



IBM Software Group

Using a Client Channel Definition Table (CCDT) in WebSphere MQ V7 for Queue Manager Groups

Angel Rivera (rivera@us.ibm.com)
WebSphere MQ Unix® Level 2 Support
21 February 2012 (Updated 17-Jun-2021)



WebSphere® Support Technical Exchange



Agenda

- Exploring different methods for a client to connect to queue manager
- Order of precedence of connection methods
- In depth explanation of CCDT
- Exploring different scenarios with queue manager groups.



Objective

- To describe in detail how a client application that is using the MQ Interface in C or the WebSphere MQ Classes for JMS Version 7, exploits the connection to multiple queue managers by using an MQ "Client Channel Definition Table" (CCDT)
- These queue managers are specified in a "queue manager group".

Problem statement

- You have many MQ client applications in different hosts and you want to connect these applications to different queue managers.
- If one of those queue managers is not available, then you want the application to try to connect to another queue manager.
- You do not want to hardcode the connection data (queue manager name, channel, port) in the application and you want a flexible way to provide the connection data to the applications.



Notes: Reasons for using a Qmgr Group

notes

- For more information, visit:

http://publib.boulder.ibm.com/infocenter/wmqv7/v7r0/index.jsp?topic=/com.ibm.mq.csqz.af.doc/cs12700_.htm

Queue manager groups in the CCDT

Some of the reasons for choosing to use a Queue Manager Group are:

- * To connect a client to any one of a set of queue managers that is running, to improve availability.
- * To reconnect a client to the same queue manager it connected to successfully last time, but connect to a different queue manager if the connection fails.
- * To automatically reconnect a client connection to another queue manager if the connection fails, without writing any client code.
- * To automatically reconnect a client connection to a different instance of a multi-instance queue manager if a standby instance takes over, without writing any client code.
- * To balance your client connections across a number of queue managers, with more clients connecting to some queue managers than others.
- * To spread the reconnection of many client connections over multiple queue managers and over time, in case the high volume of connections causes a failure.
- * To be able to move your queue managers without changing any client application code.
- * To write client application programs that do not need to know queue manager names.

Precedence for connecting to a QMgr

- This is the precedence of the features provided by the MQ client for an application to connect to a queue manager.

- 1: Pre-connect exit (introduced in MQ 7.1)
- 2: MQCONN API call
- 3: MQSERVER environment variable
- 4: mqclient.ini file (introduced in MQ 7.0)
- 5: CCDT file

1. Pre-connect exit (7.1)

- Pre-connect exit (introduced in MQ 7.1)
- Allows to store the client definitions in a global repository like an LDAP directory.
- This is the highest precedence.
- If it is available, then it will be used, superseding all other methods.
- It is an advanced topic.
- It is listed here for completeness.
- No further discussion in this presentation.

Notes: Pre-connect exit

notes

http://publib.boulder.ibm.com/infocenter/wmqv7/v7r1/topic/com.ibm.mq.doc/fa24020_.htm

WebSphere MQ > Developing applications > User exits, API exits and installable services > Writing and compiling exits and installable services

Referencing connection definitions using a pre-connect exit from a repository

- WebSphere MQ MQI clients can be configured to look up a repository to obtain connection definitions using a pre-connect exit library.
- A client application can connect to a queue manager using a client channel definition table (CCDT). Generally, the CCDT file is located on a central network file server, and have clients referencing it.
- Since it is difficult to manage and administer various client applications referencing the CCDT file, a flexible approach is to store the client definitions in a global repository like an LDAP directory, a WebSphere Registry and Repository or any other repository.
- Storing the client connection definitions in a repository makes managing client connection definitions easier, and applications can access the correct and most current client connection definitions.
- During the MQCONN/X call execution, the MQI client loads an application specified pre-connect exit library, and invokes an exit function to retrieve connection definitions. These are then used to establish connection to a queue manager. The details of exit library and function to invoke are specified in the mqclient.ini configuration file.

2: MQCONN API call

- The MQCONN or MQCONNX call connects an application program to a queue manager.
- It provides a queue manager connection handle, which the application uses on subsequent message queuing calls.
- If the name of the queue manager is hardcoded in the application that uses a Client transport type, then it supersedes the other methods:
MQSERVER, mqclient.ini and CCDT.

2: MQCONN API call

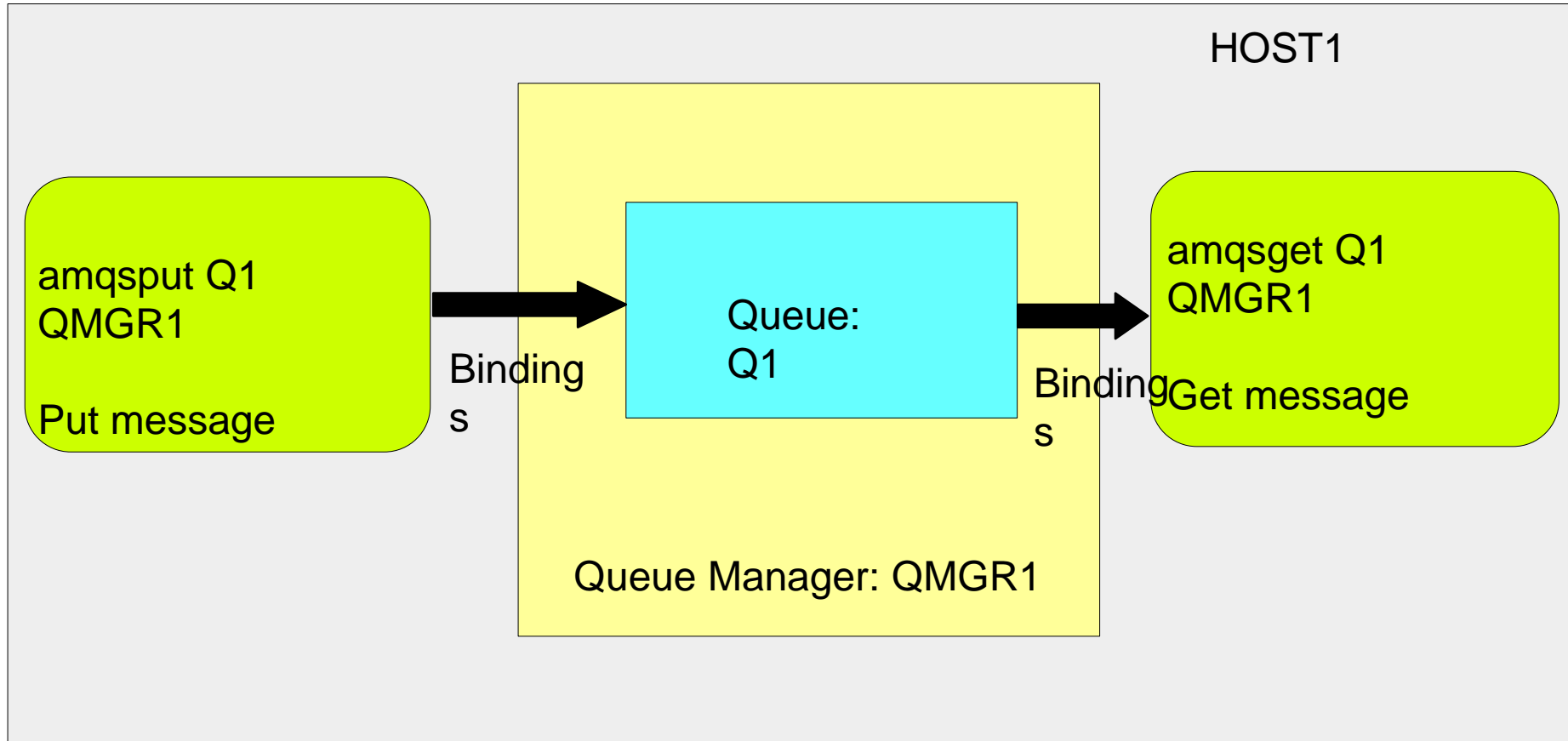
- Example from the sample “amqsbcg0.c” which is used to browse messages from a queue.
- The name of the queue manager is provided at runtime: amqsbcg Q1 QMGR1

```
/* variables for MQCONN          *****/
MQCHAR  QMgrName[MQ_Q_MGR_NAME_LENGTH];
MQHCONN Hconn = MQHC_UNUSABLE_HCONN;
MQLONG  CompCode = MQCC_OK, Reason = MQRC_NONE;
...
QMgrName[0] = '\0'; /* set to null default QM */
if (argc > 2)
    strncpy(QMgrName, argv[2], MQ_Q_MGR_NAME_LENGTH);
...
MQCONN(QMgrName, &Hconn, &CompCode, &Reason);
```

2: MQCONN API call

- Let's introduce a simple scenario.
- There are 2 client applications:
 - ▶ one for putting a message into a queue, and
 - ▶ another for getting a message
- The applications are located in the same host as the queue manager: HOST1
- These applications can connect using a transport type of: BINDINGS
 - ▶ local shared memory, no network is used

2: MQCONN API Put/Get using Bindings



3: MQSERVER environment variable

- Let's build on the previous scenario.
- The Put client application uses BINDINGS, but the Get client application is located in another host (HOST2)
- Q: What configuration / artifact is needed for the Get application to connect to the remote QMgr?
- **Answer:** The simplest option is to use the MQSERVER environment variable.
- Note: If used, then it supersedes the methods mqclient.ini and CCDT.

3: MQSERVER environment variable

- The MQSERVER environment variable is used to define a **minimal channel** between a client and a server connection (SVRCONN) channel from the queue manager.
- It specifies the location and port of the queue manager and the communication method.

