Introduction to MQ Channels

Paul Clarke

MQGem Software

support@mqgem.com

Introduction

N O T E S

Paul Clarke spent many years working for IBM in MQ Development however in 2012 he set up his own small software company, MQGem Software, to write tools and utilities to run over IBM MQ.

In his IBM career he was involved with many areas of IBM MQ but mainly in the area of communications. Amongst other things he wrote the MQ Channels and the MQ C client.

Agenda

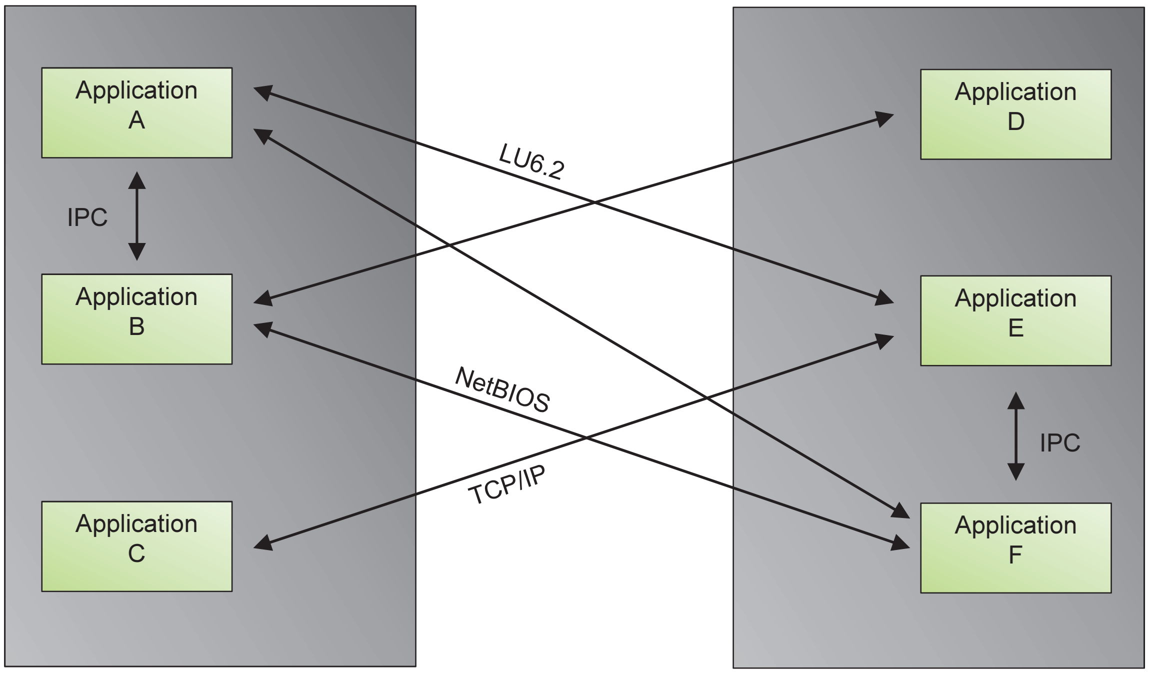
- **Channel Architecture**
  - Concepts
  - Channel Types
  - Channel Definition
- **Message Routing**
  - Direct Routing
  - Remote Queue Definitions
- **Channel Commands**
  - Starting and Stopping
  - Automatic Operation
  - Channel Status
- **Channel Operation**
  - Data Conversion
  - Channel Exits
  - Channel Protocol
  - Ideal Configuration

Agenda

N O T E S

Clearly MQ Channels is a broad subject, there are many aspects we could talk about and within the time limit of this presentation we probably couldn't do any of them real justice. However, this is an 'introduction' and after this presentation you should have a fair idea of what MQ Channels are, the terms that are used and how they operate. Enough to get you started and be able to continue to expanding your knowledge about the subject.

Before Messaging



Before Messaging

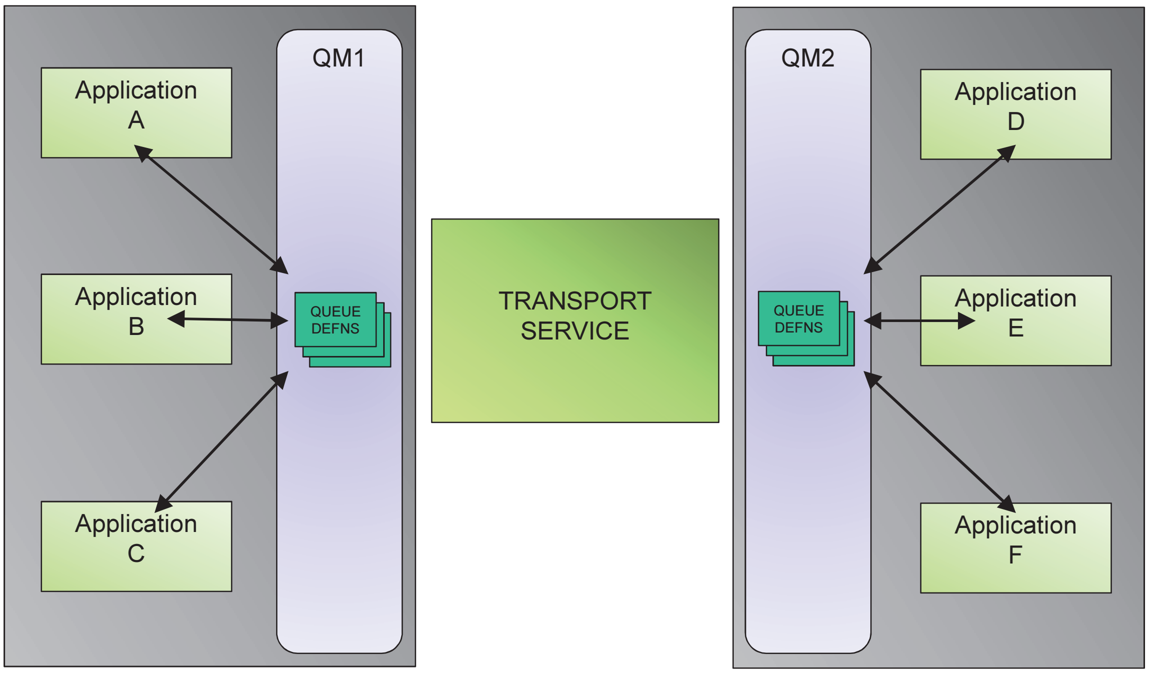
N O T E S

The days of stand-alone business applications are pretty much over. Nowadays applications are merely a piece of the solution which must be able to communicate both upstream and downstream with partner applications. These partner applications are often written by different development organisations and are located across the enterprise or even in other enterprises. The traditional approach of writing a TCP/IP application is just too complicated in modern business architectures. An application writer must concern himself with

- Network topology/machine addresses
- Data representation and formats
- Security
- Application availability and maintenance schedules
- Complicated network programming
  - Particularly if communicating with more than one partner.
- Assured delivery
  - Writing two-phase commit protocols and handling in-doubt situations

Most, if not all, of these concerns can be removed by a messaging backbone.

With IBM MQ



With IBM MQ

N O T E S

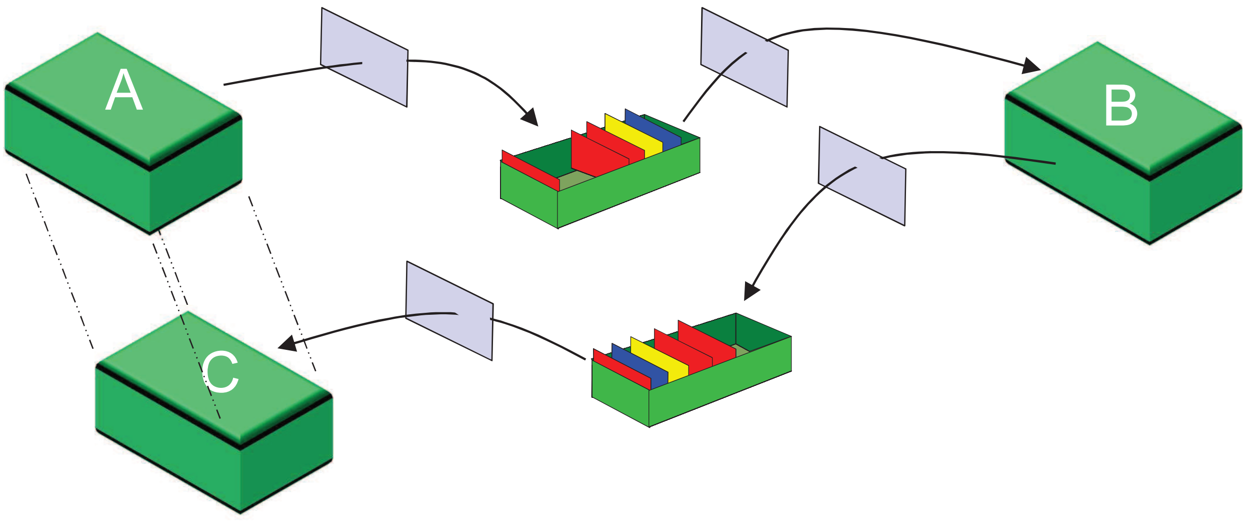
The introduction of IBM MQ vastly simplifies the work the application need do. An application now only needs to put a message to the correct queue and let IBM MQ worry about how to deliver the message across any network. The messaging paradigm, similar to in and out trays, lends itself to the concepts of being one part in a chain of business processes.

