Introduction to WebSphere MQ Clients

Morag Hughson
IBM Hursley
hughson@uk.ibm.com

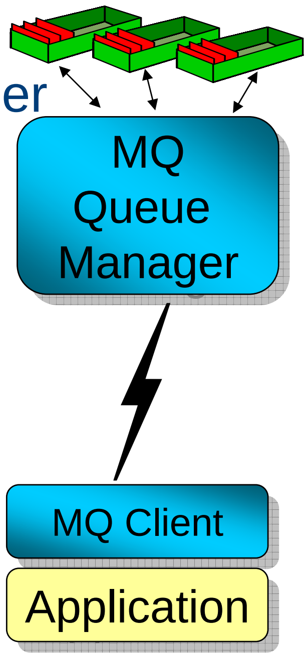
Agenda

- What are the MQ clients ?
- The MQ client and how it works
- How to connect a client to a server
  - Channel Table Configuration
- What facilities are available to clients
  - Transactions
  - Global Transactions
  - Security
  - Exits

What is a client ?

- Allows access to messaging API on a different machine than the queue manager
  - Simpler administration
  - Same programming capabilities (almost)
  - Cheaper
    - Free in most cases
- However.....

No network – No messaging.



SHARE in Anaheim 2011

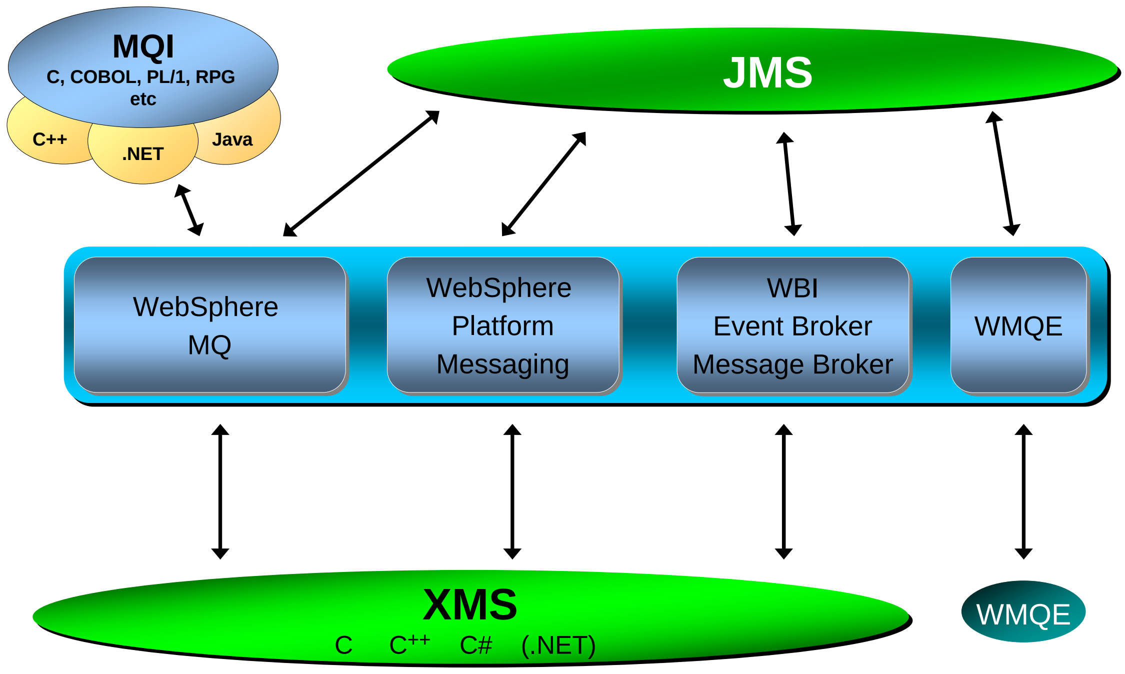
What is a client ?

N O T E S

- In this world of client/server architectures, thin-clients, thick-clients and network clients the word client is a much overused word which means different things to different people.
- For the purposes of this presentation a ‘client’ is merely an application which is issuing messaging APIs but there is a network connection between it and the queues and/or destinations.
- In most cases this means the client application is on a different physical machine that the server hosting the queues/destinations but this is by no means mandatory. It is perfectly legal, and sometimes necessary, to run a client application on the same machine as the queue manager server.
- The advantages of using a client architecture is that there is no requirement to have servers defined and managed on all the outlying machines. An enterprise may well have thousands of applications wishing to do messaging but using clients the administration can be limited to a few well controlled machines. The disadvantage is that if the network is down for any reason the applications will not be able to connect to the servers and do any messaging. It should also be noted that messaging in a client applications is generally slower than in a locally connected application.

SHARE in Anaheim 2011

Messaging Clients



Messaging Clients

N O T E S

- There are a number of messaging clients designed to suit different environments, different programming languages and different programming languages.
- In the MQ world there are essentially two programming models.
  - MQI
  - JMS (for non-Java languages use XMS)
- These programming models are available in a number of languages
  - C
  - C++
  - C#
  - Java
  - COBOL
  - Etc..

Which client to use

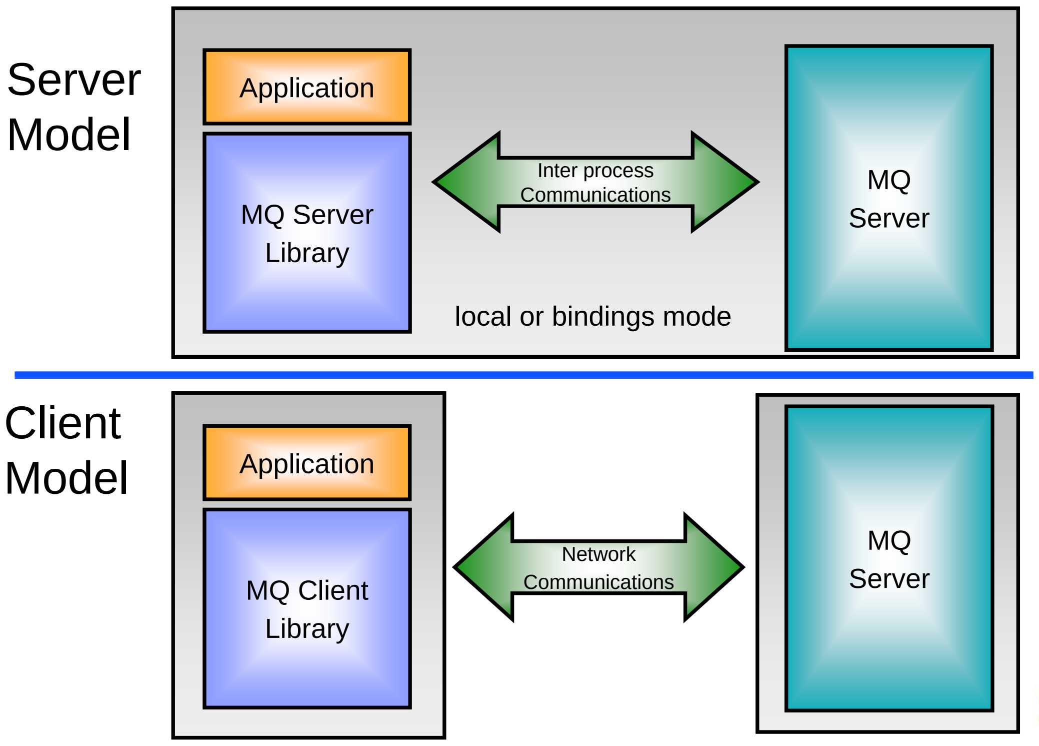
- Power of MQI vs Portability of JMS
  - JMS does not tie you to a provider (99% portable)
  - JMS available for non-JAVA languages in XMS
    - XMS is IBM specific though
- Multiple backend servers required ?
  - Choose JMS/XMS to talk to both WMQ and WPM
- Communications Protocol
  - SNA, SPX and NetBIOS only support by MQ C Client
- How important is speed ?
  - C tends to be faster than Java
  - MQI tends to be faster than JMS/XMS

Which client to use

N O T E S

- For the majority of cases the same application can be written using any of the clients. There are a few exceptions to this where there is a particular function available in only one client
  - For example MQ supports the notion of message grouping whereas JMS doesn't.
- The decision as to which client to use often comes down to which one fits in best with the current application environment. For example, if an enterprise codes all of its applications in Java then clearly choosing one of the Java clients would be sensible rather than using JNI to call the C client.
- The other major decision is what programming model should be used. This comes down to choosing between the MQI and JMS. The MQI is particular to WebSphere MQ and while it is extremely common and powerful it is not provided by any other messaging provider. As a consequence porting an application written to the MQI to another provider would require considerable effort. JMS is the standard way of doing messaging in a Java environment and, as such, applications written to JMS should port easily to another provider. Note, however, than a JMS application on one provider can not necessarily communicate with a JMS application on another provider.

What is an MQ Client ?

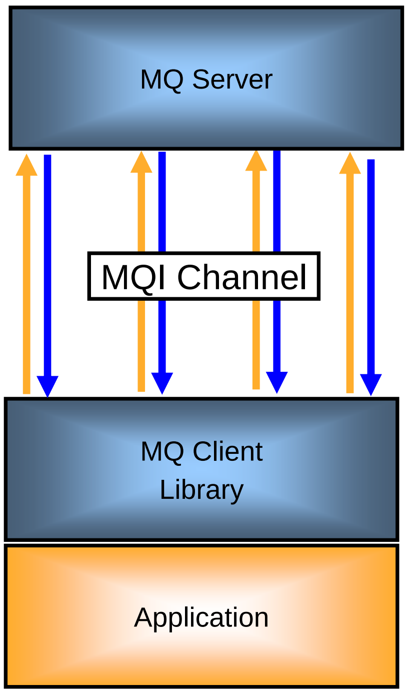


What is an MQ Client ?