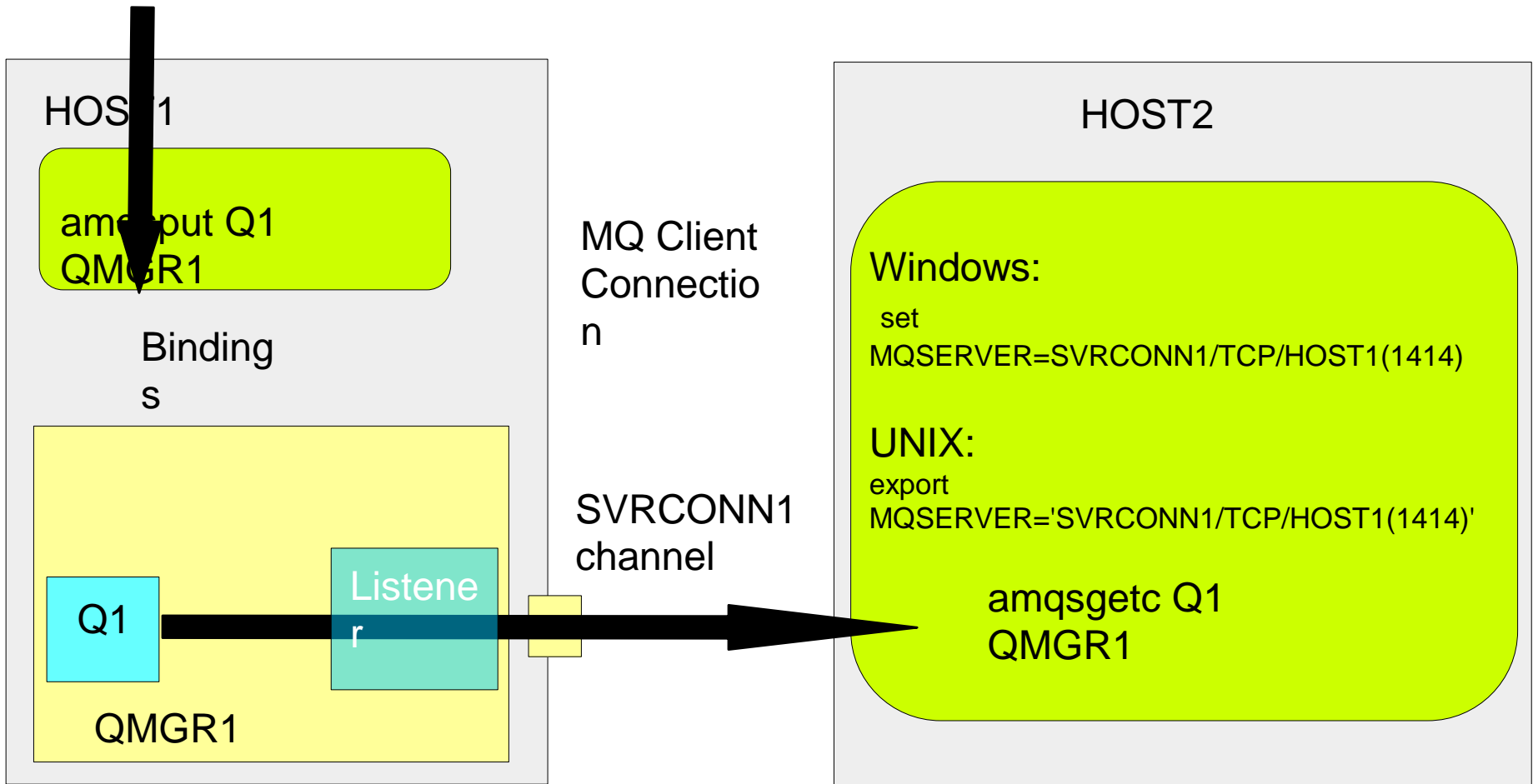
- For Windows®:

```
SET MQSERVER=ChannelName/TransportType/ConnectionName
```

- For UNIX and Linux system (notice the quotes):

```
export MQSERVER='ChannelName/TransportType/ConnectionName'
```

3: MQSERVER environment variable



3: MQSERVER and QMGr Group

- MQSERVER can be useful to specify a simple queue manager group, such as a multi-instance queue manager.
- For example, assuming that QMGR1 is a multi-instance queue manager in HOST1 and HOST2.
- The client application can automatically reconnect from HOST1 to HOST2, if HOST1 is not active.
- For UNIX and Linux® systems:
export MQSERVER=
'SVRCONN1/TCP/HOST1(1414),HOST2(1414)'

Traditional samples to put and get messages

Traditional samples to put and get messages:

- amqsput – put message, bindings mode
- amqsget – get message, bindings mode
- amqsputc – put message, client mode
- amqsgetc – get message, client mode

- They are located at:

Windows:

Execs: C:\Program Files\IBM\WebSphere MQ\tools\c\Samples\Bin

Source: C:\Program Files\IBM\WebSphere MQ\tools\c\Samples

- **UNIX:** executable: /opt/mqm/samp/bin
source: /opt/mqm/samp



High availability sample programs

- If you want to test the automatic reconnection feature, including doing a switchover to another queue manager, you will need to use additional samples.

http://publib.boulder.ibm.com/infocenter/wmqv7/v7r0/index.jsp?topic=/com.ibm.mq.csqzal.doc/fg17235_.htm

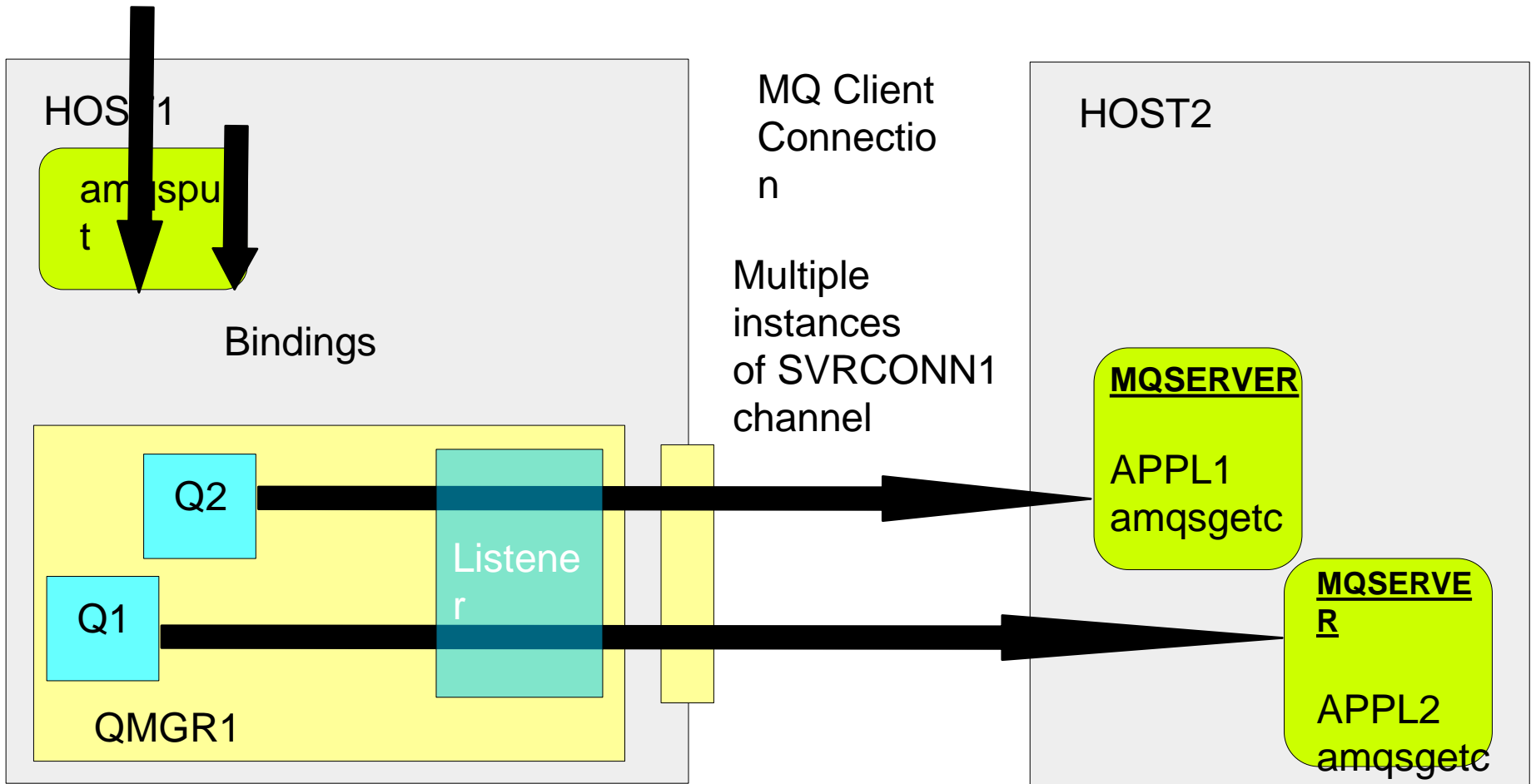
High availability sample programs

- amqsphac - put message, client mode
- amqsgnac - get message, client mode

4: mqclient.ini file (new in MQ 7.0)

- Let's continue to build on the scenario by having multiple Get client applications in HOST2:
APPL1 and APPL2
- We could use a separate MQSERVER for each of the applications.
- **Drawback:** If the queue manager or port changes, then all instances of MQSERVER will need to be updated.

Multiple clients using Client Connection



4: mqclient.ini file (new in MQ 7.0)

- To centralize the configuration of the connections the **client configuration file** can be used.
- The default name is: mqclient.ini
- The CHANNELS stanza is used to specify the channel name, host name, and port number.