Application development is simplified. Often an application can be fully developed on a single machine without any communications and then deployed across the network without any changes.

Management of the messaging network, the location of the servers etc is now part of the IBM MQ configuration and not part of the application leading to simpler more reliable, flexible designs.

Architectural Goals

- Time Independence
- Location Transparency
- Platform Independence
- Many-to-Many Connectivity



Architectural Goals

N O T E S

IBM MQ Messaging has a number of architectural goals:

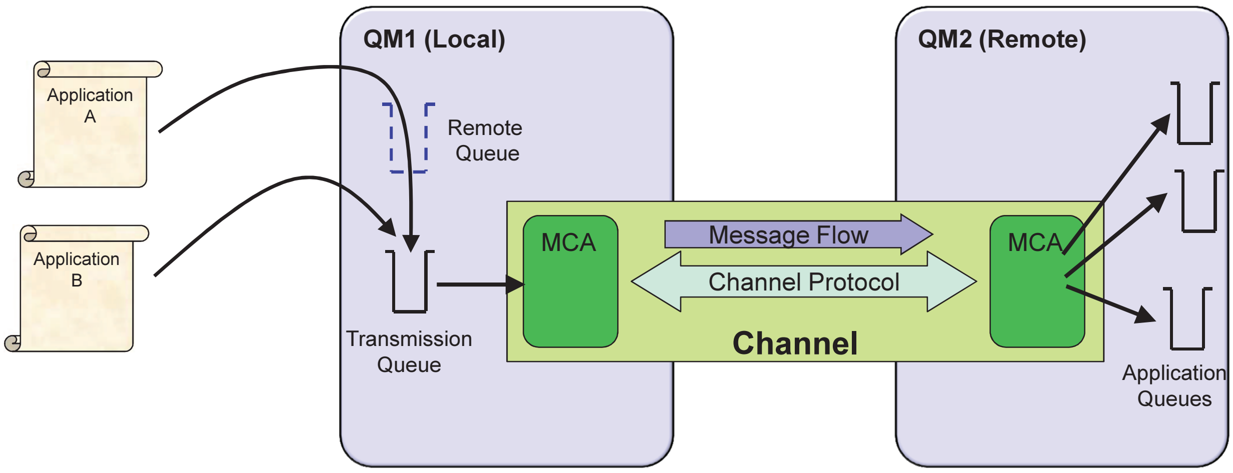
- Time independence
- Location Transparency
- Platform Independence
- Many-to-Many Connectivity

This presentation will demonstrate how these goals are achieved when going across networks of Queue Managers.

We can even add a couple of new goals:

- Protocol independence (TCP/IP, SNA etc)
- Application unaware

Channel Components



- **Transmission Queue accessed in different ways**
  - Remote Queue definitions
    - Gives Location Transparency
  - Direct Addressing via Queue Manager Name and Queue Name
    - Only partial location Transparency

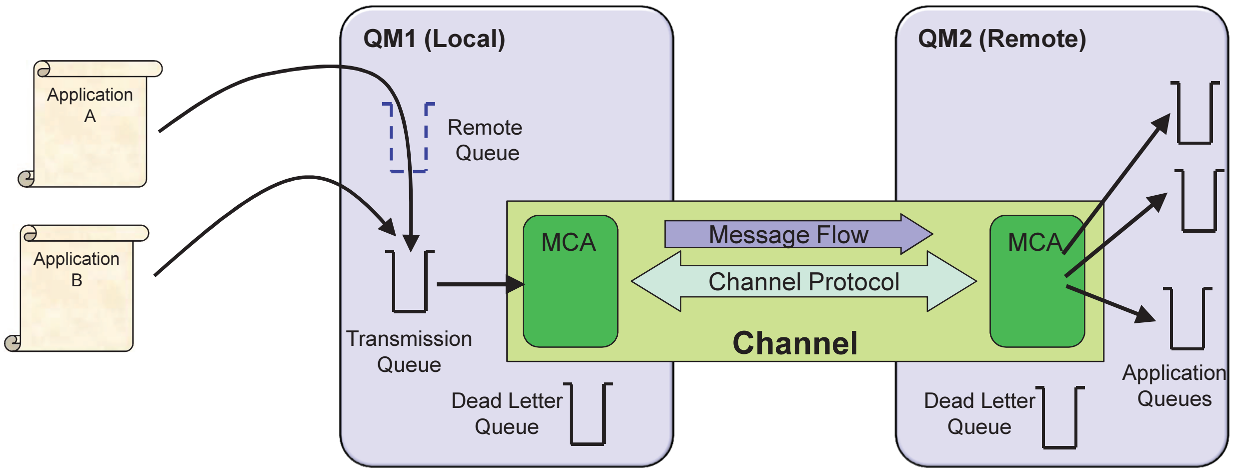
Channel Components

N O T E S

Channels are used by IBM MQ Queue Managers in order to exchange messages between Queue Managers. This chart illustrates the basic components.
When an MQ application opens a queue, it is necessary to have a queue name resolution process which will enable the appropriate destination to be targeted. In this case, the target destination is a remote queue - which resolves to a local transmission queue
There must be a local mechanism for safe storage of messages until they can be passed to the target Queue Manager and queue. This is the transmission queue. This queue is the same as any local queue - with the exception that the usage of the queue is designated as 'xmitq'. Note that transmission queue is often abbreviated to xmitq. Note that ALL messages destined for a remote Queue Manager must pass through a Transmission Queue.
There must be a process that is responsible for taking messages from the transmission queue and passing them to the remote Queue Manager using some underlying (provided) transport system. This process is called the sending Message Channel Agent (MCA).
There must be a process that is responsible for receiving messages and placing them onto target queues. This process is known as the receiving MCA.
A sending and receiving MCA pair - and the underlying transport are known as a IBM MQ channel.
There is a transport independent protocol defined for MCAs to facilitate the once-only, assured delivery of messages.

The channel architecture is defined to allow messages for any target queue on a particular target Queue Manager to be passed by a single channel. There is no need to have a channel per application or per target queue - although it will be illustrated later that this is a perfectly allowable thing to do !!

Dead Letter Queue



- **Dead Letter Queue….storage area for messages which are:**
  - Wrongly addressed, Queue Full, Unauthorised, etc
  - Badly formed, Too large, etc
- **Each Channel can decide whether to use Dead Letter Queue**
  - Setting of USEDLQ(YES | NO)

Dead Letter Queue

N O T E S

In order to accommodate error situations the Dead Letter Queue (DLQ) is also introduced into the distributed queuing 'architecture'. The DLQ is a Queue Manager attribute and is most heavily used by the distributed queuing component of IBM MQ, although there are some other occasional uses (eg. when a Queue Manager cannot put a Report or Trigger message). Note that the DLQ might be used at either end of the channel:

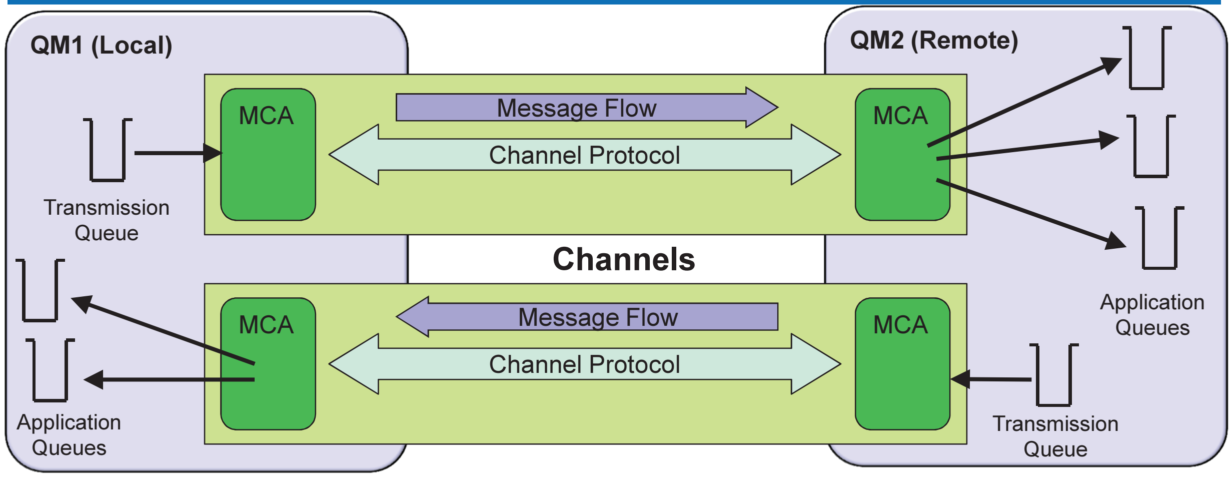
At the sending end, the DLQ will be used if a particular message is too large to be sent down the channel. This occurs if the message size exceeds the size of message defined for the channel even though it can be accommodated by the transmission queue.

At the receiving end, the DLQ will be used if the message cannot be placed (PUT) onto the target queue. Note that this might occur for any number of reasons, some of which may be transitory. The message retry parameters (defined for a channel) and the Message Retry exit are designed to handle these cases.

It should be noted that the use of the DLQ - at either side of a channel - may affect the order of delivery of messages. If there is some problem with a message which is a member of some sequence then this message will be DLQ'd and the message sequence will be interrupted.