N O T E S

- The WebSphere MQ Client support is part of the WebSphere MQ product that can be installed and used separately from the MQ server. It provides a set of libraries which can be linked with your applications to provide access to WebSphere MQ queues without requiring the application to run on the same machine as the queues.
- Generally speaking an application is linked either with the client libraries or with the server libraries (often called ‘local’ or ‘bindings’ mode). In bindings mode the application communicates with the Queue Manager via an inter-process communications link of some kind. In client mode the application communicates via a network connection. However, as can be seen from the diagram, the two models are logically equivalent. For this reason the functionality provided at the client is almost identical to that provided by local applications.
- For further explanation please see Chapter 1. Overview of WebSphere MQ clients in the WebSphere MQ Clients manual. This presentation contains references to further chapters in the Clients manual.

How does a client work ?



- Requires network access
- Each MQI Call shipped to server
- Response returned to application

MQI Calls

MQCONN	MQCONNX	MQDISC
MQOPEN	MQCLOSE	MQSUB
MQPUT	MQPUT1	MQGET
MQCB	MQCTL	
MQINQ	MQSET	
MQCMIT	MQBACK	

How does a client work ?

N O T E S

- An application that you want to run in a WebSphere MQ client enviroment must first be linked with the relevant client library.
- All the standard MQI functions, except MQBEGIN, are available to clients. The key MQI call at this point is clearly MQCONN(X). It is this call which determines either directly or indirectly which Queue Manager the application will try to connect to. We'll cover this in more detail later – let's assume that we manage to connect to a Queue Manager somewhere.
- As the application issues each MQI call, MQ client code directs the request to the queue manager over the communication link. The MQI request is essentially serialised, sent over the communications link. The server receives the request and issues the request on behalf of the client application. It then send back a reply to the client.
- The surrogate application issuing these requests on behalf of the client is a running channel of type SVRCONN. Each remotely connected client will have a SVRCONN channel running on its behalf. It is possible to have many thousands of these channels running into a single Queue Manager.

How to install a client

1. Install a MQ client and MQ server system
   - Install MQ server using the SERVER CD ROM
   - Install the MQ client using the CLIENT CD ROM
2. Install MQ client and server on the same machine
   - Install MQ server from SERVER CD ROM
   - and select MQ clients you wish to install
3. Install MQ client from SupportPacs site
   - Download SupportPac
   - Extract and run installation program

See *WebSphere MQ Clients* manual
for platform specific details

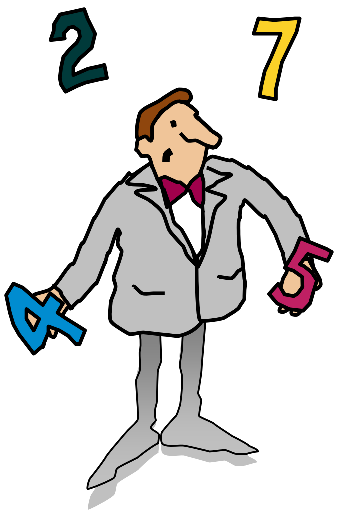
How to install a client

NOTES

- See Chapter 3. Installing client components from WebSphere MQ products and Version 5 MQSeries products (not z/OS)
- WebSphere MQ Version 5 products (not z/OS) include easy installation feature that helps install MQSeries clients quickly.
- If you're using WebSphere MQ for z/OS or another WebSphere MQ product, see Chapter 4. Installing WebSphere MQ clients with other MQSeries products.
- WebSphere MQ SupportPacs can be downloaded from
  - General Index
    - https://www-304.ibm.com/support/docview.wss?uid=swg27007197
  - MQC7 – MQ V7 Clients
    - https://www-304.ibm.com/support/docview.wss?uid=swg24019253
  - MQC6 – MQ V6 Clients
    - https://www-304.ibm.com/support/docview.wss?uid=swg24009961
  - MQC5 – MQ Client for VSE
    - https://www-304.ibm.com/support/docview.wss?uid=swg24010051
  - MQC4 – MQ Client for OpenVMS
    - https://www-304.ibm.com/support/docview.wss?uid=swg24009031

What about Licensing ?

- Installable clients can be downloaded for free
  - Available on many platforms
- Client attachment feature required for z/OS
- Extended Transactional (XA) Clients are not free



What about Licensing ?

N O T E S

- The Client Attachment Feature for z/OS is chargeable.
  - In MQ V7 5 Administration client connections, for example for use by MQ Explorer, are allowed for free
- Extended Transactional (XA) Clients are also chargeable.

Building a client application

- Compile your application as you would for local application
- Make sure you link your application with CLIENT libraries
  - libmqic* for "C" applications on UNIX systems
  - mqic32.lib for "C" applications on Windows
  - imqb23* imqc23* for "C++" applications
- Take care when linking threaded programs
  - e.g. libmqic_r.a for AIX
- Ensure that the correct runtime libraries are available
  - e.g.mqic32.dll for Windows

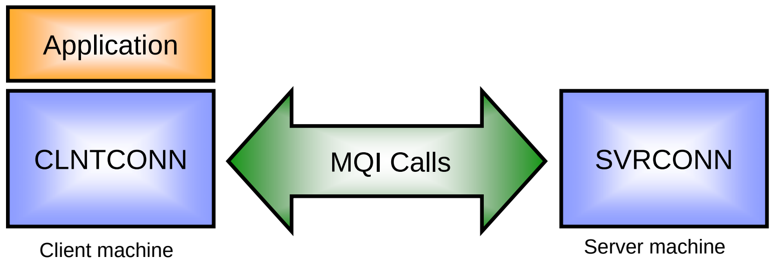
Building a client application

N O T E S

- MQ Client applications are essentially the same as normal, locally bound applications. The source and therefore the object deck is identical. The decision as to whether to run as a client is normally made at link time depending on whether the application is linked with the client or server libraries.
- Some applications delay this decision still further until run time. By dynamically loading the server or client library at run time the same application program can run either in client or server mode depending on the environment settings at run time. It is even possible for the same application to run both as a local application and a client at the same time !
  - (an example of this is SupportPac MO71).
- For further information see See Chapter 11. Building applications for WebSphere MQ clients

How to connect a client to a server

- The client must be able to identify which channel it should use to communicate with the queue manager
- How to specify the client's connection to a queue manager:
  - Explicitly on the MQCONNX verb
  - MQSERVER variable
  - Client channel tables
- Java client programs use either the MQEnvironment Java class or JNDI (using JMS)



How to connect a client to a server

N O T E S

- A channel is a logical communication link (see the WebSphere MQ Intercommunications manual). Clients communicate with a server using channel called a client connection (CLNTCONN). On the server there must be a server connection (SVRCONN) channel available to connect to.
- The client identifies which channel should be used when the application issues an MQCONN/MQCONNX call.
- The choice about which channel to use is made by checking the following (in this order):
  - The ClientConnOffset or ClientConnPtr in the MQCNO structure (if an MQCONNX was issued).
  - The MQSERVER environment variable.
  - The client channel definition table. This can be found as described by the MQCHLLIB and MQCHLTAB environment variables or using the Active Directory on Windows.
  - The channel definition table found in the default path.
- Java clients don't use the above method. The standard MQ java classes use the MQEnvironment class to identify the channel, while JMS clients use the Java Naming and Directory Interface (JNDI) to identify channels.

Configuring the client

- Environment variables can be used to configure the way the client works:

►**MQSERVER**	defines a minimal client channel
►**MQCCSID**	overrides the client machines CCSID
►**MQCHLLIB**	Path to the directory containing the client channel definition table can point to a shared drive
►**MQCHLTAB**	Name of the file containing the client channel definition table (default: amqclchl.tab)
►**MQNAME**	specifies the local NetBIOS name of the client
►**MQSSLKEYR**	specifies the location of an SSL key repository

Configuring the client

N O T E S

- See Chapter 9. Using the WebSphere MQ environment variables
- Not all the available environment variables are listed.
  See the above chapter for descriptions of variables used less often.

