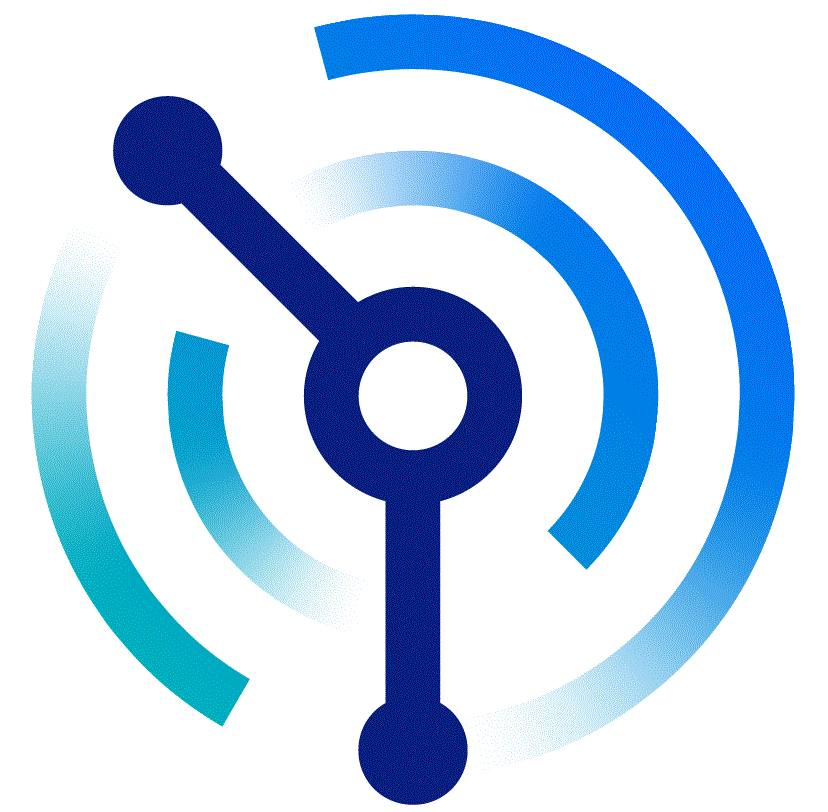
**Configuring Client-Channel Definition Tables (CCDT) with IBM MQ on Cloud**

**Author:** Ian Shore ([ishore@uk.ibm.com](mailto:ishore@uk.ibm.com)) - IBM

One of the topics during a recent presentation on 'IBM MQ and configuring for High Availability' covered the theory on improving availability for IBM MQ clients' connectivity using a Client-Channel Definition Table (CCDT). This capability enables a client application to connect to any one of a set of running queue managers (QMs), to improve availability. This article highlights the steps I went through to turn theory into practice, and the learning required to achieve connectivity from my IBM MQ client application to a local IBM MQ QM, an IBM MQ on Cloud QM in London, and an IBM MQ on Cloud QM in US-South (Dallas).

In summary, the areas covered in this article, to setup for CCDT usage are:

* Defining a CCDT JSON file
* Coding options in JMS for CCDT usage
* Configuring the mqccred channel exit

# Running the client application using CCDT and mqccred

After the client application, CCDT JSON file, and the security exit were set up (as described in this article), I was able to invoke my JMS client application to put messages to a named QM queue within a named queue manager group which all appeared across my available queue managers (at the end of the relevant connections defined in the CCDT). When one of the queue managers was not available, the messages appeared on the other available queue managers.

Outside of the JMS application, the connections are defined in the CCDT file and the credentials are defined in the mqccred.ini file. Figure 1 - Visualisation of CCDT setup shows how the client connectivity is configured.

A screenshot of a cell phone

Description automatically generated

Figure 1 - Visualisation of CCDT setup

# Defining a CCDT JSON file

## Why use CCDT definitions?

One of the key value propositions of using CCDT definitions is that you can loosely couple the connectivity from a client application to an available queue manager and its credentials by using [queue manager groups](https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.1.0/com.ibm.mq.dev.doc/q027490_.htm) and the mqccred security exit. There are configuration settings within the CCDT JSON file to route messages proportionally (with [clientWeight](https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.1.0/com.ibm.mq.ref.con.doc/q081750_.htm)) via each of the defined channels with or without channel [affinity](https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.1.0/com.ibm.mq.ref.con.doc/q081810_.htm). If a queue manager within a QM group is not available, CCDT connectivity will automatically try another in the QM group.