Some 'lazy' applications require messages to always be delivered in the same sequence as they we sent. These applications require that there is no DLQ. The consequence of not having a DLQ is that a 'bad' message on a channel will terminate the channel and therefore prevent any application from receiving their messages. For this reason it is strongly recommended that installations do have a DLQ and that applications are modified to cope with messages to occasionally be delivered out of sequence. This modification is not difficult and there are various techniques which can be employed such as using the Correlation Id as a counter, having a secondary queue or using message segmentation.

Channel Concepts…



- **Channels are uni-directional**
- **Channels provide for (application) session concentration**
- **Two-way communication requires two channels**

Channel Concepts

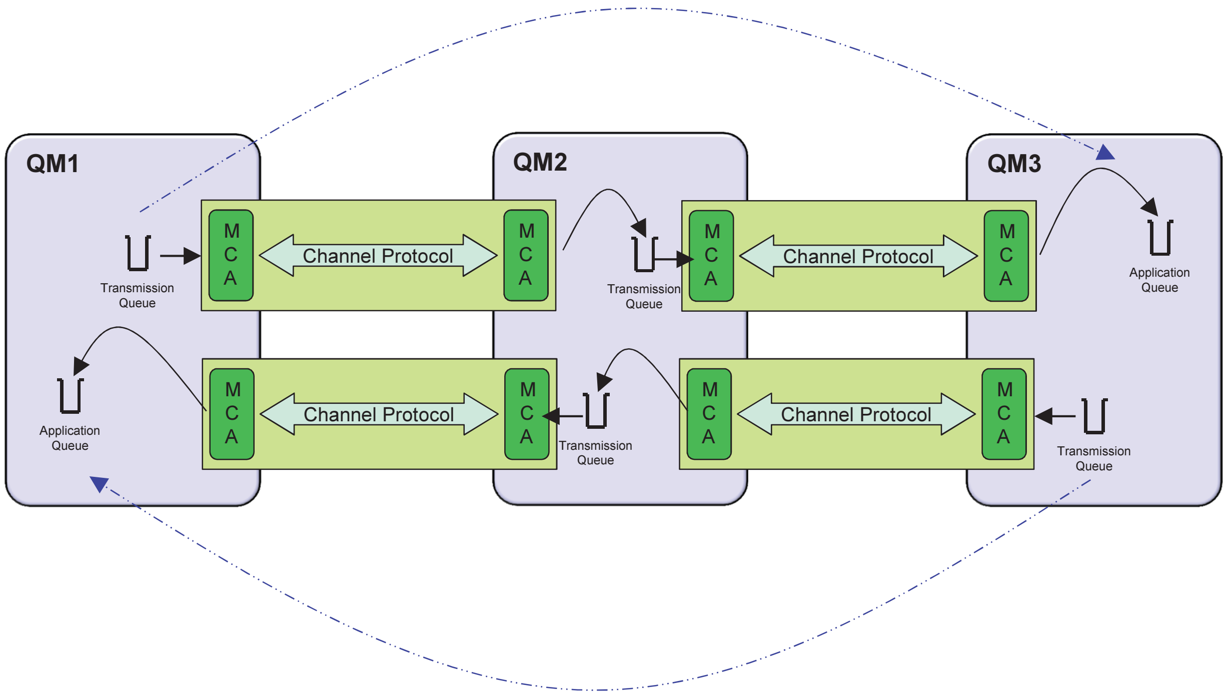
N O T E S

A channel is a one-way conduit between two Queue Managers. It provides a single pipe (session) through which all messages bound for a particular partner Queue Manager may be sent. This is in contrast to some mechanisms which require a pipe (session) per application. Because the channel is a one-way mechanism, IBM MQ messages may flow in one direction only. If two directional flow is required between Queue Managers then two channels are required.
Further, it is important to understand that although message flow for a channel is uni-directional, there are control packets flowing in both directions. Messages are not required to be passed directly to their target queue. It is quite acceptable for a message to be passed through one (or many) intermediate Queue Managers before reaching the final destination. This is simply achieved by defining appropriate transmission queues on the intermediate Queue managers.
The receiving message channel agent will place messages destined for remote Queue Managers onto these transmission queues. The method by which this is accomplished is explained later. This method of passing messages is usually known as multi-hop.
It is therefore possible to construct any topology of interconnected Queue Managers. Perhaps the most popular forms being :-

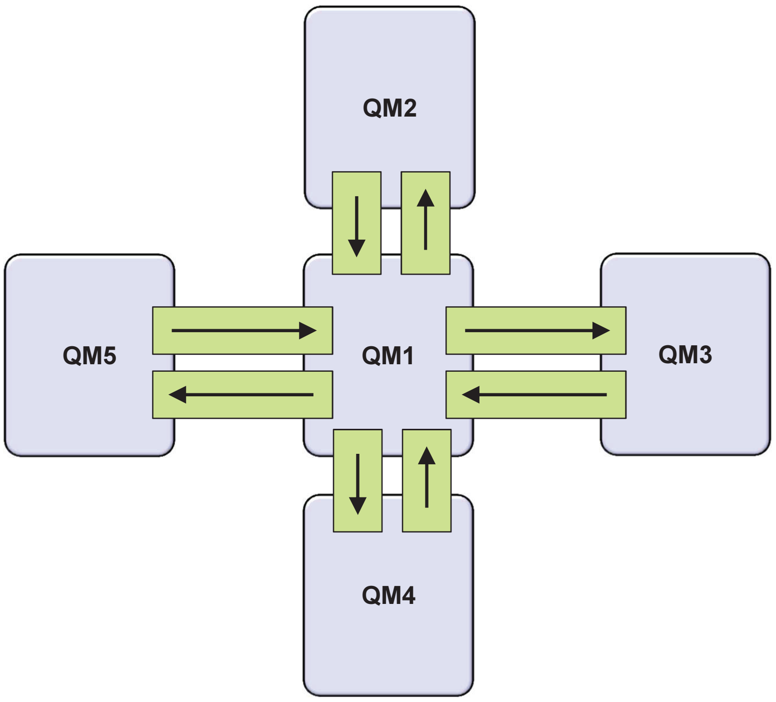
- 'any-to-any'
  - where every Queue Manager is directly connected to every other Queue Manager
- 'hub'
  - where a central Queue Managers routes messages to other Queue Manager.

Three-tier networks where MQI applications connect in to one of the leaf node Queue Managers via the client facility are also very popular.

Multi-hop



Hub Configurations



Defining Channels

- **Multiple channel types, (too) many parameters**
- **Most parameters may be left at default values**
- **Some parameters must be coordinated**
  - Channel name
  - Channel type
  - Sequence number wrap
- **Some parameters are negotiated**
  - Data conversion, MaxMsgLength, TransmissionSize, BatchSize, Heartbeat
- **Minimum Definition:**
  DEFINE CHANNEL(...) CHLTYPE(...) [TRPTYPE(...)]
  - plus CONNAME, XMITQ for some channels
- **Distributed can (if required) dynamically define Receiver channels**

Defining Channels

N O T E S

At first glance a channel definition can seem really complicated. In reality there are only a few attributes which need to be entered. The majority of parameters can be left to their default values. As the user becomes more familiar with the operation of channels then these other parameters can be adjusted if and when required.

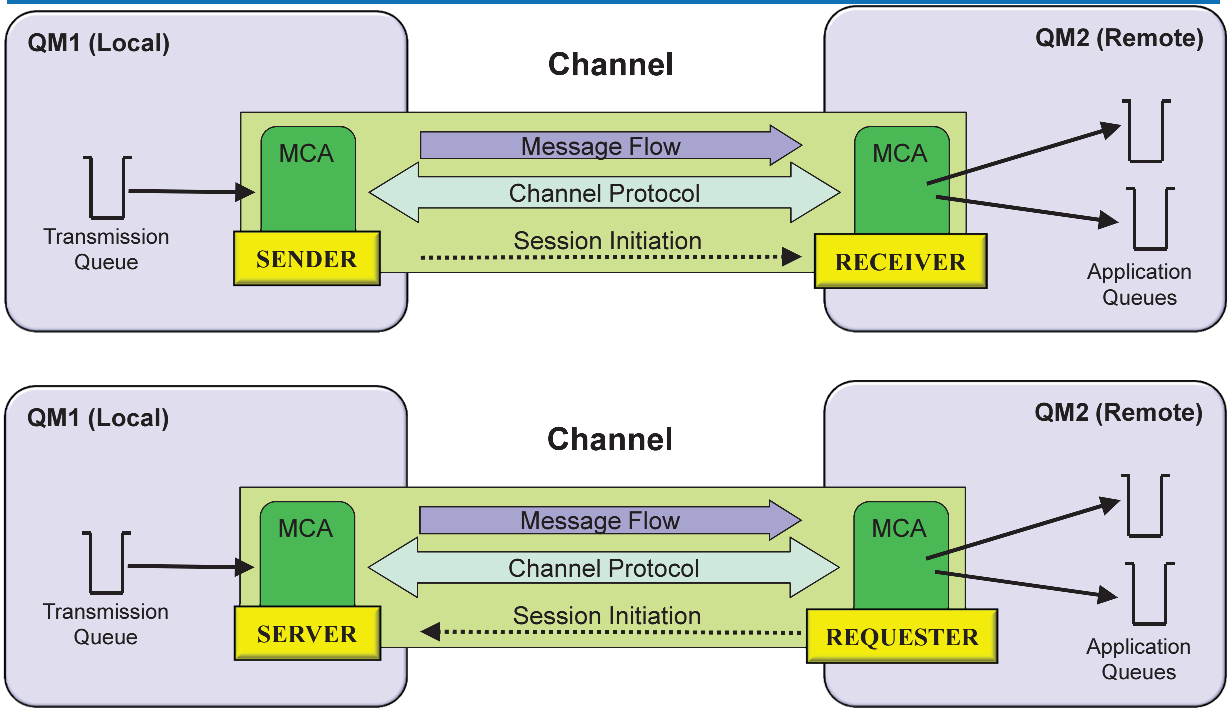
A channel definition can be considered as a template for communication. Each definition merely tells the Queue Manager the way in which you'd like it to communicate with the outside world.