- For example:

```
CHANNELS :
```

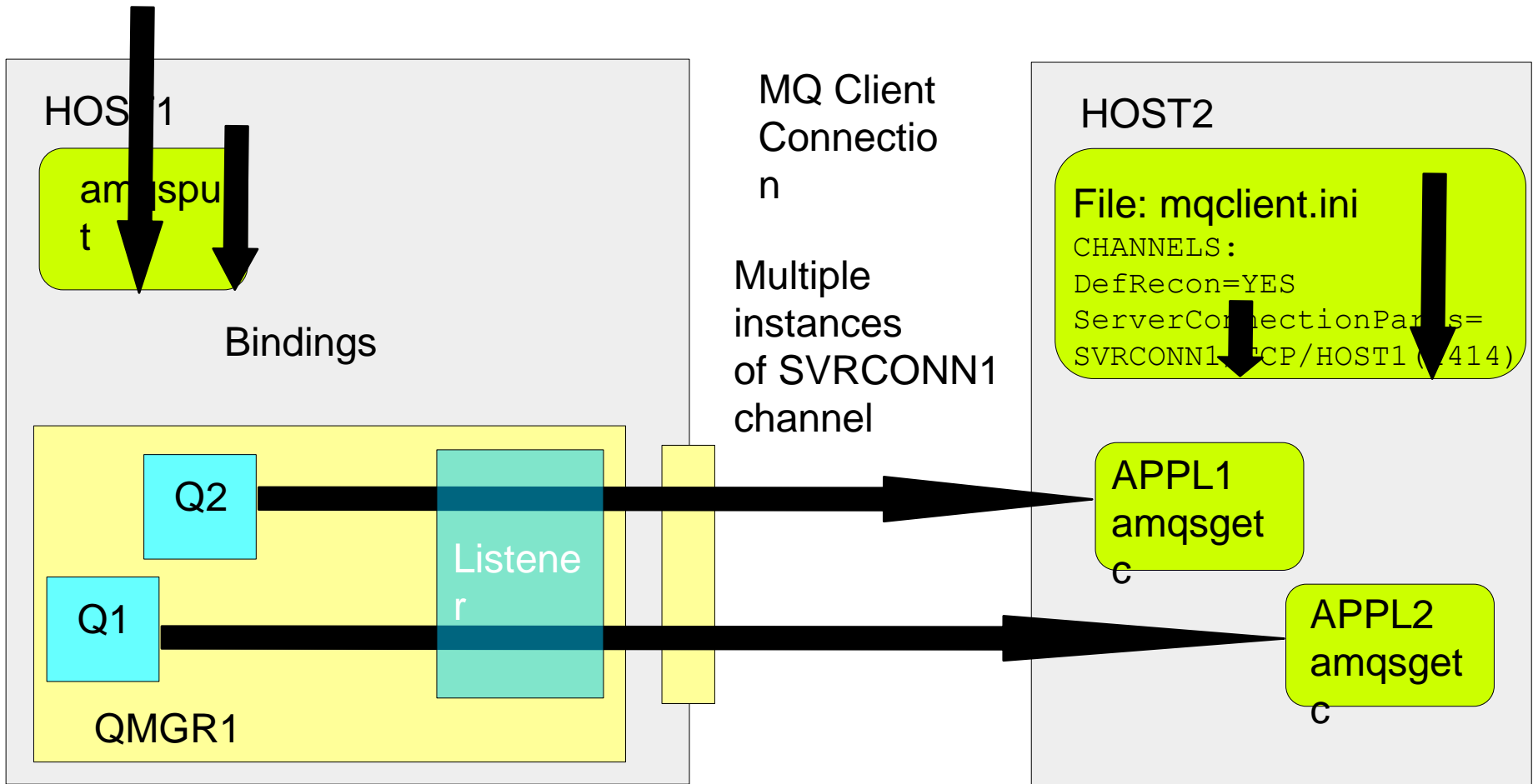
```
DefRecon=YES
```

```
ServerConnectionParms=
```

```
    SVRCONN1/TCP/HOST1 (1414)
```

- **Note:** DefRecon=YES allows for automatic reconnect.

Multiple clients using mqclient.ini



4: mqclient.ini file

- If the MQSERVER environment variable is set, then it supersedes the mqclient.ini.
- If you want to use mqclient.ini then you need to unset MQSERVER:
 - ▶ Windows: `set MQSERVER=`
 - ▶ UNIX: `unset MQSERVER`
- If there is a CHANNEL stanza in the mqclient.ini, then it supersedes the last method: CCDT

4: mqclient.ini file

- You could have multiple client configuration files with different names and/or different locations.
- You can use the following environment variable to specify which file to use:

MQCLNTCF

[http://publib.boulder.ibm.com/infocenter/wmqv7/v7r1/topic/com.ibm.mq.doc/cs13360 .htm](http://publib.boulder.ibm.com/infocenter/wmqv7/v7r1/topic/com.ibm.mq.doc/cs13360.htm)

Location of the client configuration file

- The format is a full URL. This means the file name might not necessarily be mqclient.ini and facilitates placing the file on a network attached file-system.

4: mqclient.ini file and QMgr Groups

- Similar to MQSERVER, the mqclient.ini can be configured for simple queue manager groups, such as a multi-instance queue manager:

- For example:

```
CHANNELS:
```

```
DefRecon=YES
```

```
ServerConnectionParms=
```

```
SVRCONN1/TCP/HOST1 (1414) , HOST2 (1414)
```



5: CCDT file

- Let's add more complexities to the scenario.
- You want to have multiple Get client applications located in multiple servers.
- You like some of the reasons mentioned in Page 6 such as connection balancing.
- You want a centralized location to administer the configuration artifact.
- You could use an mqclient.ini in a network drive and use the variable MQCLNTCF
- However, there are more functions in the CCDT that are not available in the mqclient.ini

5: CCDT file

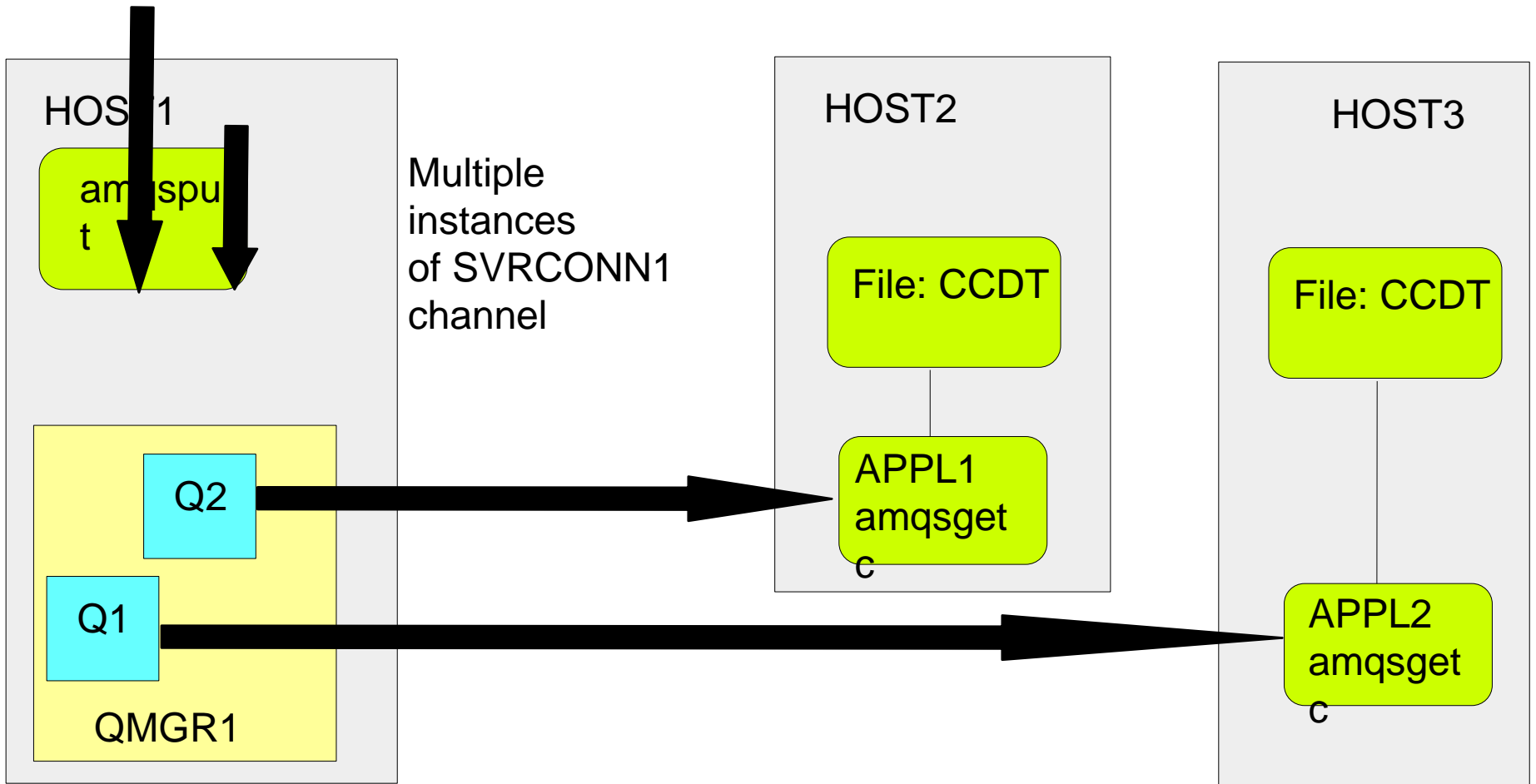
- You can use an MQ

Client Channel Definition Table (CCDT)

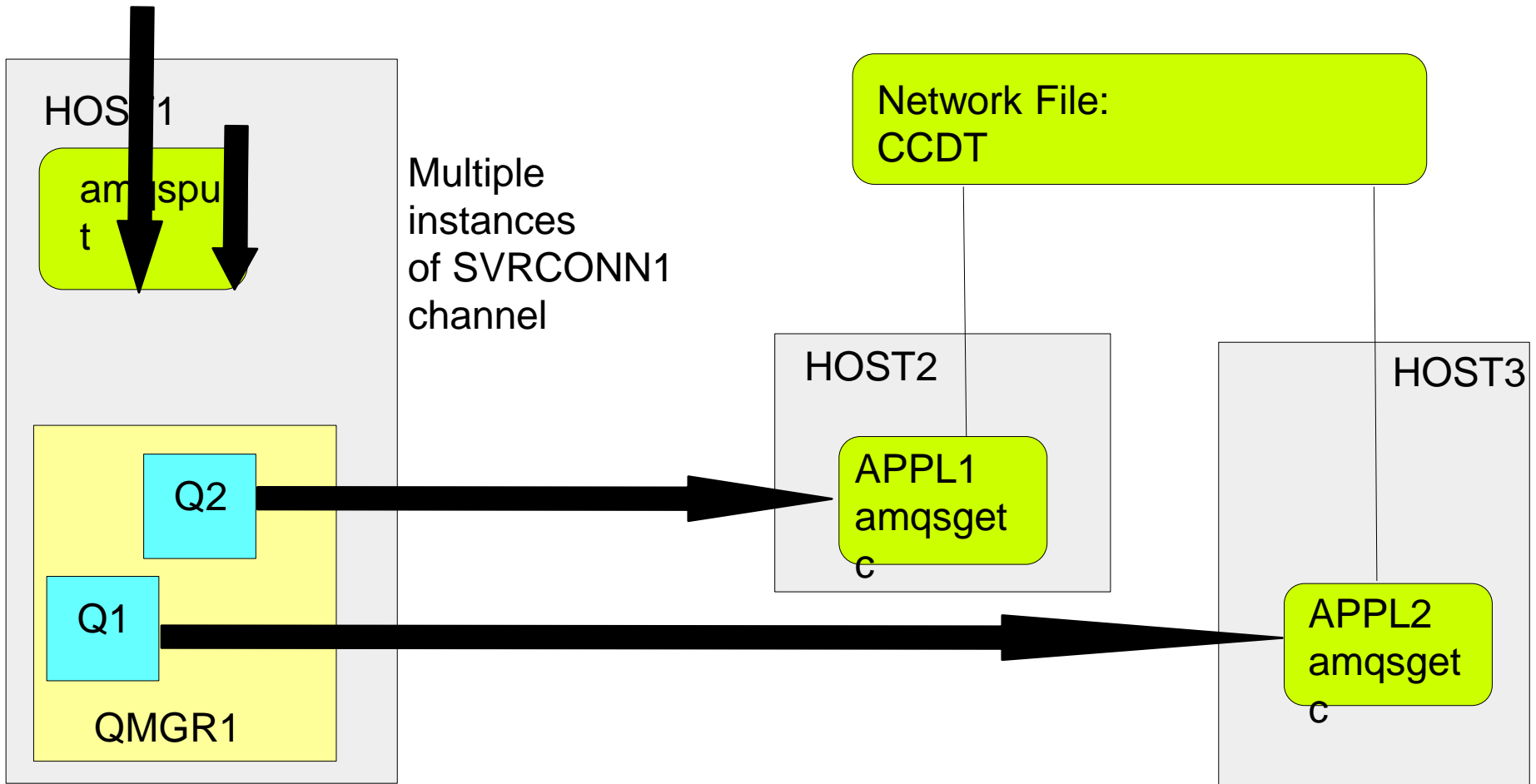
- Its purpose is to determine the channel definitions used by client applications to connect to the queue manager.
- It can be located in:
 - ▶ The same host as the client (multiple copies problem), or
 - ▶ In a network drive (single, shared copy)
 - See next 2 slides for topologies.