Using the MQSERVER variable

- The easiest way to define a client channel.
  - BUT has default CLNTCONN properties, eg 4Mb MAXMSGLEN
- Takes precedence over channel tables
  - but is superseded by the use of the MQCNO structure.
- set MQSERVER=ChannelName/TransportType/ConnectionName
  - In Windows: use Control Panel -> System -> Advanced ->Environment Variables
  - In UNIX: export MQSERVER
- Examples:
  - MQSERVER=SYSTEM.DEF.SVRCONN/TCP/127.0.0.1
  - MQSERVER=SYSTEM.DEF.SVRCONN/TCP/127.0.0.1**(1415)**
  - MQSERVER=SYSTEM.DEF.SVRCONN/TCP/JUPITER.SOLAR.SYSTEM.UNI
  - MQSERVER=SYSTEM.DEF.SVRCONN/LU62/BOX99

SHARE in Anaheim 2011

Using the MQSERVER variable

N O T E S

- See Chapter 9. Using WebSphere MQ environment variables
- Using the MQSERVER has the advantage that a client channel definition does not have to be created on a server and then the client channel table distributed as required.
- However, MQSERVER cannot be used if more advanced options are required on the channel (such as SSL) and the variable has to be set on each client machine.
- A SERVER side channel still needs to be defined (a SVRCONN channel).
- Channel name is case sensitive and it names a SVRCONN type channel.
- Certain channel options are assumed e.g. MAXMSGL is preset to 4MB.
- Use upper case for the transport type (TCP, LU62, NETBIOS, SPX).
  - If you don’t you’ll get a 2058 reason code on the connect
- ConnectionName is IP address, host name or partner LU name (or destination)

SHARE in Anaheim 2011

Channel definition tables

- A channel definition table is:
  - A binary file (not editable by a user)
  - Created by RUNMQSC (or other MQ mechanism) as AMQCLCHL.TAB (by default) when client channels are defined
    - Use CSQUTIL MAKECLNT function on z/OS
  - Located in directory (by default):
    - <mq root>\qmgrs\QMGRNAME\@ipcc (Windows)
    - <mq root/qmgrs/QMGRNAME/@ipcc (UNIX)
  - Read by the client if no MQSERVER variable defined and MQCONNX options are not used

SHARE in Anaheim 2011

Channel definition tables

NOTES

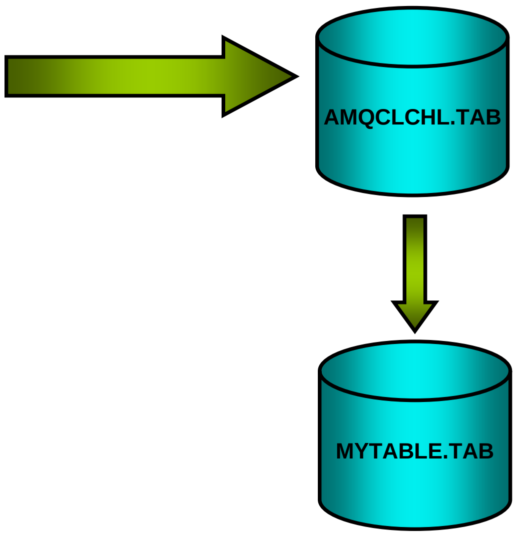
- See Chapter 8. Using Channels
- Never remove the channel definition table from its default location; always copy it.
- You cannot append channel definition tables together. If you want to define multiple client connection channels then you need to define all the channels on one of the servers.
- Channel definitions can be shared by more than one client. In other words, the client definition table can be located on a file server.
- To make a client channel definition table on z/OS you use the CSQUTIL MAKECLNT function. For details see z/OS System Administration Guide.

SHARE in Anaheim 2011

How do I create and deploy a channel table ?

RUNMQSC

def chl(...) chltype(clntconn) ….



<mq root>\qmgrs\QMGRNAME\@ipcc (Win)
<mq root>/qmgrs/QMGRNAME/@ipcc (Unix)

copy

c:\mqm\qmgrs\qmgrname\@ipcc\AMQCLCHL.TAB

to

z:\mytable.tbl

How do I create and deploy a channel table ?

NOTES

- Choose **one** of your MQ server machines to define all your CLNTCONN definitions. Find the AMQCLCHL.TAB file and copy it to a location which is accessible by the client machines. The name of the file can be changed if required but you must use the MQCHLTAB environment variable to MQ what you called it.

- By default, the client looks for the AMQCLCHL.TAB file in

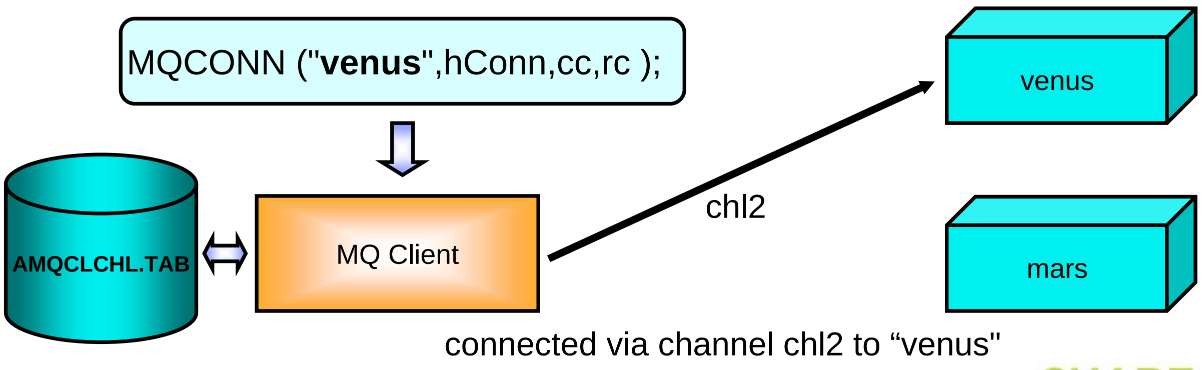
Unix	: /var/mqm
Windows	: \<mq data root>

- Environment variables, MQCHLLIB and MQCHLTAB, can be used to enable the clients to locate the channel table

- See Chapter 12. Running applications on WebSphere MQ Clients.

Using Channel Definition Tables: Example 1

- How is the QMNAME client channel attribute used?
  - def chl(chl1) chltype(clntconn) trptype(tcp) conname(host1) qmname(mars)
  - def chl(chl2) chltype(clntconn) trptype(tcp) conname(host2) qmname(venus)


MQCONN ("**venus**",hConn,cc,rc);
AMQCLCHL.TAB
MQ Client
chl2
venus
mars
connected via channel chl2 to “venus"

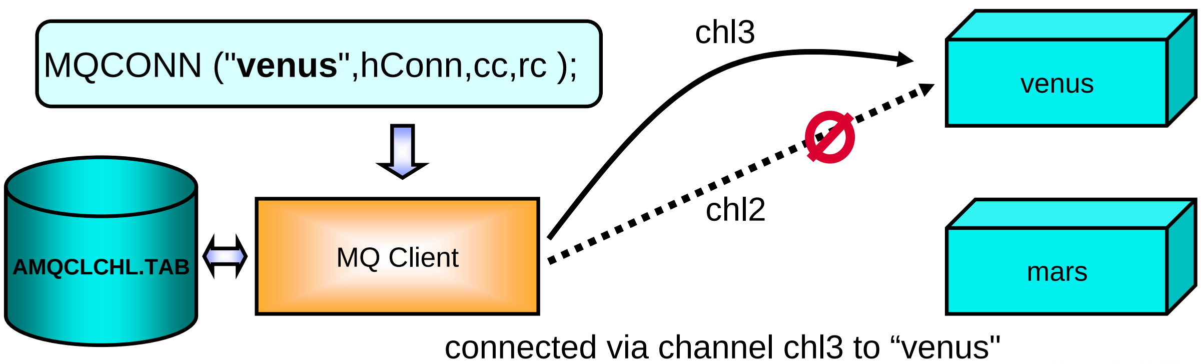
Using Channel Definition Tables: Example 1

N O T E S

- In this example the user has defined two client channels.
- The client searches through the client channels in alphabetical *channel name order.* It looks for a channel definition with a QMNAME field which matches what the application specified on the MQCONN call. We therefore find channel ‘chl2’. If we did not find any channel definitions which match the application would receive a 2058 (Queue Manager name error) reason code.
- The transmission protocol and associated connection are extracted from the channel definition and an attempt is made to start the channel to the machine identified (venus). In order for the connection to be successful clearly there must be started listener at the remote machine and the queue manager itself must be started.
- If the connection can not be established then a 2059 (Queue Manager not available) reason code is returned to the application. If you believe the Queue Manager is running then look in the client error log for an error message explaining the reason for the faliure.
- The error log is in <mq install path>\errors\AMQERR01.LOG

Using Channel Definition Tables: Example 2

- Multiple routes to the same Queue Manager
  - def chl(chl1) ….trptype(tcp) conname(host1)   qmname(mars)
  - def chl(chl2) ….trptype(tcp) conname(tokenring) qmname(venus)
  - def chl(chl3) ….trptype(tcp) conname(ethernet) qmname(venus)
  - def chl(chl4) ….trptype(tcp) conname(dialup)   qmname(venus)



connected via channel chl3 to “venus"

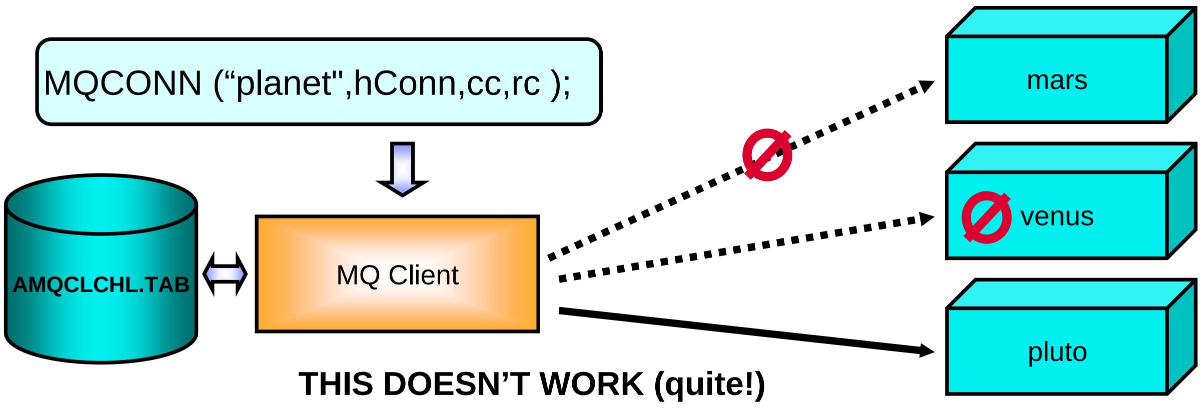
Using Channel Definition Tables: Example 2

N O T E S

- In this example there are three channels, that all connect to the same queue manager using different connections (ethernet, tokenring and dialup). This provides a level of redundancy.
- The client has to pick one, but which one?
- The client attempts to start channel 'chl2' (since the search is in alphabetical channel name order); its QMNAME attribute matches the name in the MQCONN. However the communication link is currently broken.
- Channel 'chl3' is now started instead because QMNAME still matches what was specified on the MQCONN call.
- So the client is connected to queue manager “venus" but via ethernet.