The most important value is the 20 character name of the channel itself. The key point about the channel name is that it is what ties the two ends of the channel together. There must be a channel definition of the same name at both ends of the channel. The channel name must match exactly, it is case sensitive as with all IBM MQ names.

The next most important attribute is the Channel type. The type of the channel determines what actions are allowed on the channel. The two ends of the channel must be defined with compatible channel types, described later.

It is possible in the Distributed products to have the inbound channels defined automatically. This can reduce the amount of administration required if the local queue manager is receiving messages from many other Queue Managers. This facility is known as channel auto-definition and is controlled by the CHAD Queue Manager attribute.

MCA Types



MCA Types

NOTES

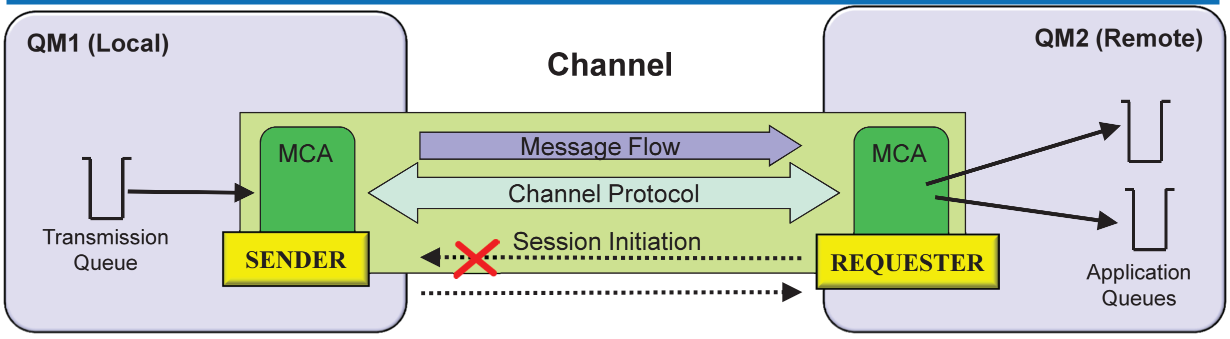
There are six different types of MCA - Sender, Server, Requester, Receiver, Cluster Sender and Cluster receiver. When MCAs connect to a partner MCA, there must be a compatible pairing - as illustrated on the previous chart. There are several things to remember about these MCA pairings:

- The MCA type controls which side of the channel initiates the connection. Generally, Senders and Requesters initiate connections and Servers and Receivers respond to connections.
- MCA types beginning with 'S' are associated with transmission queues from which they take messages to send to partner Queue Managers.
- MCA types beginning with 'R' are associated with target application queues as they will receive messages from the sending MCA.

It is possible to combine MCA types and have an MCA with multiple 'personalities'. A Server MCA may function as a Sender MCA by defining a partner address in the channel CONNAME parameter. This means that an MCA pairing of Server (with CONNAME) and Requester may be started from either partner Queue Manager. In fact, there is rarely any need to use any other MCA type than Servers and Requesters.

There are also two other MCA types, defined for the MQI client/server connections. These MCA types implement the IBM MQ Client connections and are beyond the scope of this presentation.

MCA Types



- **Requester/Sender combination implements 'call back'**
  - Both ends of the channel identify who they want to talk to
  - Can offer poor mans security
    - Usually better to configure something like SSL/TLS instead
  - Requester/Server is usually a better choice
    - since server channels can be fully qualified with CONNAME
    - only one channel negotiation so more efficient

MCA Types

N O T E S

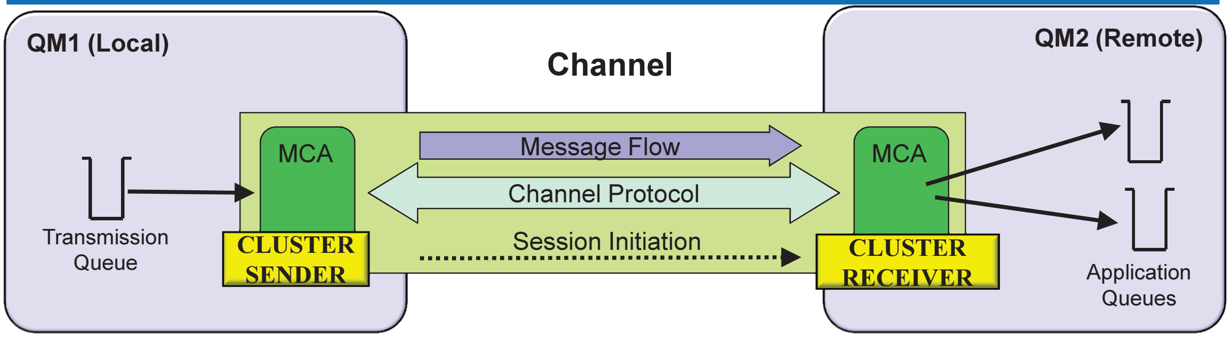
Normally a REQUESTER channel would be paired with a SERVER channel. This pairing has the advantage that it can be started from either end of the channel.

However, some installations are nervous about allowing a channel to connect in from somewhere in the network and drain their transmission queue. The full solution to this would be to install security exits or configure security such as SSL/TLS and ensure that the inbound REQUESTER channel really is authorised to receive the messages. However, another simple solution is to use the 'call back' approach.

By defining a SENDER channel, rather than a SERVER channel, the user can force a channel call-back. Any inbound connection to the SENDER channel will be ended. The sender channel will then re-connect back to the connection name defined in its own definition.

Since the connection name of the SENDER channel is used to make the call-back this technique can not easily be used in cases where the receiver of messages is using DHCP or similar. In other words, this technique really requires that the REQUESTER channel is on a machine with a well-known network address.

MCA Types: Cluster Support



- **Cluster Channels**
  - Defined as Channel Object and/or in the cluster repository
    - Priority given to repository definition
- **Same commands and attributes as previous Channel types**
- **Cluster Senders can share a single transmission queue or have their own**
  - SYSTEM.CLUSTER.TRANSMIT.QUEUE
  - Messages separated on transmission queue by CorrelId
    - CorrelId = Channel Name

MCA Types: Cluster Support

N O T E S

IBM MQ Clusters allow automatic dissemination of queue and channel definitions across the MQ network. This leads to less administration overhead and greater responsiveness to network changes.

In addition, queue manager Clustering provides load balancing support for multiple backend servers by dynamically choosing target destinations.

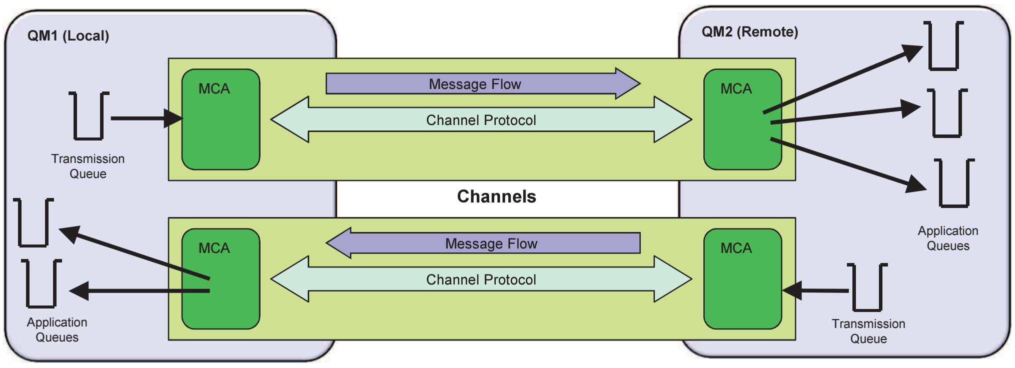
The essential idea is that a few (two) identified Queue Managers are given the responsibility of being full repositories for the cluster – that is that they know about all elements in the cluster. When a Queue Manager joins a cluster it contacts these full repositories and, as applications try to use the cluster resources, the definitions needed are automatically sent to the Queue Manager. This means that the Queue Manager automatically finds out about any queue definitions and channel definitions it needs to communicate with the cluster. In a cluster, therefore, it is not necessary to define remote queues or channels other than the ones needed to communicate with the full repositories.

So, clustering uses the same basic concepts of Channels but employs a number of other ideas and is therefore beyond the scope of this introductory presentation.