Multiple clients using separate CCDT files



Multiple clients using network file



5: CCDT file

- The CCDT is created when the queue manager is created.
- It has by default one client-connection channel:
 - ▶ `SYSTEM.DEF.CLNTCONN`
- **WARNING:** Do NOT delete the CCDT file.
 - ▶ It is NOT regenerated by the queue manager.
- The CCDT is a binary file.
 - ▶ It is NOT human readable.



5: CCDT file

- The CCDT is updated by the queue manager.
 - ▶ Client connection channels are added to the file when you use the `DEFINE CHANNEL` command, and
 - ▶ their definitions altered when you issue the `ALTER CHANNEL` command.
- The queue manager does not read the CCDT file.
 - ▶ That is, you cannot “import” client definitions from a CCDT into a queue manager.

Notes: Which Qmgr to use to create CCDT?

notes

- Because of the reasons in the previous 2 slides, it is recommended that you either:
 - a) Generate a new CCDT from a queue manager that is dedicated only for the creation and maintenance of the CCDT or
 - b) Use SupportPac MO72 (which is explained later in this presentation.)
- If there is considerable editing of the CCDT, then this file just keeps growing bigger and bigger over time, since there is no garbage collection and no “reset” to regenerate a new file via runmqsc or the MQ Explorer.

Notes: Connection Balancing - 1

notes

- The CCDTs in this presentation were created with the default values for the client-connection channels for the following attributes:
 - ▶ AFFINITY(PREFERRED)
 - ▶ CLNTWGHT(0)
- The default behavior is that there is no connection balancing and that the MQ client will choose the first channel name that is active from the list of channels in **alphabetical order of the channel name**.
- If you want to have a certain level of connection balancing, you will need to alter the following attributes for the channels in the CCDT:
 - ▶ AFFINITY(NONE)
 - ▶ CLNTWGHT(1)

Notes: Connection Balancing - 2

notes

- The following links provide more information about connection balancing:
- [http://publib.boulder.ibm.com/infocenter/wmqv7/v7r0/index.jsp?topic=/com.ibm.mq.csqz.af.doc/cs12700 .htm](http://publib.boulder.ibm.com/infocenter/wmqv7/v7r0/index.jsp?topic=/com.ibm.mq.csqz.af.doc/cs12700.htm)

Queue manager groups in the CCDT

- Question: Balance client connections across queue managers, with more clients connected to some queue managers than others.
- Answer: Define a queue manager group, and set the CLNTWGHT attribute on each client channel definition to distribute the connections unevenly.

- [http://publib.boulder.ibm.com/infocenter/wmqv7/v7r0/index.jsp?topic=/com.ibm.mq.csqz.af.doc/cs12700 .htm](http://publib.boulder.ibm.com/infocenter/wmqv7/v7r0/index.jsp?topic=/com.ibm.mq.csqz.af.doc/cs12700.htm)

Client channel weight (CLNTWGHT)

- Specify a value in the range 0 – 99. The default is 0.
- A value of 0 indicates that no connection balancing is performed and applicable definitions are selected in alphabetical order. To enable connection balancing choose a value in the range 1 - 99 where 1 is the lowest weighting and 99 is the highest.
- The distribution of connections between two or more channels with non-zero weightings is approximately proportional to the ratio of those weightings. This distribution is not guaranteed

Scenarios for Qmgr Groups using CCDT

- 1 - Simplest case, using same QMNAME for a single queue manager (QM3 in Host-1)
- 2 - Simplest case, using same QMNAME for a single qmgr (QM3's host renamed: Host-2)
- 3 - Multi-Instance Queue Manager
- 4 - QMgr Group: ' ' Default Group (1 blank character)
- 5 - QMgr Group: '*' Default Group, application uses 1 asterisk, but QMNAME is ' ' (1 blank character)
- 6 - QMgr Group: *QMGROUP1

Notes: Categories of Queue Manager Groups

n
o
t
e
s

- The following categories of Queue Manager Groups might be useful to remember their usage.
- **MQCONN QMNAME('NAME') == CCDT QMNAME('NAME') == QMgr QMNAME('NAME')**
 - 1 - Simplest case, using same QMNAME for a single queue manager (QM3 in Host-1)
 - 2 - Simplest case, using same QMNAME for a single qmgr (QM3's host renamed: Host-2)
 - 3 - Multi-Instance Queue Manager
- **MQCONN QMNAME(' ') == CCDT QMNAME(' ') != QMgr QMNAME('???')**
 - 4 - QMgr Group: ' ' Default Group (1 blank character)
- **MQCONN QMNAME('*') == CCDT QMNAME(' ') != QMgr QMNAME('???')**
 - 5 - QMgr Group: '*' Default Group, Application uses 1 asterisk
- **MQCONN QMNAME('*GROUP') == CCDT QMNAME('GROUP') != QMgr QMNAME('???')**
 - 6 - QMgr Group: *QMGROUP1

Scenario 1- simplest case

- Simplest case, using same QMNAME for a single queue manager (QM3 in Host-1)
- Queue Manager Group: 'QM3'
- Application uses: 'QM3'
- Sample Put Invocation: amqsphac Q1 QM3



MQCONN QMNAME('NAME') == CCDT QMNAME('NAME') == QMgr
QMNAME('NAME')

Scenario 1- Creating channels via runmqsc

- There is ONLY 1 entry in the CCDT with QMNAME 'QM3':

- To Queue Manager: QM3 (in host aemaix1) via client connection channel:

```
DEFINE CHANNEL(CCC3) CHLTYPE(CLNTCONN) +  
TRPTYPE(TCP) CONNAME('aemaix1.x.com(1451)') +  
QMNAME(QM3)
```

- Needs a corresponding server-connection channel:

```
DEFINE CHANNEL(CCC3) CHLTYPE(SVRCONN) +  
TRPTYPE(TCP)
```



Notes: Using runmqsc to create channels

notes

- Login as an MQ Administrator in the host that has the queue manager with the CCDT that will be copied to other hosts.

- Issue: `runmqsc QmgrName`

- Inside `runmqsc`, enter:

```
DEFINE CHANNEL(CCC3) CHLTYPE(CLNTCONN) +  
TRPTYPE(TCP) CONNAME('aemaix1.x.com(1451)') QMNAME(QM3)
```

```
DEFINE CHANNEL(CCC3) CHLTYPE(SVRCONN) TRPTYPE(TCP)
```

```
END
```

Notes: Using a text file with runmqsc

Notes

- The examples in this presentation are short and simple.
- However, if you have to create dozens of channels, it could be cumbersome to type each channel by hand when manually interacting with runmqsc (or when using the MQ Explorer – next section).
- Instead, you could create a text file with all the desired channel definitions and then invoke runmqsc to use the entries of that file.
- This file is known as an MQ Script Command (MQSC) and a good practice is to use the suffix: .mqsc
- You can run the MQSC script by redirecting the input and capturing the output into another file:

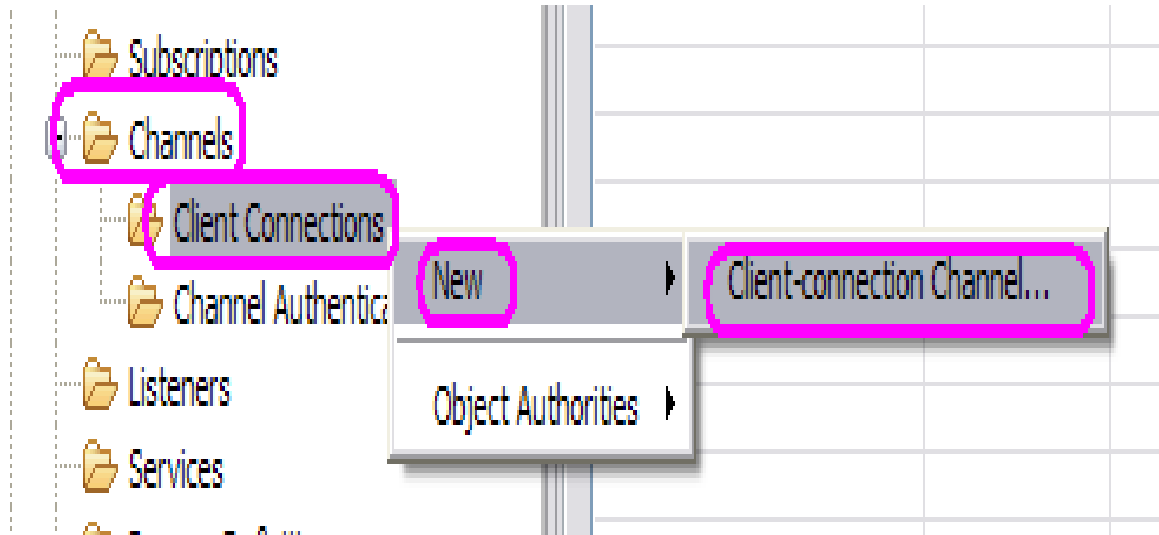
```
runmqsc QMgrName < filename.mqsc > filename.out
```

- Inside the file, you can use an asterisk in the first column to indicate a comment.
- Also, in case that you need to rerun the script, it is a good practice to use the REPLACE token at the end of each DEFINE command. For example:

```
▪ * This is a comment - Scenario 1 - Simplest case  
DEFINE CHANNEL(CCC3) CHLTYPE(SVRCONN) TRPTYPE(TCP) REPLACE
```