Using Channel Definition Tables: Example 3

- How do we have back-up Queue Managers ?
  - def chl(chl1) ….trptype(tcp) conname(ip.mars)   qmname(planet)
  - def chl(chl2) ….trptype(tcp) conname(ip.venus) qmname(planet)
  - …..
  - def chl(chl5) ….trptype(tcp) conname(ip.pluto)   qmname(planet)



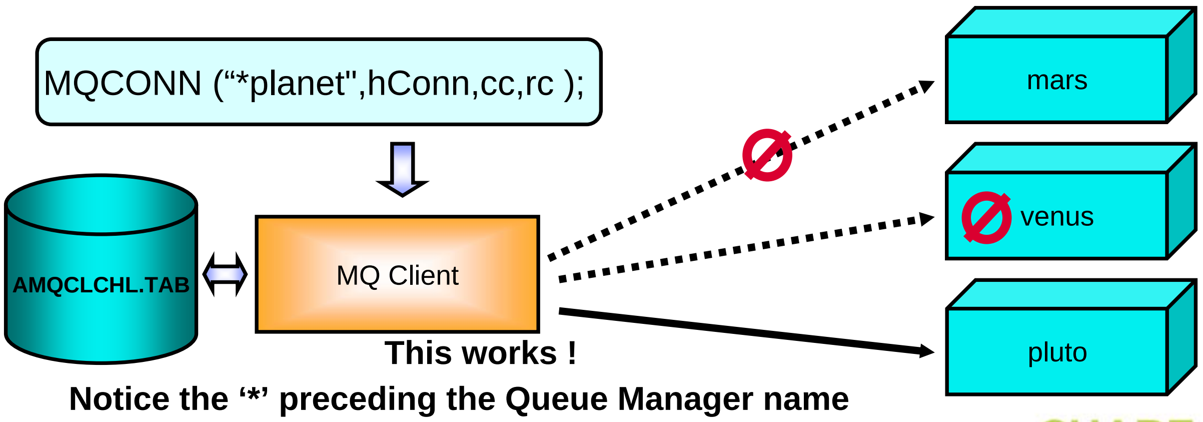
Using Channel Definition Tables: Example 3

N O T E S

- In this example the client tries to connect to a queue manager first using "chl1" but the communication link is down.
- Secondly it tries "chl2" but the queue manager is not currently running.
- Finally the client tries to connect using channel "chl5". The communications link is running and the queue manager is running.
- **However**, the name of the queue manager "pluto" does not match the one specified on the MQCONN call “planet” and so this connection fails.
- There are no remaining client channel definitions and so the MQCONN call fails with reason code MQRC_Q_MGR_NOT_AVAILABLE.
- What we need is a way to tell MQ that we, the application, don’t really care what the actual Queue Manager name is.

Using Channel Definition Tables: Example 4

- How do we have back-up Queue Managers ?
  - def chl(chl1) ….trptype(tcp) conname(ip.mars)   qmname(planet)
  - def chl(chl2) ….trptype(tcp) conname(ip.venus) qmname(planet)
  - …..
  - def chl(chl5) ….trptype(tcp) conname(ip.pluto)   qmname(planet)



This works !

Notice the ‘*’ preceding the Queue Manager name

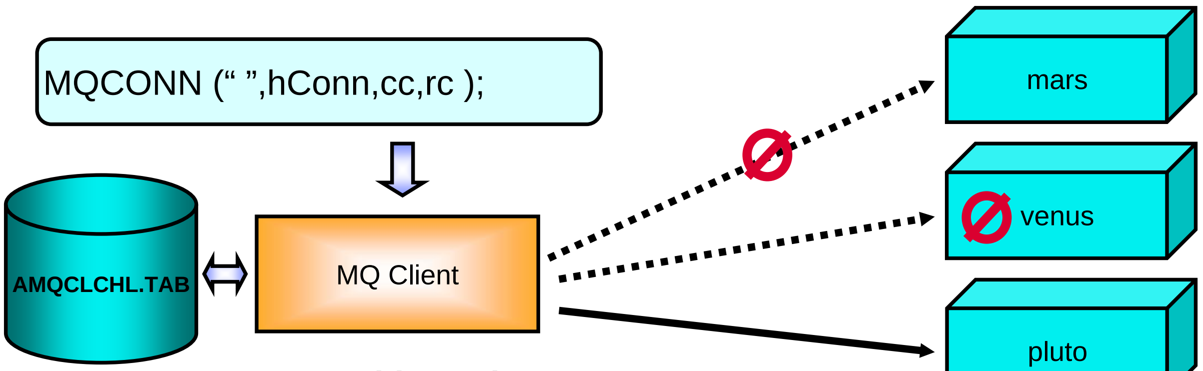
Using Channel Definition Tables: Example 4

N O T E S

- This example is only different to example 3 in that the user has specified "*planet“ rather than just “planet”.
- The * specifies that the client does not care if the actual name of the Queue Manager does not match the name given.

Using Channel Definition Tables: Example 5

- How do we have back-up Queue Managers ?
  - def chl(chl1) ….trptype(tcp) conname(ip.mars) qmname()
  - def chl(chl2) ….trptype(tcp) conname(ip.venus) qmname()
  - …..
  - def chl(chl5) ….trptype(tcp) conname(ip.pluto) qmname()



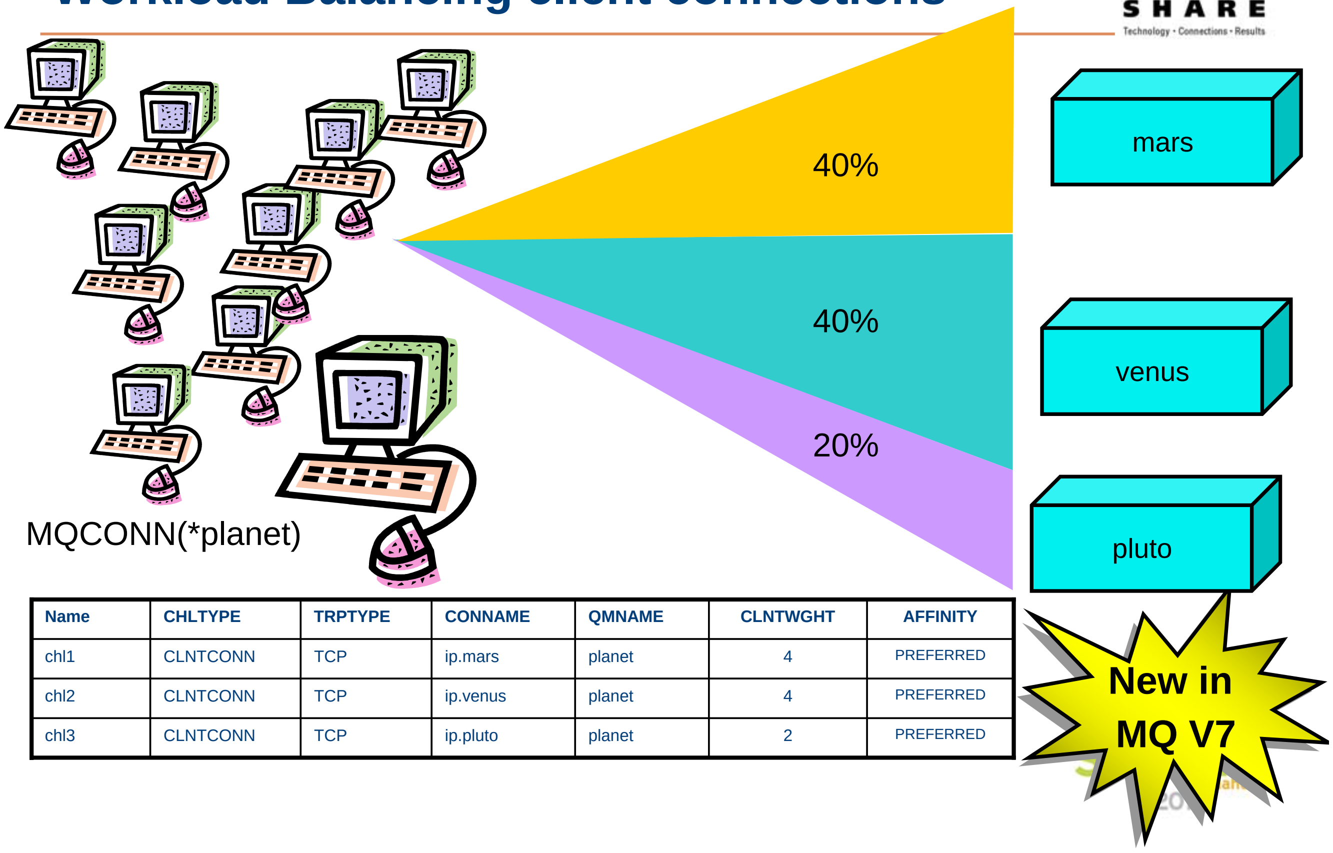
This works too !