## What’s in a CCDT JSON file?

The CCDT JSON file describes connectivity attributes for a channel to connect directly to a specifically named queue manager or to one within a queue manager group.

Prior to IBM MQ v9.1.2 CCDT files could only be updated using runmqsc as they were in a proprietary binary format. However, since IBM MQ v9.1.2, they can now be created and configured in a [human-readable JSON format](https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.1.0/com.ibm.mq.con.doc/q132890_.htm). This article is going to focus on exploiting the CCDT JSON format.

IBM MQ on Cloud provides the ability to download a generated (starting point) CCDT JSON file from the "Connection information" section (as shown in Figure 2 - Download connection info in IBM MQ on Cloud).

A screenshot of a social media post

Description automatically generated

Figure 2 - Download connection info in IBM MQ on Cloud

The attributes that are allowed within this JSON CCDT are all described in the [IBM MQ documentation](https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.1.0/com.ibm.mq.con.doc/q132900_.htm). I found the '[Complete list of CCDT channel attribute definitions](https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.1.0/com.ibm.mq.con.doc/q132900_.htm)' very useful when I needed to correctly parent the ClientWeight and Affinity attributes for the channel.

At this point I was able to define JSON objects for three channels, each pointing at a different directly named queue manager, using:

* channel.name
* channel.clientConnection.connection.host
* channel.clientConnection.connection.port
* channel.clientConnection.queueManager
* channel.type

# Coding options in JMS for CCDT usage

## JMS code to use a CCDT file

To exploit this CCDT JSON file in your JMS client code, where factory is a MQConnectionFactory object, you need to:

* reference the CCDT JSON file using factory.setCCDTURL and choose which specific queue manager using factory.setQueueManager (Note: this can be a queue manager group name – see below)
* remove any usage of factory.setHost, factory.setPort, and factory.setChannel - as these settings will be pulled from the CCDT JSON file

At this point connection credentials still need to be passed in via the factory.createConnection method. We will introduce the security exit later in this article to pull in credentials from mqccred.ini.

## Creating a queue manager group

Up this this point, you should be able to use the CCDT file to connect to one specific queue manager, and connectivity credentials are hardcoded in the JMS client code. What is required is for a queue manager to be chosen transparently for the JMS client code from a **queue manager group** so that it does not need to be named directly in the JMS code. The only changes that are needed to achieve this are:

1. Modify all channel.clientConnection.queueManager entries in the CCDT JSON file to be the same name, and
2. Update the factory.setQueueManager in the JMS code to be the same value except prefix the name with an asterisk.

At this point, everything worked as expected; it kept failing with authentication errors! We now need to introduce the channel security exit, **mqccred**.

## Weighting where messages are sent

There are two main settings to control how the CCDT-enabled code chooses which client connection to select:

* channel.connectionManagement.clientWeight
* channel.connectionManagement.affinity

These values alter how a specific channel connection is selected. If, for example, each clientWeight is the same non-zero value and affinity is NONE, then messages will tend towards being evenly distributed across all the available queue manager connections in the queue manager group. The [IBM MQ knowledge center documentation](https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_9.1.0/com.ibm.mq.ref.con.doc/q081750_.htm) provides more detail.

# Configuring the mqccred channel exit

## Defining the channel security exit, mqccred

The channel security exit, mqccred is used to supply credentials on the channel when connecting to a specific queue manager. mqccred picks up the real name of the queue manager during connection time and then attempts to retrieve the appropriate credentials from [the mqccred.ini file](https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_9.1.0/com.ibm.mq.sec.doc/q114370_.htm) (referenced by the system's environment variable, MQCCRED). To activate the security exit for a channel, set mqccred(ChlExit) on the following JSON object in the CCDT:

* channel.exits.security.name

**mqccred** is a provided sample exit within your IBM MQ installation. Therefore, your client code must load the library using System.loadLibrary("mqccred") and you need to ensure that the  
-Djava.library.path Java VM argument includes .../tools/c/Samples/mqccred/bin64 and .../java/lib64.

The mqccred security exit will also perform checks on the mqccred.ini file to ensure that the passwords are obfuscated and that the file itself is protected from all users being able to view it. The channel exit will fail if these two checks do not pass (unless NOCHECKS is set on channel.exits.security.userData). runmqccred is provided to obfuscate the plain text passwords and instructions on how to use it can be found [here](https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_9.1.0/com.ibm.mq.ref.adm.doc/q114380_.htm).