Creating CLTCONN via MQ Explorer

- Login as MQ Administrator
- Start the MQ Explorer
- Under the desired Qmgr, select:
- Channels >
Client Connections >
New >
Client-connection Channel



Creating CLTCONN

- Enter the name for the channel.
- It must match the same name of the Server-Connection Channel
- Click Next

New Client-connection Channel

Create a Client-connection Channel
Enter the details of the object you wish to create

Name:
CCC3

Select an existing object from which to copy the attributes for the new object.
SYSTEM.DEF.CLNTCONN

< Back **Next >** Finish Cancel

Creating CLTCONN

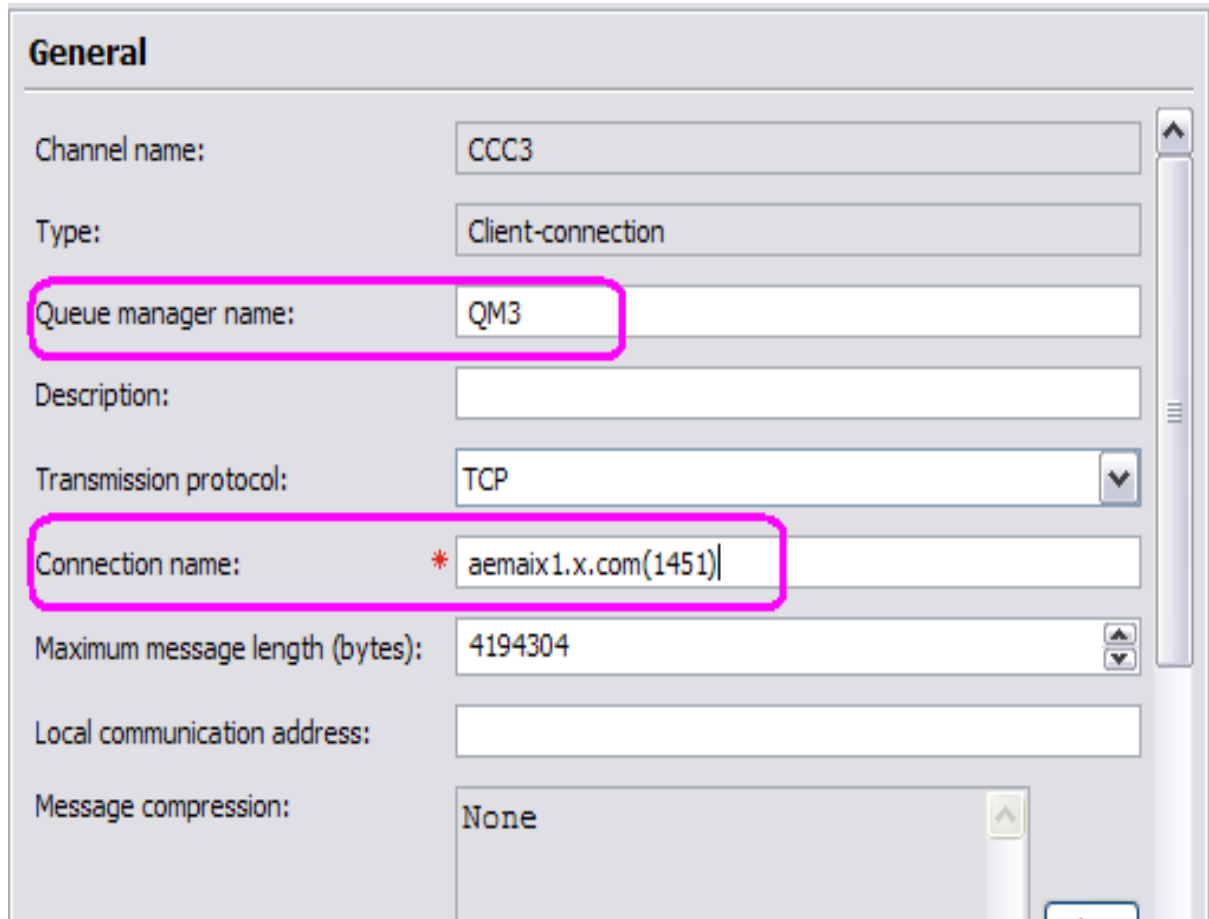
- Enter the name for the queue manager.

NOTE:

- It is REALLY the Queue Manager Group!

- Enter Connection Name.

- Click Finish



General

Channel name: CCC3

Type: Client-connection

Queue manager name: QM3

Description:

Transmission protocol: TCP

Connection name: * aemaix1.x.com(1451)

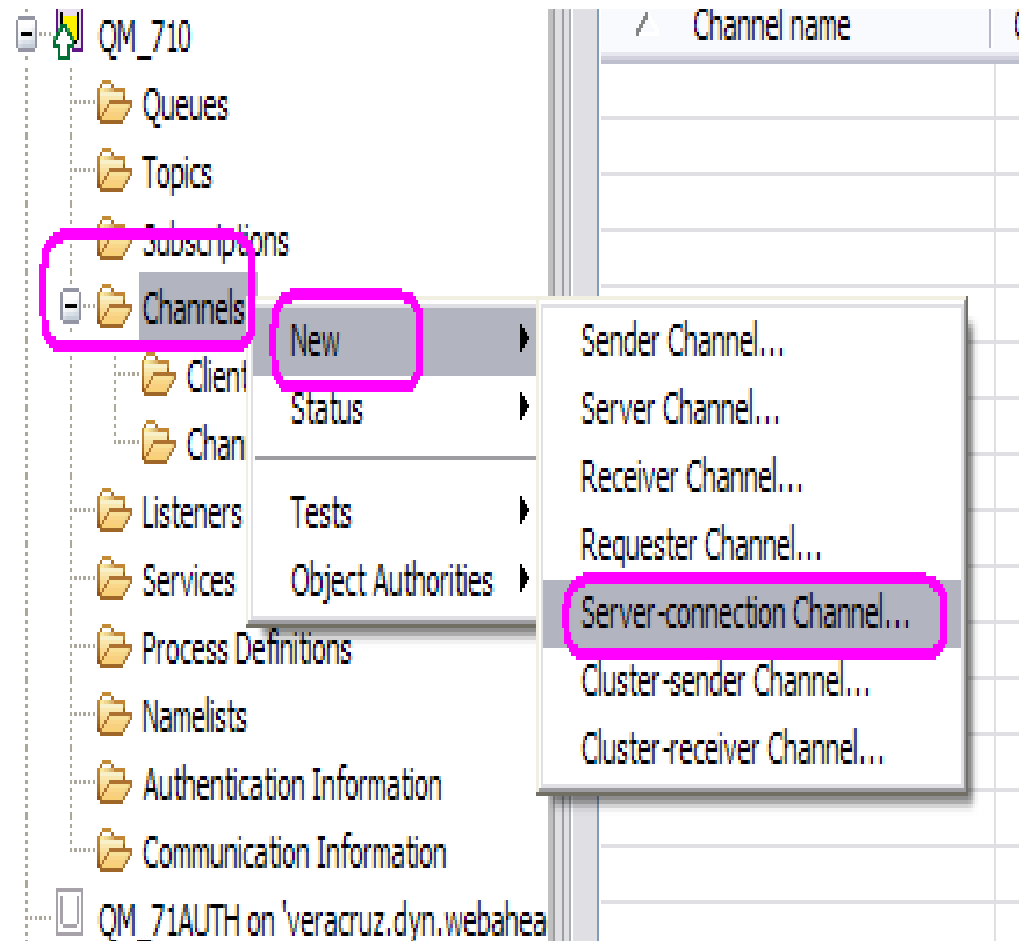
Maximum message length (bytes): 4194304

Local communication address:

Message compression: None

Creating SVRCONN channel

- Login as MQ Administrator
- Start the MQ Explorer
- Under the desired Qmgr, select:
 - Channels >
New >
Server-Connection Channel



Creating a SVRCONN channel

- Enter the name for the channel.
- It must match the same name of the Client-Connection Channel
- Click Finish

New Server-connection Channel

Create a Server-connection Channel
Enter the details of the object you wish to create

Name:
ccc3

Select an existing object from which to copy the attributes for the new object.
SYSTEM.DEF.SVRCONN

< Back Next > **Finish** Cancel

Location of CCDT – local host

- When a client-connection channel is created, then the queue manager will update the CCDT file.
- Note: When a server-connection channel is created, the CCDT file is NOT updated.
- This CCDT file is named:
 - ▶ AMQCLCHL.TAB
- It is located at:
- Windows:
 - ▶ MQ_DATA_DIR\qmgrs\QMgrName\@ipcc
- Unix: /var/mqm/qmgrs/QMgrName/@ipcc

Location of CCDT – copying to remote host

- You need to copy this BINARY file from the location of the local host (shown in previous slide) into a designated location in the remote host or network drive.
- You need to transfer this file as BINARY.
- If you transfer it as ASCII (the default by some FTP tools) then it will arrive corrupted!
- In this scenario, the remote location is:
`/var/mqm/ccdt-files`

Environment variables to locate the CCDT

- There are 2 environment variables that will assist the application to locate the desired CCDT file.

MQCHLLIB => The Directory (but not file name)

MQCHLTAB => The file name

Full File Path: MQCHLLIB + MQCHLTAB

- The default value for MQCHLTAB is:
 - ▶ AMQCLCHL.TAB
- Thus, in practice, need to specify only MQCHLLIB

Environment variables to locate the CCDT

- Example for this presentation:

- UNIX:

```
export MQCHLLIB=/var/mqm/ccdt-files
```

- Windows:

```
set MQCHLLIB=C:\var\mqm\ccdt-files
```

- Full File Path:

```
/var/mqm/ccdt-files/AMQCLCHL.TAB
```



CCDT file is last in precedence (5th)

- The use of the Client Channel Definition Table (CCDT) is the last method to be considered in the order of precedence.
- Thus, if MQSERVER or MQCLNTCF are defined, then the CCDT will NOT be used.