Using Channel Definition Tables: Example 5

N O T E S

- This example shows it also possible for a client to specify a blank Queue Manager name, in fact this is a common scenario.
- In a local application this means 'connect to the default Queue Manager'. In a client application is means 'connect to any of the 'default' Queue Managers'. In other words, any CLNTCONN channel with a blank Queue Manager field.
- Now, since the application has not specified the name of the Queue Manager there is no problem with whatever the target Queue Manager happens to be. In other words, "<blank>" is equivalent to "*".

Workload Balancing client connections



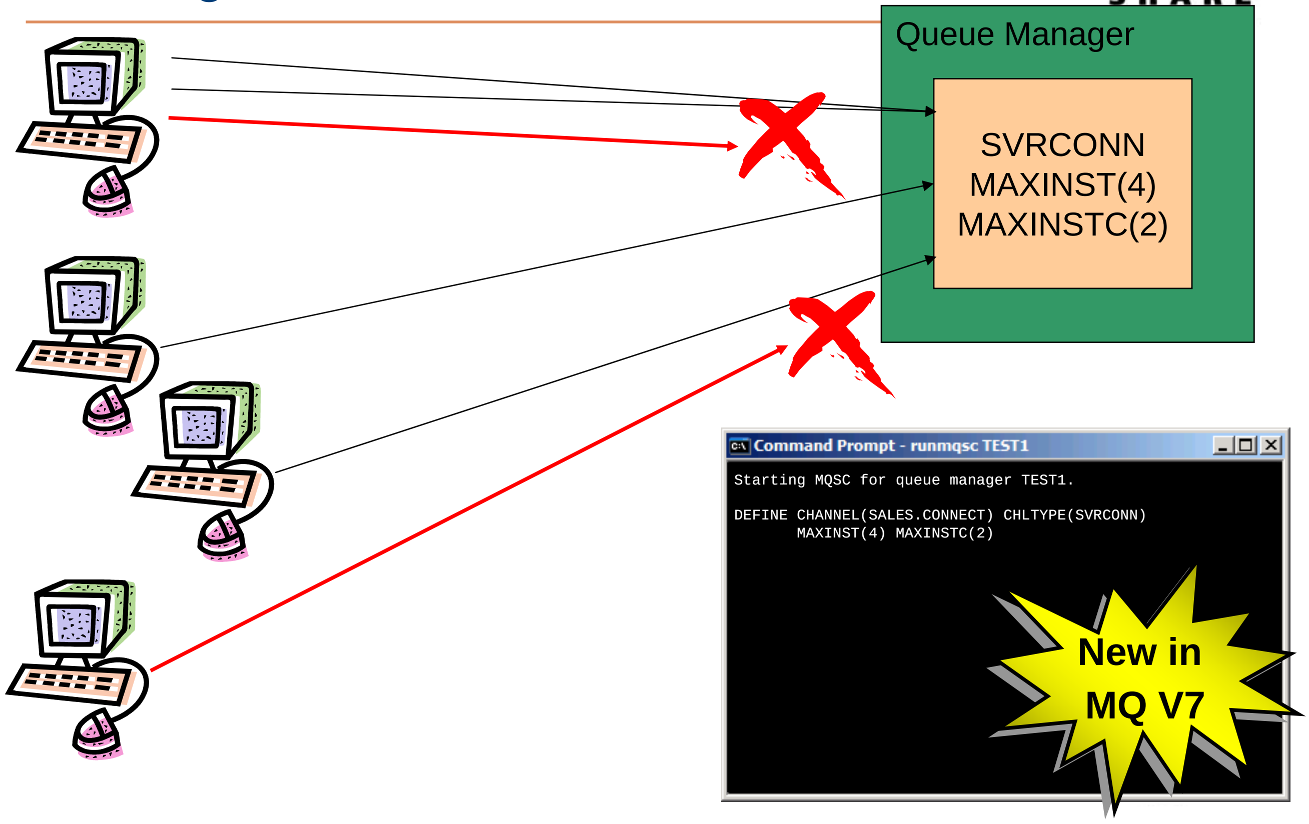
Name	CHLTYPE	TRPTYPE	CONNAME	QMNAME	CLNTWGHT	AFFINITY
chl1	CLNTCONN	TCP	ip.mars	planet	4	PREFERRED
chl2	CLNTCONN	TCP	ip.venus	planet	4	PREFERRED
chl3	CLNTCONN	TCP	ip.pluto	planet	2	PREFERRED

Workload Balancing client connections

N O T E S

- When using a client channel definition table (CCDT) to configure the client connectivity used by your client applications, you can provide a number of destination queue managers to choose from in order to provide redundancy and alternate destinations when one fails.
- You can define these destinations with a weighting so that the spread of client connections between the group of queue managers is as you require.
- You can then use the same CCDT with all your clients – no need to produce different copies of the CCDT to spread out your client connections across all the back-end servers.
- The default value of CLNTWGHT is 0 – which retains the V6 behaviour of primary then secondary choices chosen by alphabetical order.
- By default client channels have AFFINITY(PREFERED) set. This means that any particular client application will attempt to connect to the same queue manager each time. This is the same behaviour that was available in V6 with the mechanism that the primary connection was attempted first, then if it was not available, the secondary connection was attempted, and so on. If it is desired that connections from the same machine are to be workload balanced as above, AFFINITY(NONE) can be chosen.

Limiting client connections



```
Starting MQSC for queue manager TEST1.

DEFINE CHANNEL(SALES.CONNECT) CHLTYPE(SVRCONN)
       MAXINST(4) MAXINSTC(2)
```

Limiting client connections

N O T E S

- New attributes on your server-connection channels allow you to restrict the number of client-connection instances that can connect in. Now you can configure your system so that server-connection instances cannot fill up your maximum number of channels.
- There are in fact two attributes on your server-connection definition.
- MAXINST restricts the number of instances in total for the specific channel name.
- MAXINSTC restricts the number of instances from a specific IP address for that channel name.

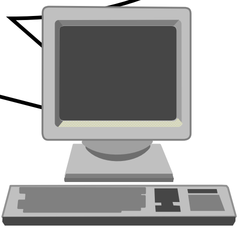
Using MQCONNX

MQCONNX (qmgr name, CNO, Hconn, cc, rc)

```
MQCNO - Connection Options:
MQCHAR4  StrucId;                /* Structure identifier                       */
MQLONG   Version;                /* Structure version number                   */
MQLONG   Options;                /* Options that control the action of MQCONNX    */
. . .
MQLONG   ClientConnOffset;       /* Offset of MQCD structure for client connection */
MQPTR    ClientConnPtr;          /* Address of MQCD structure for client connection*/
. . .
```

```
MQCD - Channel Definition

. . .
MQCHAR   ChannelName[20];        /* Channel definition name                    */
. . .
MQCHAR   ConnectionName[264];    /* Connection name                            */
```

If used, overrides MQSERVER and CHANNEL tables



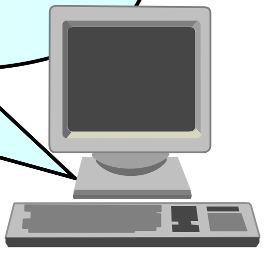
Using MQCONNX

N O T E S

- See Chapter 8. Using channels and Chapter 10. Using the message queue interface (MQI)
- MQCONNX calls provide an alternative way to identify which channel a client should use. This method overrides the use of both the MQSERVER environment variable and the use of channel definition tables.
- The MQCNO structure allows you to pass an MQCD (channel definition) structure to use directly to the client library. This means the channel can be provided programmatically at run time.
- The MQCD definition can either be provided via a pointer or via an offset. The offset field is for those languages which often don't have pointers such as COBOL.
- You can provide SSL related information in the MQSCO structure of the MQCONNX call.
- See sample **amqscnxc**.

Using MQCONNX

```
MQCD cd = {MQCD_CLIENT_CONN_DEFAULT};

cno.Version = MQCNO_VERSION_2;    // CD ignored if CNO not V2 or greater

cno.ClientConnPtr = &cd;

strcpy(cd.ChannelName,"SYSTEM.DEF.SVRCONN“);

strcpy(cd.ConnectionName,"VENUS.SOLAR.SYSTEM.UNI“);

MQCONNX ( “”, &cno, &hQm, &cc, &rc)
```



Using MQCONNX

N O T E S

- The ClientConnOffset or ClientConnPtr can be used to specify the location of the channel definition structure. In order for the location to be picked up by the client the version of the MQCNO structure must be 2 or greater.
- The details about the channel can now be placed in the MQCD structure.
- Note: MQCNO_STANDARD_BINDING and MQCNO_FASTPATH_BINDING are ignored when calling MQCONNX from a client. Whether the channel actually runs using standard or fastpath is controlled via the MQIBINDTYPE setting in the server configuration.

