, messageId){
        writelog("Received Object for #messageId#");

        // your business logic goes here...

        writeDump(object, "console");
    }
}

```

*Note: Although the code above is written using CF9's cfsyntax you can easily write this using traditional CFML as well!*

Looking at the callback.cfc above, you can see the two functions we're using on it. These functions have two arguments:

**xml -or- object** – This argument receives either an Object or XML data (in string format) that your function can work with. You'll want to insure that in your sendXmlRequest() or sendRequest() functions that you indicate the correct format for the callback function.

**messageId** – This is the generated ID of the message you sent. This is useful if you need to make a response to a specific request.

### Tips for Debugging

Because the callback function is called by ColdFusion in its own thread, debugging problems in these functions can be very challenging. Here are a few tips that will make your life at least a little bit easier:

- Try using the ColdFusion step-through debugger and add stop points in the callback CFC where you can step through the code.
- Run ColdFusion from the command line. When you do this you can see any data you log using the cflog tag or the writeLog() function shown as output in the console. You can also use the cfdump tag and set its output attribute to the console to see dumped

data in the console. You can also use the `writeDump()` function and set the second argument to “console”.

- Write a unit test. You can either use a testing framework like MXUnit or simply write a CFM script which instantiates your CFC and passes known data into these functions. Calling this directly will let you see what these functions are doing and you can use all the standard procedures you already do for debugging.

## ListId and EditSequence

All objects in QuickBooks are assigned a ListId value. This is essentially a unique Id specific to a particular instance of an object. The results from querying QuickBooks will always include a ListId. Any time you modify a record you will need to provide the ListId to identify the specific value. Overall, if you are trying to synchronize data in your application and QuickBooks you will likely need to know this value.

The EditSequence value is also returned in queries and is provided by you when editing a record. The EditSequence value changes in QuickBooks every time a record is updated. Here's what the QuickBooks SDK documentation has to say about this field:

Every time an object is changed, QuickBooks changes the value of the EditSequence element. In response to an Add, Modify, or Query request, QuickBooks returns the EditSequence. When your application attempts to modify an object, QuickBooks compares the EditSequence of your application's version of the object with the EditSequence of its own version of the object. If they match, then your application is up to date and QuickBooks continues with the operation. If they don't match, QuickBooks rejects the request and returns an error.

If the Modify request is processed successfully, QuickBooks returns an ObjectRet response (where Object is the name of the object modified).

## Read the QuickBooks SDK Documentation as Well

The documentation provided with the QuickBooks SDK is quite extensive. However, there is good information on the specifics of the data that can be exchanged with QuickBooks. It's worth a read, at least in part. Overall, if you don't see an answer to your question in this documentation (and it's not ColdBooks specific), check the SDK documentation.

## Support For ColdBooks

If you run into problems working with ColdBooks you are welcome to email Alagad using the Contact Us form on the Alagad.com website. As much as possible, we will work with you help you resolve problems.

**If you either need hands on assistance or implementation services,  
Alagad is happy to help you via our paid services!**