- You will need to unset these variables:

- UNIX:

```
unset MQSERVER
```

```
unset MQCLNTCF
```

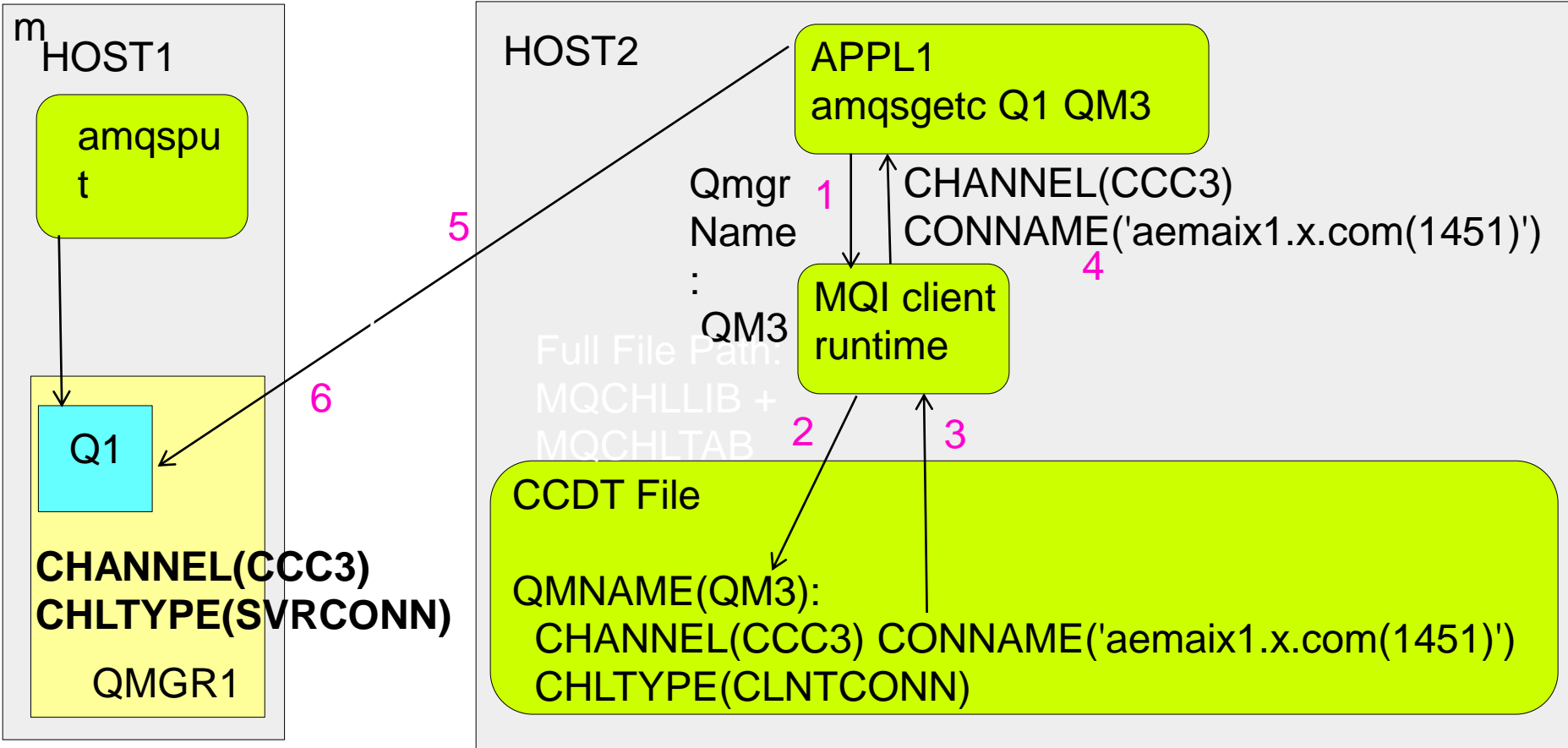
Notes: Finding the Client Connection Channel

Notes

- The contents of the QMgrName parameter of the MQCONN or MQCONNX call determines which queue manager the client connects to.
- The CCDT file is accessed by the MQ client to determine the channel an application will use.
- In this simplest case (see next figure)
 - 1) The application specifies the QmgrName as QM3,
 - 2) The MQI client locates the CCDT. It uses the full path established by these environment variables:
MQCHLLIB + MQCHLTAB
 - 3) An entry is found in the CCDT for QMNAME(QM3):
CHANNEL(CCC3) CONNAME('aemaix1.x.com(1451)')
 - 4) The MQI client returns the CHANNEL and CONNAME to the application.
 - 5) The application uses this client-connection channel to talk to ...
 - 6) ... the corresponding server-connection channel CCC3

Runtime Interaction with CCDT

aemaix1.x.co



Cannot use runmqsc to view a CCDT

- Interesting Problem:
- Your company may have several CCDT files with different names and for different purposes
- **Question:** how do you look at the contents of the CCDT files to verify that it has the desired data?
- **Answer:**
- You cannot use runmqsc or the MQ Explorer.
- But you can use the SupportPac: MO72

SupportPac to view contents of CCDT

- To view the contents of CCDT file, you can use:

<http://www.ibm.com/support/docview.wss?uid=swg24007769>

Short URL: <http://ibm.co/SupptPacMO72>

MO72: MQSC Client for WebSphere MQ

“mqsc” is a command line tool for issuing MQSC commands to a queue manager or client.

- Note: It is a Category 2 SupportPac: provided in good faith and AS-IS (no supported by MQ Support Team)



How to use the "mqsc" utility - 1

notes

- 1) Visit the following web page and download the zip file with the code and the PDF file for the documentation:

<http://www.ibm.com/support/docview.wss?uid=swg24007769>

MO72: MQSC Client for WebSphere MQ

- Let's assume that it is downloaded into:
 - ▶ Linux Intel® 32-bit: /downloads/mq/mo72-mqsc
 - ▶ Windows: C:\MQ-SupportPac\MO72 MQSC Client
- 2) Change to the directory of the platform:
 - ▶ Linux Intel 32-bit: `cd /downloads/mq/mo72-mqsc/Linux Intel`
 - ▶ Windows: `cd C:\MQ-SupportPac\MO72 MQSC Client\Windows`
- 3) For Unix, you must give execution permission:
 - ▶ `$ chmod u+x mqsc`

How to use the "mqsc" utility - 2

notes

- 4) Run the command which uses the default CCDT file name of AMQCLCHL.TAB.

You need to specify the following flags:

-n => run in client mode to access the CCDT

-t full-path-ending-with-slash => Full directory location

NOTICE that you MUST end the string with a directory separator, or slash!

INCORRECT usage:

For example, the last character of the path name is NOT a directory separator

```
mqsc -n -t C:\var\mqm\Qmgrs\QM_ANGELITO\@ipcc
```

Current channel table file not found

```
Can not find file 'C:\var\mqm\Qmgrs\QM_ANGELITO\@ipcc'
```

```
>
```

CORRECT usage:

You need to ensure that the end of the path name has a directory separator.

Notice that you get a prompt: >

Linux example:

```
$ mqsc -n -t /mqexport/701/data/QMMI1/@ipcc/
```

```
>
```


How to use the "mqsc" utility - 3

notes

Windows example:

```
C:\MQ-SupportPac\MO72 MQSC Client, reads CCDT\Windows>  
mqsc -n -t C:\var\mqm\Qmgrs\QM_ANGELITO\@ipcc\  
>
```

- 5) Issue the following to show the contents.

Note: only showing selected entries to keep this document shorter.

```
> display channel(*)
```

AMQ8414: Display Channel details.

```
CHANNEL(QMMI1) CHLTYPE(CLNTCONN) DESCR( )
```

```
TRPTYPE(TCP) CONNAME(veracruz.x.com(1421)
```

```
,cbeech.x.com(1421)) QMNAME(QMMI1)
```

```
LOCLADDR( ) USERID( ) PASSWORD( )
```

```
MODENAME( ) TPNAME( ) MAXMSGL(4194304)
```

```
... SSLCIPH( ) SSLPEER( ) SHARECNV(10)
```

```
CLNTWGHT(0) AFFINITY(PREFERRED) ALTDATE(2010-05-26)
```

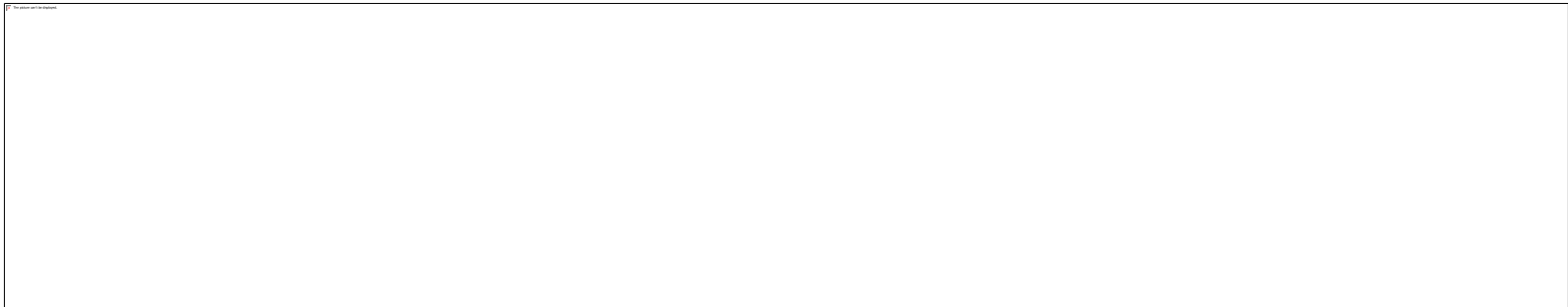
```
> display channel(*) version
```

AMQ8414: Display Channel details.

```
CHANNEL(QMMI1) CHLTYPE(CLNTCONN) VERSION(7.0)
```

Scenario 2- simplest case, renaming host

- Using same QMNAME for a single queue manager (QM3: Host-1 was renamed to Host-2)
- Queue Manager Group: 'QM3'
- Application uses: 'QM3'
- Sample Put Invocation: amqsphac Q1 QM3



**MQCONN QMNAME('NAME') == CCDT QMNAME('NAME') == QMgr
QMNAME('NAME')**

Scenario 2- Creating channels

- There is ONLY 1 entry in the CCDT with QMNAME 'QM3':

- 1: To Queue Manager: QM3 (Host-1 was renamed **Host-2**) via channel:

```
DEFINE CHANNEL(CCC3) CHLTYPE(CLNTCONN) +  
TRPTYPE(TCP) CONNAME('aemaix2.x.com(1451)') +  
QMNAME(QM3) REPLACE
```

- The **REPLACE** keyword allows to overwrite an existing definition.