Debugging Connection problems

- Check the error logs!
  - Server error log <root>\qmgrs\<QM>\errors\AMQERR01.LOG
  - Client error log <root>\errors\AMQERR01.LOG
- Double check the MQSERVER variable
- Does the amqsputc sample work?
- Is the network working ?
  - Can you "tcp ping" the host?
- Is there an MQ listener running?
- Is the channel table specified correctly
  - Do the environment variables point to the right place?

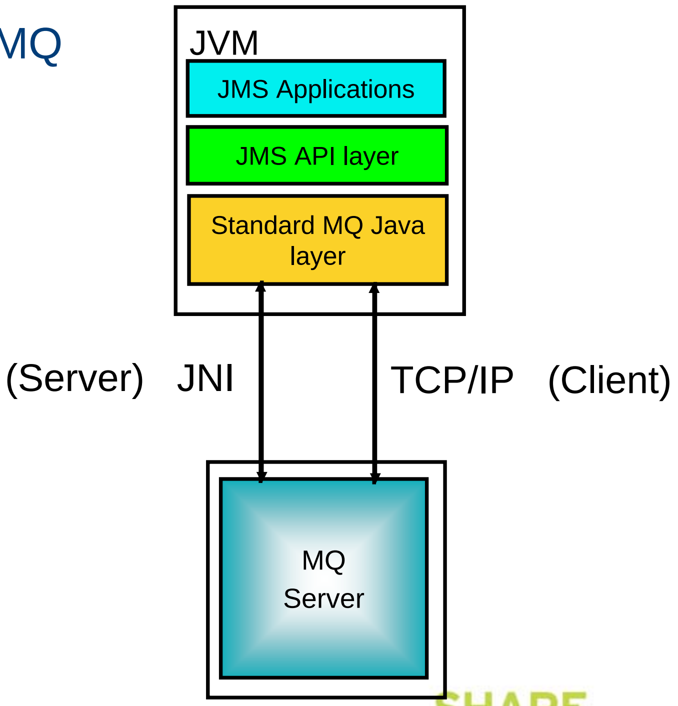
Debugging Connection problems

N O T E S

- These are some of the simple ways to try and diagnose why you can't connect to a queue manager.
- Don't forget about the error logs. Both the client and the server machine have error logs which will tell you why an MQCONN is failing.
- Try your configuration with a tried and trusted application such as the sample AMQSPUTC.
- Check that the server you are trying to connect to is available, the network connection is available, that the queue manager is running and that a listener for that queue manager has been started.
- Check that you have correctly identified the whereabouts of your channel definition table.

MQ Java Client

- Java classes for accessing MQ
- May be optional Install component (e.g. Windows)
- JMS interface also provided



SHARE in Anaheim 2011

MQ Java Client

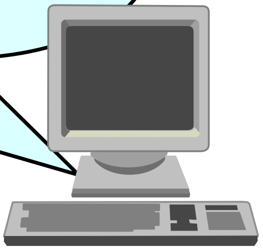
N O T E S

- The MQ Java client can be used to access a server directly using the Java Native Interface (JNI) or as a client using the TCP/IP protocol.
- The MQ Java interface maps fairly closely to the MQI in many ways, however a JMS interface which complies with Sun standards is also provided.
- The MQ API is more complex but offers more control.
- JMS is a simpler, higher-level API, although it does offer some facilities not available in the MQ API. For example: the publish/subcribe model, and message selectors.

SHARE in Anaheim 2011

Connecting Clients in Java

```
import com.ibm.mq.*;            // Include the MQ package

MQEnvironment.properties.put( MQC.TRANSPORT_PROPERTY,
                              MQC.TRANSPORT_MQSERIES);

MQEnvironment.hostname = "VENUS.SOLAR.SYSTEM.UNI";
MQEnvironment.channel  = "SYSTEM.DEF.SVRCONN";

try
{
   MQQueueManager qmgr = new MQQueueManager("");
}
catch (MQException ex) { ex.printStackTrace(System.err);}
```



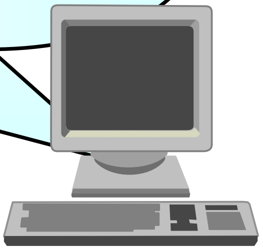
Connecting Clients in Java

N O T E S

- This is a simple example showing how a Java client identifies which queue manager it wishes to connect to.
- The presence of a non blank hostname informs the client that the bindings mode (direct server connection) cannot be used.
- The other MQEnvironment variables (such as channel) can be used to configure the client connection to the queue manager.

Connecting Clients in Java

```
import com.ibm.mq.*;            // Include the MQ package

MQEnvironment.properties.put( MQC.TRANSPORT_PROPERTY,
                              MQC.TRANSPORT_MQSERIES);

URL chanTab = new URL(ftp://ftp.server/mq/AMQCHLCL.TAB);
try
{
   MQQueueManager qmgr = new MQQueueManager(“venus“,chanTab);
}
catch (MQException ex) { ex.printStackTrace(System.err);}
```



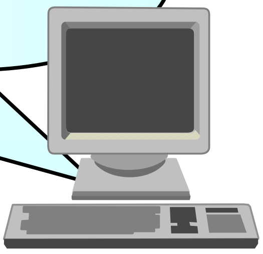
Connecting Clients in Java

N O T E S

- This is a simple example showing how a Java client identifies which queue manager it wishes to connect to.
- The presence of a non blank hostname informs the client that the bindings mode (direct server connection) cannot be used.
- The other MQEnvironment variables (such as channel) can be used to configure the client connection to the queue manager.

Connecting Clients in JMS

```
import javax.jms.*;
import javax.naming.*;
import javax.naming.directory.*;
 .
java.util.Hashtable environment = new java.util.Hashtable();
environment.put(Context.INITIAL_CONTEXT_FACTORY, icf);
environment.put(Context.PROVIDER_URL, url);
Context ctx = new InitialDirContext( environment );

QueueConnectionFactory factory;
factory = (QueueConnectionFactory)ctx.lookup("cn=ivtQCF");
```



Connecting Clients in JMS

N
O
T
E
S

- This is a short and incomplete example showing the start of a JMS application which connects to a queue manager.
- The application provides the location of a context where objects are placed which can be used by JMS to start a connection to a queue manager.
- The QueueConnectionFactory object can contain a channel name and other details which identify how the application is to connect to a queue manager.

Programming Considerations

- Take care when specifying the queue manager name on MQCONN if using client channel definition table...
- Most MQI calls are SYNCHRONOUS and tend to be slower than in a server environment.
- Always be prepared for MQRC_CONNECTION_BROKEN.
- Always code MQGMO_FAIL_IF_QUIESCING.
- For optimum performance don't use really short lived connections (MQCONNs)
- Carefully code MQWI_UNLIMITED on MQGET calls.
- Use Asynchronous MQPUT and Read Ahead if appropriate

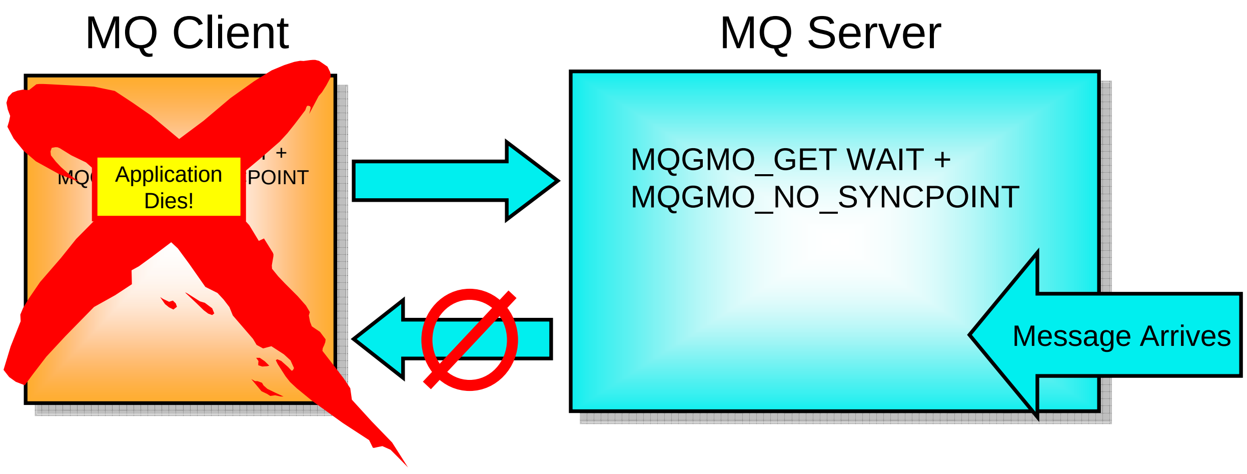
As always If you don't want to lose messages, code MQ*_SYNCPOINT on MQGET and MQPUT calls then issue MQCMIT

Programming Considerations

N O T E S

- In general the rules for programming clients is the same as for local applications. However, effects tend to be exaggerated – calls are slower, more can go wrong and windows tend to be larger..
- Because there's a real network and the Queue Manager is usually on a different machine than the client a client is much more likely to receive an MQRC_CONNECTION_BROKEN reason code from an MQI call. Be prepared for this and deal with it appropriately.
- The most expensive call is the MQCONN itself. For optimum performance it is imperative you don't connect too often and do a reasonable amount of work under each connection.
- As in local applications make sure all the processing of messages you care about are done under a transaction. This means MQPUT and MQGET calls should use the SYNCPOINT option. Failure to do so could lead to messages becoming lost. This behaviour tends to be more obvious in the client environment because the failure windows are much larger.
- If large numbers of non-persistent messages are involved it is worth considering using Asynchronous MQPUT and/or Read Ahead to avoid a line turnaround from the client per message.