Channel Definitions

QM1

```
DEF QL(QM2)      USAGE(XMITQ)
DEF CHL(TO.QM2) CHLTYPE(SDR) TRPTYPE(TCP) CONNAME(QM2.MQGEM.COM) XMITQ(QM2)
DEF CHL(TO.QM1) CHLTYPE(RCVR)
```



QM2

```
DEF QL(QM1)      USAGE(XMITQ)
DEF CHL(TO.QM1) CHLTYPE(SDR) TRPTYPE(TCP) CONNAME(QM1.MQGEM.COM) XMITQ(QM1)
DEF CHL(TO.QM2) CHLTYPE(RCVR)
```

Channel Definitions

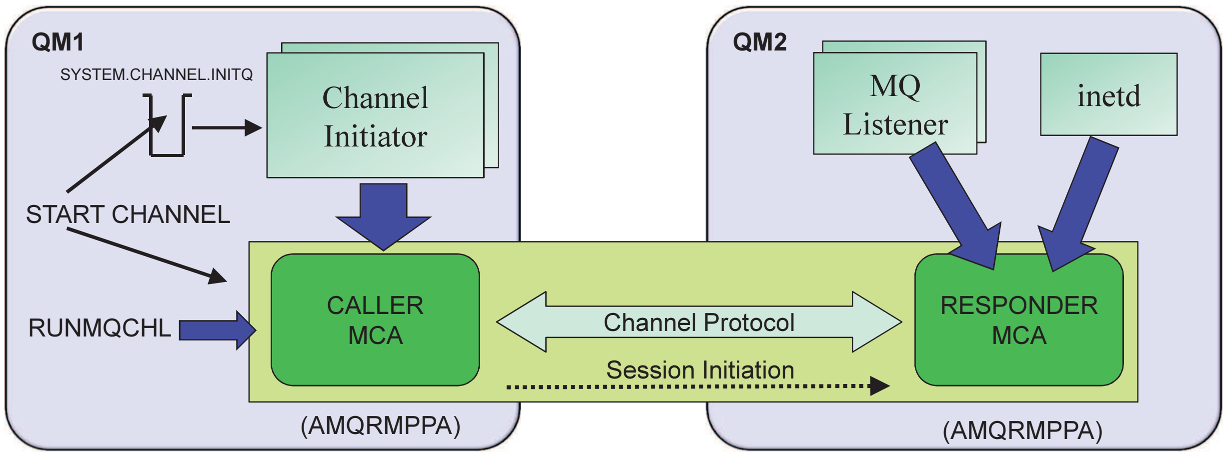
N O T E S

Channels are just Queue Manager objects. There are created, displayed and monitored by MQSC commands or PCF messages. The commands are :-

```
DEFINE  CHANNEL  or DEF CHL        Create a new channel
DISPLAY CHANNEL  or DIS CHL        Display a channel definition
DELETE  CHANNEL  or DEL CHL        Delete a channel definition
ALTER   CHANNEL  or ALT CHL        Alter an existing channel definition
RESET   CHANNEL                    Reset Channel Sequence Number
RESOLVE CHANNEL                    Resolve a channel in-doubt
START   CHANNEL                    Start a channel
STOP    CHANNEL                    Stop a channel from running
DISPLAY CHSTATUS or DIS CHS        Displays the status of a channel
```

Channels do have a lot of attributes associated with them but as you can see from the example most can be left to the default values.

Starting Channels



- **Caller and Responder MCA**
  - Most channel types can operate as either
- **Channels can run as threads or processes on Distributed**
  - Threads are most common
  - Processes offer good isolation if you don't trust your exits.

Starting Channels

N O T E S

For a channel to start successfully clearly both ends of the communication must be started. The end which initiates the conversation is known as the Caller and the end which accepts the connection request is known as the Responder. The starting of these two ends are very different :-

- **Caller**
  Caller channels are started in the following ways
  - An explicit MQSC or PCF commands i.e START CHANNEL(FRED)
  - Run directly from the command line i.e. RUNMQCHL -c FRED
  - Triggered by messages arriving on the transmission queue.
    - Requires configuration of the transmission queue and Channel Initiator

On most platforms caller channels can be run as a process or a thread. For a Caller channel this is determined by the MCATYPE Channel attribute

- **Responder**
  Responder channels are started by a network connection request being received by some form of listening process. This listening process varies depending on the protocol and platform but most common are MQ Listener and inetd.

The Responder MCA will run as a thread or process depending on how it has been started. For example, the MQ Listener will start it as a thread. inetd will start it as a process (amqcrsta).

Stopping Channels

- **Self Terminate**
  - Empty transmission queue and expired DISCINT
  - Channel is available for restarting
  - Should really set up triggering
  - Recommended method for channel control
- **Manual Stop**
  - Explicit command
    - MQSC/PCF Stop Channel
    - FORCE or QUIESCE
  - Channel is STOPPED ie. DISABLED
    - Unless you specify STATUS(INACTIVE)
  - Transmission Queue is Get(Disabled)
  - explicit command required for channel restart
  - MCA shuts down as soon as possible
    - May take until next heartbeat for receiver channels if idle
  - STOPPED state maintained across Queue Manager termination
- **Queue Manager shutdown**
  - All channels are ended ... but not STOPPED

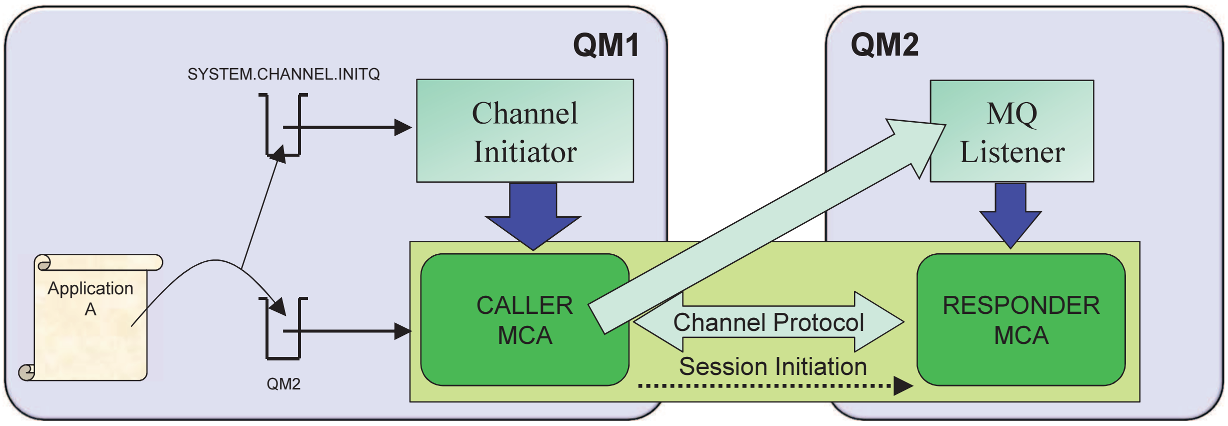
Stopping Channels

N O T E S

There are two distinctly separate ways of stopping a channel :-
...allowing the channel (the MCA actually) to control its own destiny and to self-terminate when there are no more messages to be sent to the partner Queue Manager. This is the preferred method of operation and - in particular - it allows the channel to be restarted (automatically) at some later time. This controlled by the DISCINT channel attribute.

...or using an MQ system commands to explicitly stop it. The major difference between this and the former method is that explicitly stopping a channel will DISABLE that channel by putting it in STOPPED state. This means that the channel will not be available for (automatic) restart unless the channel is explicitly enabled - using some form of the START CHANNEL command. When a channel is stopped, there are two variations (modes) of the stop request, as follows:
The quiesce mode means that the channel is stopped at the next available opportunity. For a sending MCA, this means that the current message is processed and then commit processing is completed. This means that the channel should not be in-doubt when the channel is stopped. For a receiving MCA, it is not possible to stop the channel until the end of the current batch - as this is the first opportunity that the receiving MCA has to get control of the channel. This has significant consequences if the sending MCA is waiting on an empty transmission queue and the receiving channel does not get the chance to indicate its shutdown intentions for a significant length of time.
The force mode will immediately terminate the MCA and then the channel. This means that the current batch of messages will be backed out and, in some circumstances, the channel will be in-doubt. Use of the force mode of STOP CHANNEL is not recommended except in unusual circumstances.
When a channel is stopped, the transmission queue associated with the channel, for non-cluster channels, is set to NOTRIGGER and ‘Get(Disabled)’. This is done to prevent inappropriate activity for a STOPPED channel. These actions are reversed for a START CHANNEL and so, in normal circumstances, there will be no adverse effects of these actions.
When the Queue Manager shuts down it will stop all of the channels associated with the Queue Manager. In contrast to a 'normal' Stop Channel, this activity does not set the transmission queue to NOTRIGGER and Get(Disabled). Thus, when the Queue Manager restarts, the channels are available for restart - particularly useful for triggering.

Automatic Operation



```
DEFINE QLOCAL(QM2) USAGE(XMITQ)
       TRIGTYPE(FIRST) INITQ(SYSTEM.CHANNEL.INITQ)
       TRIGDATA(TO.QM2)   [Optional]

DEFINE CHANNEL(TO.QM2) CHLTYPE(SDR) TRPTYPE(TCP) CONNAME(QM2.MQGEM.COM) XMITQ(QM2)
       SHORTRTY(10) SHORTTMR(60) LONGRTY(999999999) LONGTMR(1200)
       DISCINT(600)
```

Automatic Operation

N O T E S

On an installation of any appreciable size you will want to set up your channels to start and end automatically. Doing so will save you a huge amount of effort. Luckily it is very simple to do.