Scenario 2- Updating channel

- If the host that has QM3 is renamed from Host-1 to Host-2, then only the CCDT needs to be updated to reflect this network change:

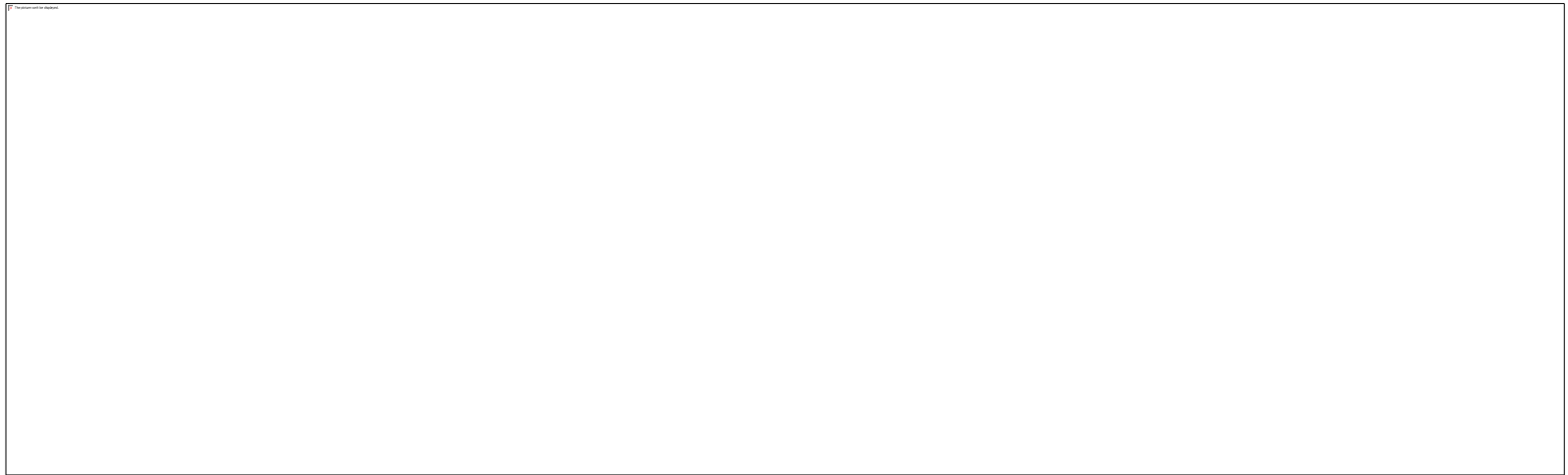
From: CONNAME('aemaix1.x.com(1451)')

To: CONNAME('aemaix2.x.com(1451)')

- There is no need to modify the client application, which continues using QM3.
- Note: You will need to copy the updated CCDT file from the originating queue manager into the host of the client appl.

Scenario 3 – Multi-Instance Queue Manager

- Queue Manager Group: 'QMMI1'
- Application uses: 'QMMI1'
- Sample Put Invocation: amqsphac Q1 QMMI1



MQCONN QMNAME('NAME') = = CCDT QMNAME('NAME') == QMgr
QMNAME('NAME')

Scenario 3 – creating channels / choosing

n

o

t

e

s

- There is only 1 entry in the CCDT with QMNAME 'QMMI1'.
- Notice the use of multiple entries for host(Port).

```
DEFINE CHANNEL(CCC1) CHLTYPE(CLNTCONN) TRPTYPE(TCP) +
CONNAME('veracruz.x.com(1421),cbeech.x.com(1421)') QMNAME(QMMI1)
```

- Define SVRCONN in qmgr in the active instance (veracruz or cbeech):

```
DEFINE CHANNEL(CCC1) CHLTYPE(SVRCONN) TRPTYPE(TCP)
```

- **Choosing Channel**

▪ Notice that because the group name is the same as the single entry queue manager, it is OK to not use the asterisk:

amqsphac Q9 QMMI1

▪ Do not get confused that there are 2 boxes in the figure for the same queue manager name (QMMI1), it is still a group with a single entry but for multiple-instances.

▪ That is also the reason for a single definition for the client-connection and server-connection channel: it is the same queue manager.

Scenario 4 – Default Group, ' ' (1 blank)

- Queue Manager Group: ' ' (Default group)
- Application uses: ' ' (notice: 1 blank character)
- The use of ' ' (single space) is suitable for working directly with the MQCONN call, and not necessarily via samples.

- Similar to Scenario 5.

MQCONN QMNAME(' ') == CCDT QMNAME(' ') != QMgr
QMNAME('???')

Scenario 4 – creating CLTCONN channels

n
o
t
e
s

- There are 3 entries in the CCDT with QMNAME ' ' (1 blank character):

```
DEFINE CHANNEL(DEFAULT.A) CHLTYPE(CLNTCONN) +  
TRPTYPE(TCP) CONNAME('veracruz.x.com(1431)') QMNAME(' ')
```

```
DEFINE CHANNEL(DEFAULT.B) CHLTYPE(CLNTCONN) +  
TRPTYPE(TCP) CONNAME('aemaix1.x.com(1451)') QMNAME(' ')
```

```
DEFINE CHANNEL(DEFAULT.C) CHLTYPE(CLNTCONN) +  
TRPTYPE(TCP) CONNAME('aemtux1.x.com(1434)') QMNAME(' ')
```


Scenario 4 – creating SVRCONN channels

n
o
t
e
s

- **Define SVRCONN in qmgr in veracruz:**

```
DEFINE CHANNEL(DEFAULT.A) CHLTYPE(SVRCONN) TRPTYPE(TCP)
```

- **Define SVRCONN in qmgr in aemaix1:**

```
DEFINE CHANNEL(DEFAULT.B) CHLTYPE(SVRCONN) TRPTYPE(TCP)
```

- **Define SVRCONN in qmgr in aemtux1:**

```
DEFINE CHANNEL(DEFAULT.C) CHLTYPE(SVRCONN) TRPTYPE(TCP)
```

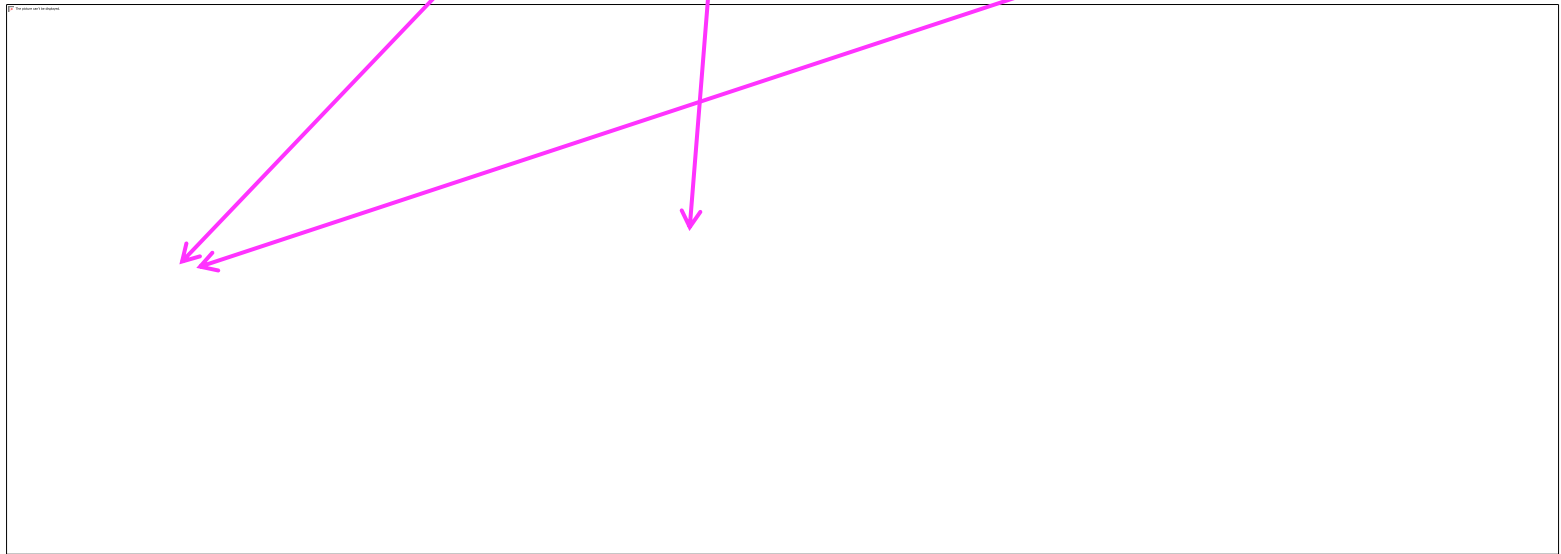
Scenario 4 – choosing the channel

notes

- This scenario illustrates use of the default group.
- In this case the client application passes a blank ' ' or an asterisk ('*' - see next Scenario), as the QmgrName parameter to its MQCONN or MQCONNX MQI call.
- By convention in the client channel definition, a blank QMNAME attribute signifies the default queue manager group and either a blank or asterisk QmgrName parameter matches a blank QMNAME attribute.
- In this example the default queue manager group has client channel connections to all the queue managers. By selecting the default queue manager group the application might be connected to any queue manager in the group.
- Note: The default group is different to a default queue manager, although an application uses a blank QmgrName parameter to connect to either the default queue manager group or to the default queue manager.
- The concept of a default queue manager group is only relevant to a client application, and a default queue manager to a server application.

Scenario 5 - Default group ' ', Appl uses '*'

- Similar to Scenario 4, but application uses *
- Queue Manager Group: ' ' (1 blank, Default Group)
- Application uses: '*' (notice: 1 asterisk)
- Sample Put Invocation: amqsphac Q1 *



MQCONN QMNAME('*') == CCDT QMNAME(' ') != QMgr

QMNAME('???')