Transactions



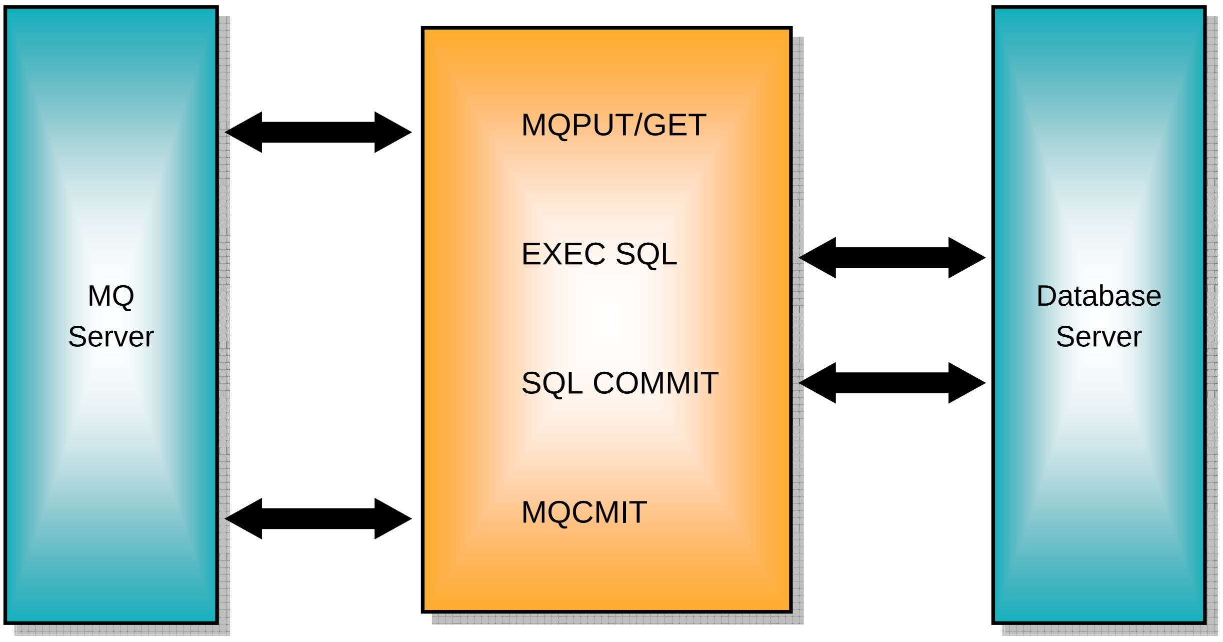
Message Is Lost!

Transactions

N O T E S

- This slide demonstrates the last point on the Programming Considerations slide.
- Message data can be lost if applications end while performing an MQGET with wait without being in syncpoint:
  - An application issues an MQGET with MQWI_UNLIMITED, but not in syncpoint.
  - There are currently no messages available on the server so the server starts waiting for one to arrive.
  - Meanwhile, the application ends unexpectedly before the MQGET returns.
  - A message now arrives, so the server gets it from the queue to send it to the application.
  - The server finds the application dead and so the message is discarded.
- **By getting messages in syncpoint if they are not correctly delivered they will not be discarded.**
- It is recommended that an application always explicitly states MQGMO_SYNCPOINT or MQGMO_NO_SYNCPOINT because the syncpoint default varies between servers. (z/OS the default is syncpoint, Distributed the default is no syncpoint).
- **The key point here, though, is that the syncpoint model for local and client machines is identical.**

Global Transactions



Multiple Resource Managers involved in the transaction

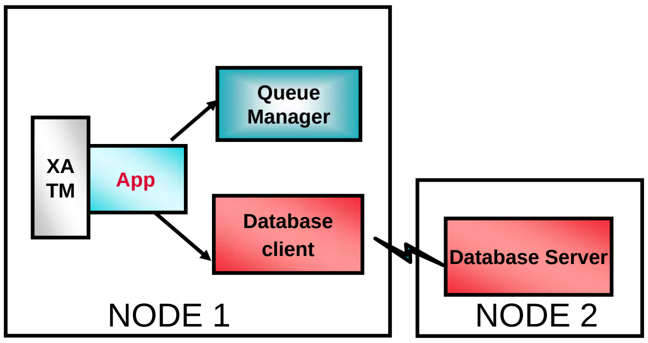
Global Transactions

N O T E S

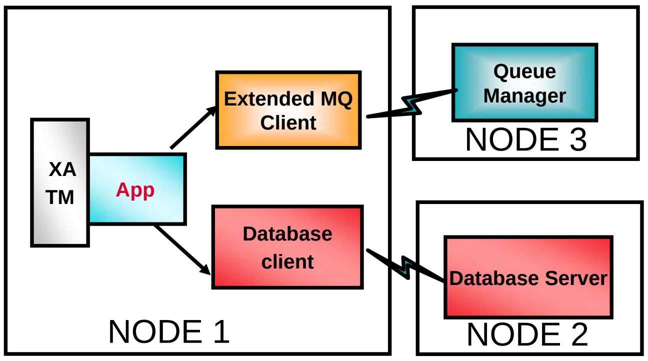
- The MQ client available for free download can only commit a unit of work carried out on the queue manager it is connected to. The client cannot be used to manage work carried out on other resource managers. Therefore the MQBEGIN call is not available within normal MQ clients.
- Work between different resource managers can only be loosely coordinated, as shown by the slide, where different resource managers are accessed using their native interfaces and under different units of work.
- However, this is often sufficient. In the example shown the MQGET operation is not committed until a successful commit of the database update. This means that if the application crashes the request message is rolled back to the queue and can be reprocessed. If the application crashes after the database update but before the MQ transaction is committed then the transaction would merely be replayed.

Extended Transactional Client

Local Application



Extended Transactional Client

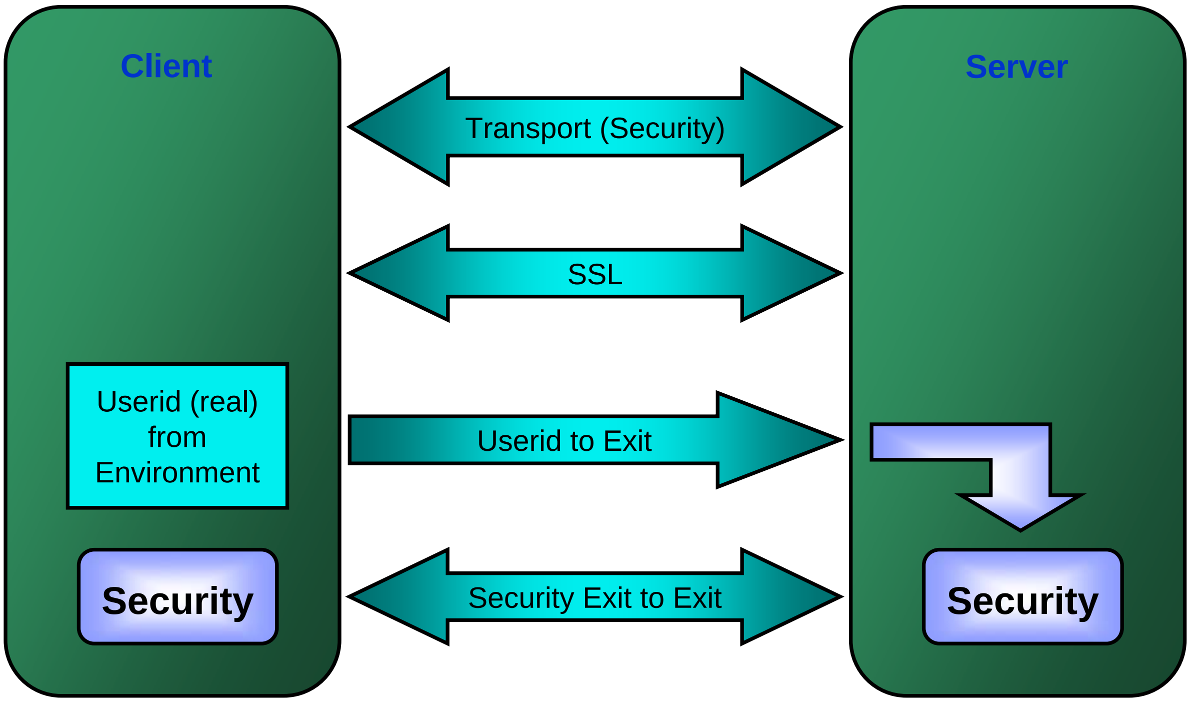


Extended Transactional Client

N O T E S

- The function provided by the Extended Transactional Client allows a client to participate in units of work coordinated by an XA transaction manager.
- Externally coordinated transactions can now work with queue managers located on different machines from where the transaction coordination takes place.
- The Extended Transactional Client still does not support the MQBEGIN call - all units of work must be started using the XA interface by an external transaction manager.
- Potentially allows simpler administration by separating the machines with the databases on from the machines with the queue managers on.
- Note: you will only find benefit in the Extended Transactional Client if you use another resource manager other than WebSphere MQ!