Starting

Channels are started using the standard MQ Triggering mechanism but there are a few key differences:

- The trigger monitor that is used is the Channel Initiator
- The process object is optional
  A process object tells a trigger monitor 'what' to start. In the case of the Channel Initiator there is never any
- The Channel Initiator, unlike the standard Trigger Monitors, will also reattempt the start at regular intervals according the channel definition.
  Use sensible values for retry – do not ask the channel to retry every second for 10 hours!

Stopping

The ideal way for a channel to end is just for it to end naturally when it runs out of messages. The channel definition parameter which controls this is DISCINT with a default value of 6000. This is 1 hour and 40 minutes. Personally I think this is a bit on the long side and would reduce it to something more meaningful. Clearly you don't want to set it too low though, otherwise your channel will be constantly bobbing up and down.

Channel Status

- **Displayed with command DISPLAY CHSTATUS**

```
eg. DIS CHS(TO.QM*) yields
          CHANNEL(TO.QM2)              XMITQ(QM2)
          CONNAME(QM2.MQGEM.COM)       CURRENT
          CHLTYPE(SDR)                 RQMNAME(QM2)
          STATUS(RUNNING)              SUBSTATE(MQGET)
```

- **Lots of other information such as bytes sent, remote version etc**

- **Status values**

▶ STOPPED	- Channel is currently 'disabled'
▶ STARTING	- Waiting for available resources to start
▶ INITIALIZING	- MCA program has been started
▶ BINDING	- Channel is negotiating with partner
▶ RUNNING	- Channel is running normally
▶ PAUSED	- Receiving channel is trying to put a message to target queue
▶ STOPPING	- Channel is stopping
▶ RETRYING	- Channel is periodically retrying connection

- **Only active channels displayed with DIS CHSTATUS(*) CURRENT**

Channel Status

NOTES

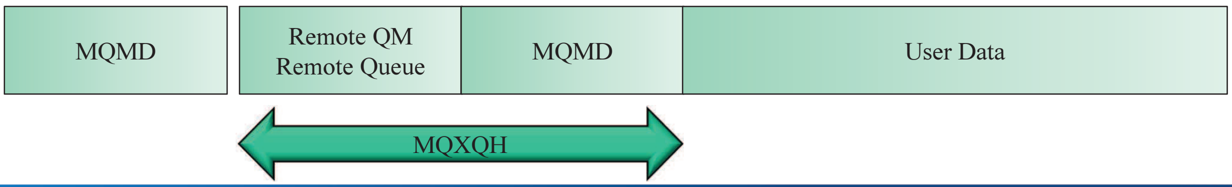
Channels have two types of status information, known as current and saved. Current status is what we're looking at here and gives information relating to the activity of the channels at the instant the command is issued. Saved status is the information written to disk by a channel to ensure assured message delivery. It is obtained by adding the keyword SAVED to the DIS CHSTATUS command and is beyond the scope of this presentation.
The status values are:-

- STOPPED - A channel will be in this state if the user has issued a STOP CHANNEL command or a channel has completed it's set of retries when trying to connect to a partner. Must be started manually by the user.
- STARTING - It is possible to configure MQ to only allow a certain number of channels to be running at any one time. If a channel would like to start but this limit has been reached it will be in this state waiting for one of the running channels to end.
- INITIALIZING - An MCA program needs to perform some initialisation before it gets into 'binding' status, such as connecting to the Queue Manager. During this phase it will be in initialising state. This phase is very brief and unlikely to be seen by users.
- BINDING - This is the status of the channel during the whole negotiation phase including trying to initially make contact with the remote machine.
- RUNNING - Channel is running normally
- PAUSED - If a receiver channel fails while trying to put a message to a target queue it may reattempt the put according to the MsgRetry values and MsgRetry exit. A channel in paused state suggests that the channel is having difficulty putting it's messages to the designated queues.
- STOPPING - When an user issues a STOP CHANNEL command it may be some time before the channel actually ends depending on the values used on STOP CHANNEL.
- RETRYING - If a sending channel ends abnormally and its definition includes Retry values it will enter Retry state. Retry essentially signifies a wait state before the attempt to start the channel. When a channel is in Retry no actual MCA program is running.

If there are multiple instances of the same channel running from different partner machines each instance will be reported. The CONNAME field can be used to determine the instance.

Remote Addressing

- **Remote messaging requires:**
  - Remote Queue Name
  - Remote Queue Manager Name
  - Route to remote Queue Manager
    - Transmission Queue and Channel
- **Application issues MQOPEN which is resolved to transmission queue:**
  - Remote Queue definition eg MQOPEN(QM1, RQ1)
    - provides location transparency
  - Explicit addressing eg. MQOPEN(QM2,XYZ)
    - limited location transparency
- **Destination data is placed at the head of the message on Transmission Queue:**



Remote Addressing

N O T E S

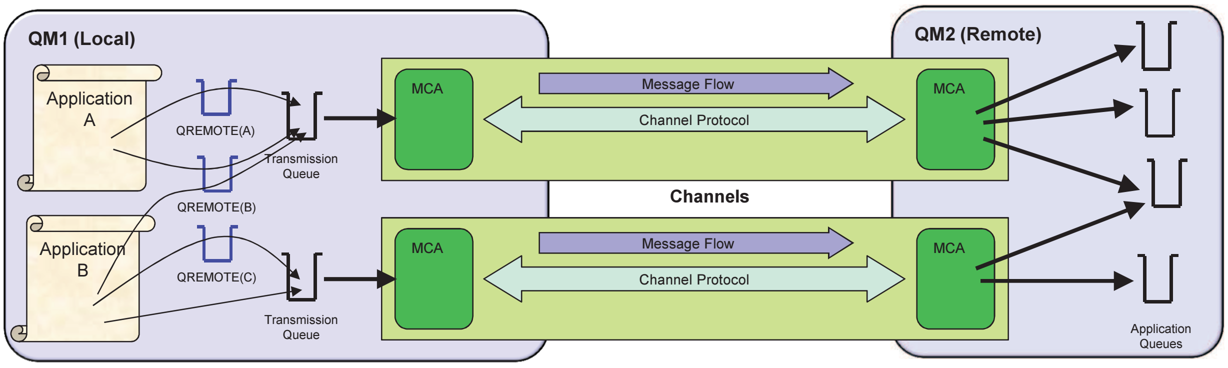
There are quite a number of different ways that you can configure your MQ definitions so that applications moves messages from one machine to another.

All messages going between Queue Managers will do so via transmission queues so the key part of the configuration is to associate the correct transmission queue with the correct target Queue Manager although this may be just the next Queue Manager along the route.

The rules the Queue Manager follows are :-

- Is there a transmission queue with the same name as the target Queue Manager ?
- Is there a remote queue definition with the name of the target queue ?
- Is there a remote queue definition with the same name as the target Queue Manager ?
  - This is known as a Queue Manager alias and is described later
- Is there a cluster queue manager object with the same name as the target Queue Manager ?
  - Clustering is really beyond the scope of this presentation
- Is there a default transmission queue associated with the Queue Manager ?
  - A default transmission queue can be used to construct your Queue Managers in a hierarchy. The lower Queue Managers pass any messages destined for Queue Managers they don't know to the higher Queue Manager in the tree. Clearly the uppermost Queue Manager must know the route to all Queue Managers.

Parallel Channels



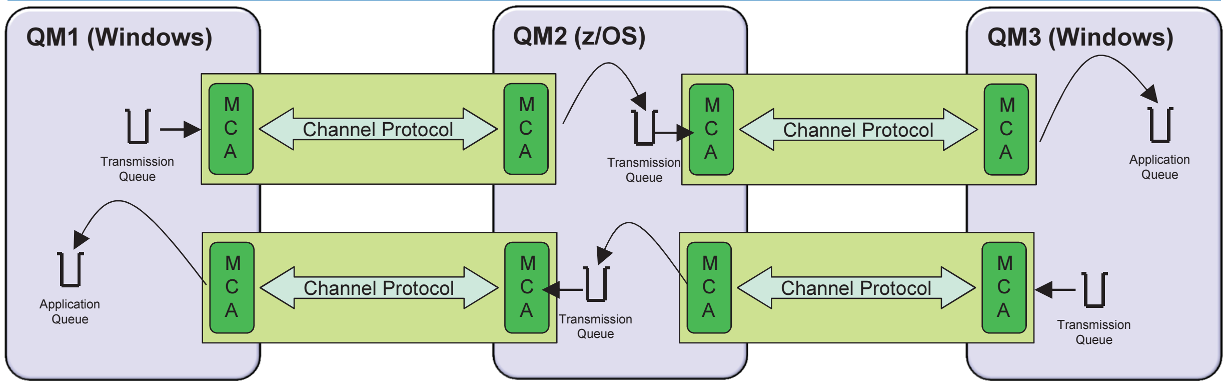
- **Different Quality of Service for different message types**
  - Message Size
  - Message Security
  - Message Priority
- **Multiple channels to handle additional load**
- **Often requires Alias definitions**

Parallel Channels

N O T E S

Although a single channel will pass any and all messages to a target Queue Manager, it is quite acceptable to have multiple (parallel) channels between a pair of Queue Managers. There are a number of reason that this might be appropriate:-
For performance reasons, a second channel is provided. Note that this will be necessary only if the 'first' channel is very heavily loaded - as channels work most efficiently when they are fully utilized. It is difficult to predict the improved throughput of a secondary channel but it is not generally linear. In other words, a second channel will not double throughput. Obviously there are certain restrictions on a machine which can not be overcome in this way. The machine has a fixed bandwidth on its communications, it's disks spin at a certain speed and there's only so much CPU available. Users looking to improved the performance of channels should first determine where the bottleneck on there system lies (Disk, Network or CPU). Having said that a second channel often yields a throughput increase of 20% -> 30%.
If different classes of messages are being handled then it may be appropriate to use multiple channels. For instance there may be a justification for having very large messages passed on one channel and other, smaller messages passed on another channel. This is usually referred to as Quality of Service.
If different transports or classes within a particular transport are needed then it may be appropriate to use parallel channels. A common example of this would be security. It may be useful to have a channel which performs high levels of encryption reserved for sensitive data.
The use of parallel channels means that there has to be multiple ways to direct messages to the same partner Queue Manager. This will require either multiple Remote Queue definitions, which resolve to different transmission queues or the use of Queue Manager Alias definitions.
Clustering supports parallel channels by allowing multiple CLUSRCVRs to be defined.