Scenario 5 – choosing the channel

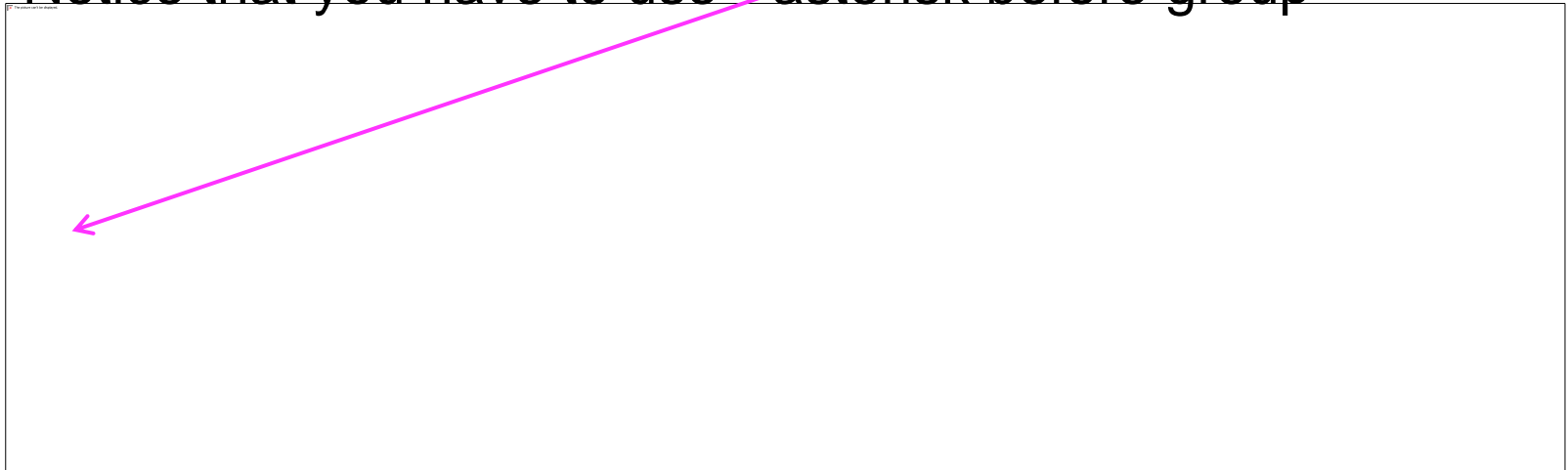
notes

- Note: Same channels as in Scenario 4. Using Default Group: QMNAME(' ')
- The use of ' ' (single space) is suitable for working directly with the MQCONN call, and not necessarily via samples that interact with the command prompt, such as amqsphac (in which you will need to use the asterisk '*')
- For this scenario, the following queue managers are running:
 - QM1 in Veracruz
 - QM3 in aemaix1
 - QM4 in aemtux1
- Invoking the sample: amqsphac Q1 *
- In this case, the messages were received by QM1 in Veracruz, because its channel name is first in **alphabetical order**: DEFAULT.A
- Stop Veracruz. Try again.
- The messages are received by QM3 in aemaix1, because its channel name is next in the alphabetical list, DEFAULT.B, and the previous channel is not operational.

Scenario 6 - *QMGROUP1

- Queue Manager Group: 'QMGROUP1'
- Application uses: '*QMGROUP1'
- Sample Put Invocation: amqsphac Q1 *QMGROUP1

Notice that you have to use * asterisk before group



```
MQCONN QMNAME('*GROUP') == CCDT QMNAME('GROUP') != QMgr  
QMNAME('???')
```

Scenario 6 – creating channels

n

o

t

e

s

- There are 2 entries in the CCDT with QMNAME 'QMGROUP1':

```
DEFINE CHANNEL(QMGROUP1.A) CHLTYPE(CLNTCONN) +  
TRPTYPE(TCP) CONNAME('aemaix1.x.com(1422)') +  
QMNAME(QMGROUP1)
```

```
DEFINE CHANNEL(QMGROUP1.B) CHLTYPE(CLNTCONN) +  
TRPTYPE(TCP) CONNAME('veracruz.x.com(1423)') +  
QMNAME(QMGROUP1)
```

- Define SVRCONN in qmgr in aemaix1:

```
DEFINE CHANNEL(QMGROUP1.A) CHLTYPE(SVRCONN) +  
TRPTYPE(TCP)
```

- Define SVRCONN in qmgr in veracruz:

```
DEFINE CHANNEL(QMGROUP1.B) CHLTYPE(SVRCONN) +  
TRPTYPE(TCP)
```

Scenario 6 – choosing the channel

notes

- In this scenario, the client application passes a queue manager name prefixed with an asterisk, ***QMGROUP1** as the QmgrName parameter to its MQCONN or MQCONNX MQI call.
- The WebSphere MQ client selects the matching queue manager group, QMGROUP1.
- This group contains two client connection channels, and the WebSphere MQ client tries to connect to any queue manager using each channel in turn. In this example, the WebSphere MQ client needs to make a successful connection; the name of the queue manager that it connects to does not matter.
- The rule for the order of making connection attempts is the same as before: in **alphabetical order**. The only difference is that by prefixing the queue manager name with an asterisk, the client indicates that the name of the queue manager is not relevant.
- The MQCONN or MQCONNX call issued by the client application succeeds when a connection is established to a running instance of any qmgr.

What happens if asterisk is not used?

n

o

t

e

s

- **Question:** What happens when the leading asterisk is not specified?

- **Answer:**

When the queue manager group has multiple entries and when the name of the group is NOT specified with a leading asterisk, then the attempts to connect to the queue manager will fail and a typical return code is:

2058 MQRC_Q_MGR_NAME_ERROR.

- For example, QMGROUP1 is a queue manager group with multiple entries, thus a leading asterisk is required; if the asterisk is omitted, then the connection will fail because it will try to connect to a queue manager with the literal name of QMGROUP1

- C:\> amqsphac Q9 QMGROUP1

Sample AMQSPHAC start

MQCONNX ended with reason code 2058

Sample AMQSPHAC end

Migration and Compatibility

- The CCDT has changed from V5.3/V6.0 to V7.
- Older clients must continue to use existing CCDTs.
- Newer clients are able to understand older CCDTs.
- For an application to use a newer version CCDT, you must update the MQ client.
- When the queue manager that is used to update the CCDT file is upgraded (say from V6 to V7), then the corresponding CCDT file in the queue manager's @ipcc directory is also upgraded.

Migration and Compatibility

- If you have 5.3, 6 and 7.0 clients in your network, then when editing the CCDT, you need to either:
 - - a) Edit the CCDT using the MQ queue manager at the oldest version (in this example: V5.3), or
 - b) Use the MO72 SupportPac, see section: “Channel Definition Version”
mqsc -n -v6.0 “Write all channel records at the WebSphere MQ Version 6.0 level. Versions supported are : 5.0, 5.1, 5.2, 5.3, 6.0 and 7.0”

JMS and CCDTURL - 1

- In MQ V6 a new JMS attribute was introduced.
- It can be configured with a URL where the CCDT resides.
- It is called:
 - ▶ CCDTURL
- It is specified in the Connection Factory and can be looked up just like any other JMS attribute.

JMS and CCDTURL - 2

- For details on the syntax for the “file” protocol see: http://en.wikipedia.org/wiki/File_URI_scheme
- Examples:
- UNIX: Same /directory/AMQCLCHL.TAB file:
<file://localhost/directory/AMQCLCHL.TAB>
<file:///directory/AMQCLCHL.TAB> (local, 3 slashes)
- Windows:
<file://hostname/directory/AMQCLCHL.TAB> (network)
<file:///C:/directory/AMQCLCHL.TAB> (local)



JMS and CCDTURL in WAS

- Example from the WAS Administrative Console:
file:///var/mqm/qmgrs/QM_MDB/@ipcc/AMQCLCHL.TAB

Specify client channel definition table

Specify client channel definition table

Enter information about the client channel definition table that will be used to establish a connection to WebSphere MQ.

Step 1: Configure basic attributes

Step 2: Select connection method

→ Step 2.1: Specify client channel definition table

Step 3: Test connection

Step 4: Summary

Specify client channel definition table

Enter information about the client channel definition table that will be used to establish a connection to WebSphere MQ.

* Client channel definition table URL

Queue manager

Previous Next Cancel

Notes: Tracing

notes

- <http://www.ibm.com/support/docview.wss?rs=171&uid=swg21321254>
- Determining how a client established a connection to the server: environment variables or MQCONNX
- V7:

| | | | | |
|--|-----------------|--------|--------------|-------------------------------|
| 00044C9A | 02:44:17.083028 | 8296.1 | RSESS:000001 | Using |
| ChannelDefinitionDirectory / MQCHLLIB value of C:\Program Files\IBM\WebSphere MQ | | | | |
| 00044C9B | 02:44:17.083047 | 8296.1 | RSESS:000001 | Using ChannelDefinitionFile / |
| MQCHLTAB value of AMQCLCHL.TAB | | | | |
- V6:

| | | | |
|------------------------|-----------------|--------|--|
| 00000244 | 17:48:31.832381 | 5200.1 | Using MQCHLLIB env var of C:\Program |
| Files\IBM\WebSphere MQ | | | |
| 00000246 | 17:48:31.832408 | 5200.1 | Using env var MQCHLTAB for name of channel file of |
| AMQCLCHL.TAB | | | |

Related techdocs

- This presentation is based on these techdocs:
- The following techdoc is for C-based clients:
<http://www.ibm.com/support/docview.wss?uid=swg27020848>
Using a Client Channel Definition Table (CCDT) in WebSphere MQ V7 for Queue Manager Groups
- The following is for JMS:
<http://www.ibm.com/support/docview.wss?uid=swg27020862>
Using a CCDT file to connect to multiple WebSphere MQ queue managers using JMS

Other References - 1

- <http://www.ibm.com/support/docview.wss?rs=171&uid=swg27020701>

Using a CCDT file to connect to WebSphere MQ multi-instance queue managers from WebSphere Application Server V7

- <http://www.ibm.com/support/docview.wss?uid=swg27020901>

Using Custom Property and CCDT File to Connect to WebSphere MQ Multi-Instance Queue Managers from WebSphere Application Server V7
Webinar recording: audio MP3 and PDF file



Other References - 2

<http://www.ibm.com/support/docview.wss?rs=171&uid=swg27023313>

Exploiting the Automatic Client Reconnect feature in WebSphere MQ JMS 7.0.1

<http://www.ibm.com/support/docview.wss?rs=171&uid=swg27017882>

Exploiting the Automatic Client Reconnect feature introduced in WebSphere MQ 7.0.1 – MQI (C Code)



Additional WebSphere Product Resources

- Learn about upcoming WebSphere Support Technical Exchange webcasts, and access previously recorded presentations at:
http://www.ibm.com/software/websphere/support/supp_tech.html
- Discover the latest trends in WebSphere Technology and implementation, participate in technically-focused briefings, webcasts and podcasts at:
<http://www.ibm.com/developerworks/websphere/community/>
- Join the Global WebSphere Community:
<http://www.websphereusergroup.org>
- Access key product show-me demos and tutorials by visiting IBM® Education Assistant:
<http://www.ibm.com/software/info/education/assistant>
- View a webcast replay with step-by-step instructions for using the Service Request (SR) tool for submitting problems electronically:
<http://www.ibm.com/software/websphere/support/d2w.html>
- Sign up to receive weekly technical My Notifications emails:
<http://www.ibm.com/software/support/einfo.html>

Questions and Answers