Client Security

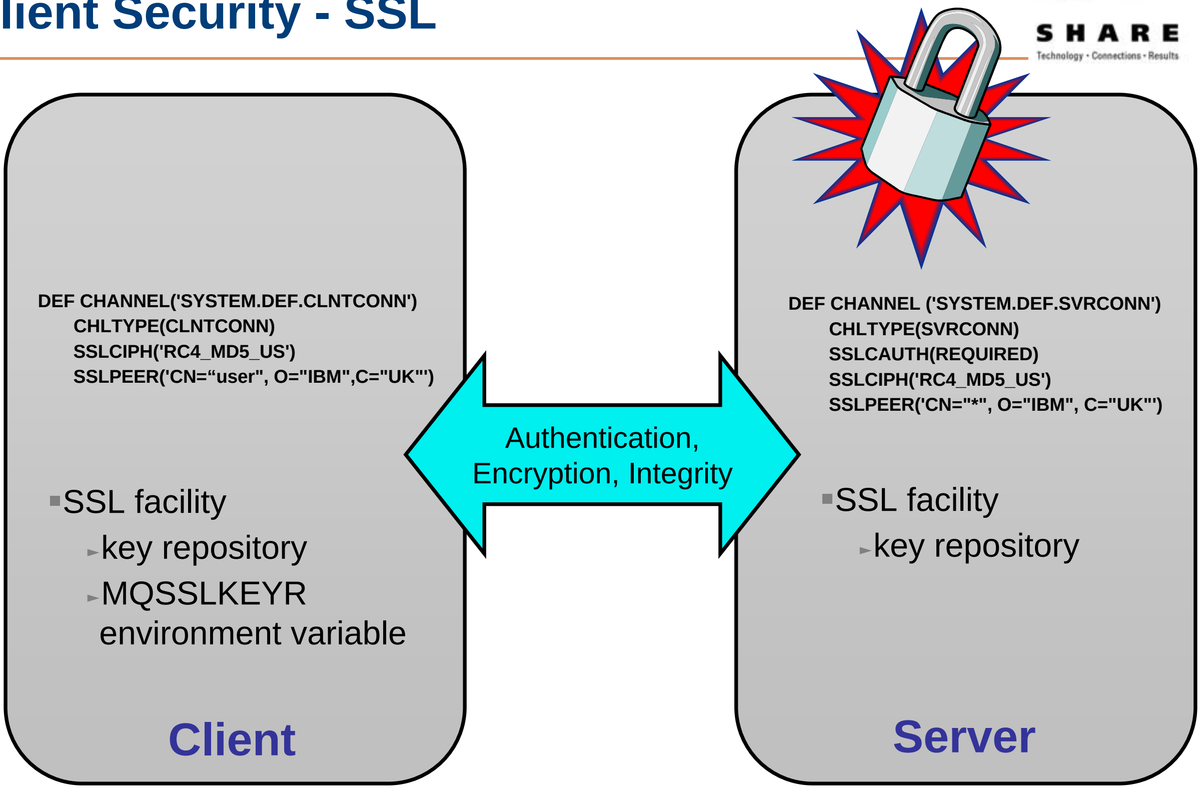


Client Security

N O T E S

- See Chapter 7. Setting up WebSphere MQ client security.
- Channel security exits
  - The channel security exits for client to server communication can work in the same way as for server to server communication. A protocol independent pair of exits provide mutual authentication of both the client and the server.
- See next slide for SSL
- If no client security exit, userid passed in MQCD
  - Windows NT/2K/XP and Unix -- pass the logged on UserID
  - Windows NT/2K/XP only
    Security ID (SID) passed in "Accounting" field in the message descriptor
- If the MQ server and client are both on Windows NT/2K/XP, and if the MQ server has access to the domain on which the client user ID is defined, MQ supports user IDs of up to 20 characters.
- On all other platforms and configurations, the maximum length for user IDs is 12 characters.

Client Security - SSL



Authentication, Encryption, Integrity
SSL facility
key repository
MQSSLKEYR environment variable
Client
SSL facility
key repository
Server

```
DEF CHANNEL('SYSTEM.DEF.CLNTCONN')
     CHLTYPE(CLNTCONN)
     SSLCIPH('RC4_MD5_US')
     SSLPEER('CN=“user", O="IBM",C="UK"')
```

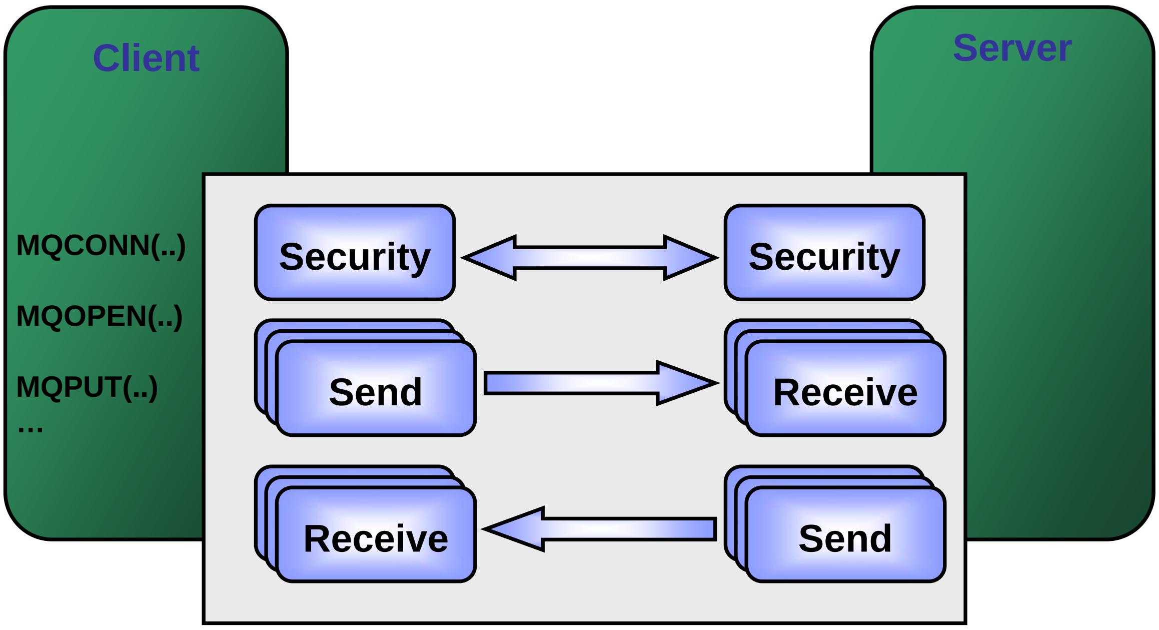
```
DEF CHANNEL ('SYSTEM.DEF.SVRCONN')
     CHLTYPE(SVRCONN)
     SSLCAUTH(REQUIRED)
     SSLCIPH('RC4_MD5_US')
     SSLPEER('CN="*", O="IBM", C="UK"')
```

Client Security - SSL

N O T E S

- See Chapter 9. The Secure Sockets Layer (SSL) on WebSphere MQ clients and the WebSphere MQ Security book.
- The Secure Sockets Layer (SSL) provides an industry standard protocol for transmitting data in a secure manner over an insecure network. The SSL protocol is widely deployed in both Internet and Intranet applications. SSL defines methods for authentication, data encryption, and message integrity for a reliable transport protocol, usually TCP/IP.
- SSL can be enabled on client channels by specifying a CipherSpec on the client and server connection channel definitions.
- SSL cannot be used if using the MQSERVER environment variable.
- If using the MQCNO structure to pass in the client channel on an MQCONNX call, a CipherSpec can be set in the MQCD structure.
- If using Active Directory on Windows you can use the setmqcsp control command to publish the client-connection channel definitions in Active Directory. One or more of these definitions can specify the name of a CipherSpec.

Exits



Message exits and Retry Exits are not applicable

Exits

N O T E S

The ExitPath stanza of the ini file determines location of exits, if not fully qualified on the DEF CHL command

Add to mqs.ini which is installed with both the server and the client.

Client : `ClientExitPath:`
`ExitsDefaultPath=`*`path`*

Server: `ExitPath:`
`ExitsDefaultPath=`*`path`*

Conversion is always done on the server to which the client is connected
Conversion exits, therefore, must be located on the server.

Summary

- Clients are a simple, low administration and cheap way of providing queuing throughout your network.
- Consider which client to use based on
  - Programming Language required (C,Java,C#, C++)
  - Programming model required (MQI vs JMS)
  - Performance
- Client applications can do the same as local applications
  - However, no network - no queuing

SHARE in Anaheim 2011

Further Information

N O T E S

- WebSphere MQ Information Center
  - Main index
    - http://publib.boulder.ibm.com/infocenter/wmqv7/v7r0/index.jsp
  - MQ Client information
    - http://publib.boulder.ibm.com/infocenter/wmqv7/v7r0/topic/com.ibm.mq.csqzaf.doc/cs10120_.htm
- WebSphere MQ home page
  - http://www.ibm.com/software/integration/wmq/
- WebSphere MQ SupportPacs:
  - General Index
    - https://www-304.ibm.com/support/docview.wss?uid=swg27007197
  - MQC7 – MQ V7 Clients
    - https://www-304.ibm.com/support/docview.wss?uid=swg24019253
  - MQC6 – MQ V6 Clients
    - https://www-304.ibm.com/support/docview.wss?uid=swg24009961
  - MQC5 – MQ Client for VSE
    - https://www-304.ibm.com/support/docview.wss?uid=swg24010051
  - MQC4 – MQ Client for OpenVMS
    - https://www-304.ibm.com/support/docview.wss?uid=swg24009031

SHARE in Anaheim 2011