Data Conversion



- **Channels will not convert application data by default**
  - Leads to unnecessary work in multi-hop
  - Two opposite translations doesn't always yield original data
  - Applications should use MQGET(GMO_CONVERT)
  - Channel will always convert headers eg. MQMD
- **Can ask channel to perform conversion but not advised**
  - CONVERT(YES) channel attribute
  - Only works if next hop is final target!

Data Conversion

NOTES

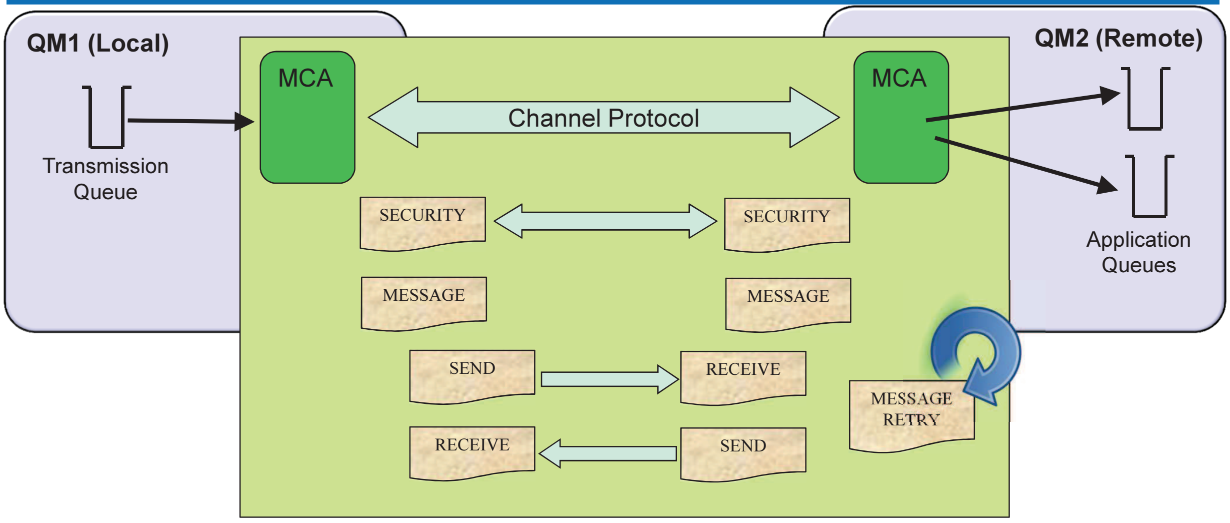
Channels will always convert the Message Descriptor (MQMD) and IBM MQ headers as messages are moved between Queue Managers. However, the application message itself is not converted by default. It is recommended that all applications always issue MQGMO_CONVERT on the MQGET's so that the message is converted into the codepage and encoding required by the application. There is no overhead issuing MQGMO_CONVERT if the arriving message is already in the correct codepage/encoding. In those cases the conversion exit isn't even called.

If absolutely necessary the channels can be asked to perform the conversion of the application messages with the CONVERT(YES) attribute. This is not recommended though, Convert(YES) was provided on the Channels as an interim mechanism when there were still Queue Managers which didn't support the MQGMO_CONVERT option. In those cases the only choice was to convert the data before it was sent to these Queue Managers.

If using CONVERT(YES) the administrator should be aware of the following....

- The Channel does not know the final destination of the message. It will convert the data to the codepage/encoding of the target Queue Manager. If the message is subsequently forwarded to a different Queue Manager or if the application wanted it in a different format then this will obviously not work.
- In a multi-hop environment the data will be converted more times than necessary. Consider this network sending messages Windows -> z/OS -> Windows. In this case no conversion at all is necessary. However, with CONVERT(YES) the data will be converted twice. Worse still, since not all ASCII characters have EBCDIC equivalents the resulting data may not be exactly the same as the original.
- Once a channel has been made available with CONVERT(YES) it will encourage lazy programming where applications rely on the channel to do the conversion. Consequently it will be very difficult to revert back to CONVERT(NO).

Channel Exits



- **Installation written code at key points for adding:**
  - Authentication & Encryption
  - Logging & Auditing
  - Installation specifics

Channel Exits

N O T E S

There are five different channel exit points defined. These exit points allow channel processing to be customised to the requirements of an individual channel. Note that the exits are specified by channel and so it is possible to have different actions for different channels.

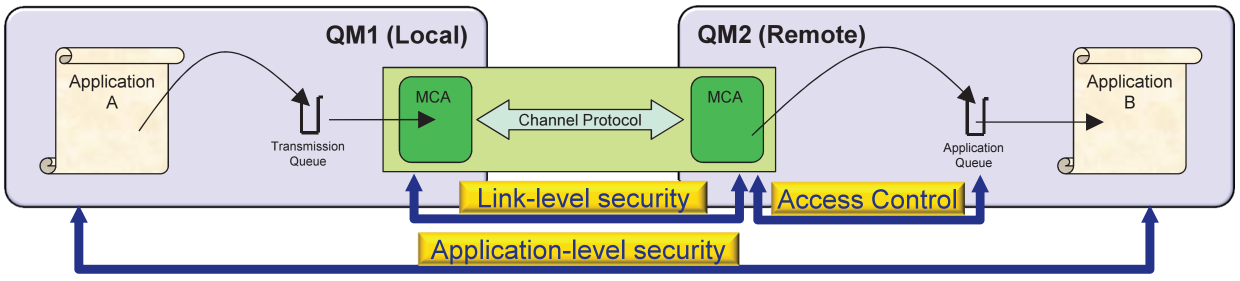
- Security exits are more appropriately termed authentication exits. Their purpose is to allow mutual authentication of the partner MCA for security purposes - to ensure that the partner Queue Manager (or MCA, at least) is valid. The security exits associated with the two MCAs for the channel can exchange an arbitrary amount of data and normally operate as (cooperating) pairs with either exit able to terminate the channel connection.

- Message exits are called once per message on each side of the channel. On the sending side, the message exit is called immediately after the message is taken from the transmission queue. On the receiving side the exit is called just before the message is put to the target queue. These exits are used for message specific processing such as accounting (message tracking), message integrity and privacy.

- Send/Receive exits are called once per message segment. If a message is too large to be transmitted as a whole, it is segmented and transmitted in pieces. The Send exit is called when sending each segment and the receive exit is called on receipt of each segment. This exit would be used for data (rather than message) specific functions such as compression. As mentioned previously, there are private flows that the MCAs use as a part of the message exchange protocol. These flows also cause the Send/Receive exits to be called. Thus the Receive exit may be invoked for Sender/Server MCAs - which are only sending MQ messages and not receiving them.

- If messages can not be PUT to the target queue a Message Retry may be called either to retry the PUT or to redirect the message to an alternate queue. Use of this exit is an effective way to implement an overflow queue when a particular target queue is full (due to its serving application being inactive). Clearly, it may also be used for any other situation where a target queue cannot be opened or a put fails.

Security



- **Link-level security**
  - Authentication
    - SSL/TLS
    - CHLAUTH
    - Security Exits
  - Encryption
    - SSL/TLS
    - Message Exits
- **Access Control**
  - MCAUSER set by…
    - Security Exit
    - CHLAUTH
  - PUTAUT
- **Application-Level security**
  - Advanced Message Security (AMS)
  - Application 'roll-your-own'

Security

N O T E S

Security is a big subject in its own right. There are lots of possibilities and what you wish to implement will depend on a number of factors such as the importance of the messages, the sensitive nature of their content, the physical security of your servers and network and your general attitude toward security and level of paranoia!

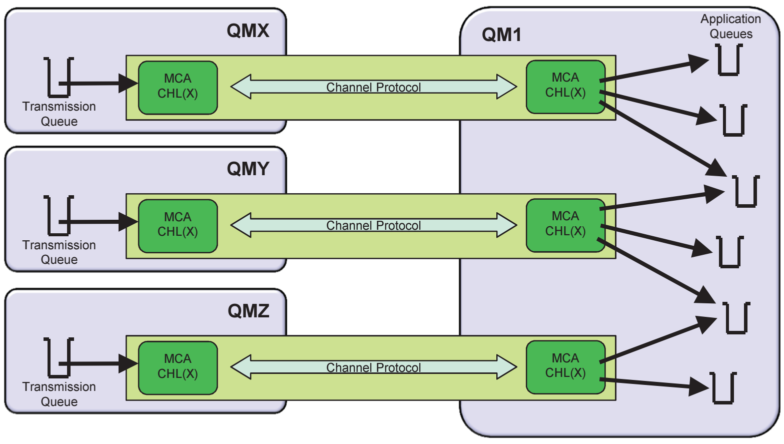
IBM MQ allows you to run your channels with next to no security or you can make them ultra-secure if you wish.

There are a number of vendors who will offer security solutions which make use of Channel exits, these methods were very popular in the early days of IBM MQ. Nowadays though, with the advent of SSL/TLS support and CHLAUTH built-in to the product, these features tend to be used more as the way of securing your link-level security. The downside of SSL/TLS, of course, is that you need to manage the certificates. However, most users of MQ are already managing SSL/TLS certificates for other aspects of their installation so the overhead is minimal.

Access control is done using the normal IBM MQ access control and is based off the channels effective userid. This is the channels MCAUSER field. It can be set at define time and can also be modified at run-time by Security Exits and CHLAUTH definitions. Using the PUTAUT setting you can also specify whether the channel should use the MCAUSER for all the access control checks or whether it should use the userid contained within the message itself.

If you are very concerned about security then you may wish to also configure application-level security. The easiest approach is to use AMS. AMS allows MQ to encrypt messages before they are even put on the transmission queue. The encryption is user based which means that it is only the intended recipient who will be able to decrypt the data. The advantage of this is that messages are encrypted, and tamper-proof, throughout their whole journey through the MQ network, even while sitting at rest on a transmission or application queue. This means that even an Administrator, with access to the logs, the transmission queue, product traces etc, would not be able to view or change the data.

Generic Receiving Channels



```
MQSC on QM1

DEFINE CHANNEL(X)
       CHLTYPE(RCVR)
```

- **Systems Management issues:**
  - Can be slightly harder to see what is going on with DISPLAY CHSTATUS
  - Stop channel may require use of CONNAME
  - RESET CHANNEL awkward to use at receiver end
    - Shouldn't really be using it that end anyway!
- **Benefits usually outweighs the disadvantages**

Generic Receiving Channels

N O T E S

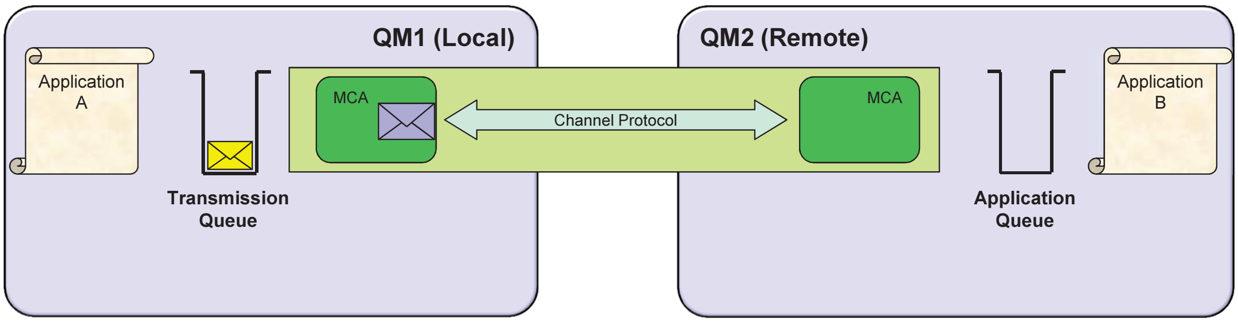
When queue managers connect to one another there needs to be a channel pair defined, one definition in each queue manager. While a sending channel definition and instance must be unique (by name) within a particular queue manager, this restriction does not apply to a receiving channel (either a Receiver or a Requester). Thus, many connecting queue managers may connect to different instances of the same receiving channel definition. There is an internal IBM MQ table which holds information about running channel instances and each receiving channel in this table is further qualified by the name of the partner queue manager. Thus, so long as queue managers are uniquely named within the network, the number of receiving channel definitions may be minimised.

While this facility is very useful in terms of reducing the numbers of channels that need to be defined within a queue manager, there are some issues. With (potentially) so many receiving channels that have the same name, it may be difficult to distinguish one channel from another - from a systems management viewpoint. It is necessary to use the ConnectionName to differentiate one receiving channel instance from another. Note also that the stop channel command, if issued for a receiving channel will attempt to stop all channels matching the channel name in the command so again you may need to qualify the STOP CHANNEL command with the connection name so that only the specific instance will be stopped.

In practice the restrictions on the granularity of the commands is not a problem and this facility is useful for reducing the numbers of channels that need to be defined within a particular queue manager.

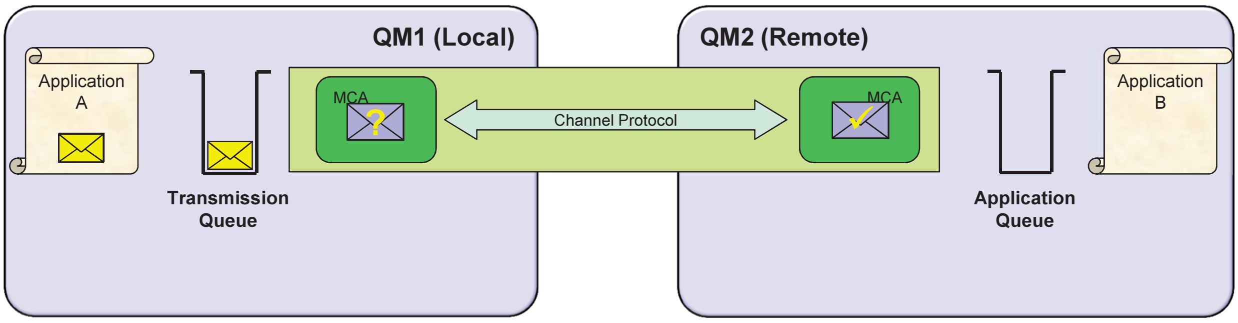
A more comprehensive solution to the issue of MQ channel management is to use queue manager Clusters.

Channel Format & Protocol (FAP)



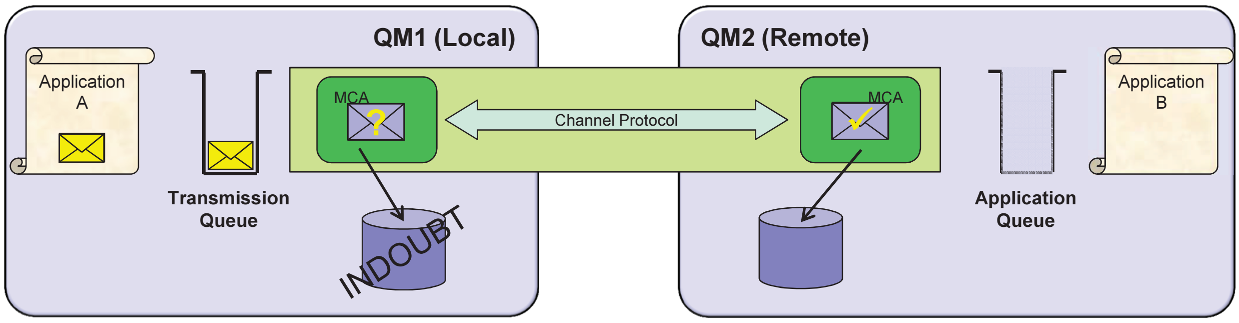
- **Binding Phase**
  - Establishing authentication
  - Running any security exits
  - Negotiating channel parameters such as:
    - Heartbeat Interval
    - Batch Size

Channel Format & Protocol (FAP)



- **Running phase (Non-persistent messages)**
  - Message sent in ‘batches’
  - Sender periodically confirms batches with remote channel

Channel Format & Protocol (FAP)



- **Running phase (persistent messages)**
  - Message sent in ‘batches’
  - Sender periodically confirms batches with remote channel
  - Every batch will cause the sender channel to, briefly, go indoubt

Channel Format & Protocol (FAP)

- **Once/once-only message delivery**
- **Resynchronisation process for failed channel**
- **Also:**
  - Negotiation for mismatching parameters
  - Message segmentation for large messages
  - Automatic header data conversion
- **Channel batching**
  - Messages are grouped into batches for efficiency
  - BatchSize - maximum number of messages sent before confirmation
  - BatchInterval - minimum time that a batch lasts
    - Batch is usually less than the maximum ... transmission queue empties
- **Specific processing for non-persistent messages**
  - Fast Messages ... for interactive messaging

Channel Format & Protocol (FAP)

NOTES

The MCA operation conforms to a quite simple model:

```
Do until DISCINT or STOP CHANNEL
{
    Do until batchsize or no more messages
    {
        Local MCA gets a message from the transmission queue
        A header is put on the data and sent using APPC, TCP etc.
    }
   Send "End of batch flag"
   Remote end commits and sends "OK" flag back
   Local end commits
}
```

If there is any failure in the communications link or the MCA processes, then the protocol allows for resynchronisation to take place and messages to be appropriately recovered.

Probably the most misunderstood part of the message exchange protocol is Batchsize. Batchsize controls the frequency of commit flows used by the sending MCA. This, in turn, controls how often the communications line is turned around and - perhaps more importantly - how quickly messages at the receiving side are committed on the target application queues. The value for Batchsize that is negotiated at channel startup is the maximum Batchsize only - if the transmission queue becomes empty then a batch of messages is automatically committed.

In order to accommodate channels where the transmission queue becomes empty too frequently and causes too much commit activity Batch Interval was introduced. This defines the minimum time in milliseconds for which a batch will be open.

Ideal Channel Configuration

- **Use sensible Queue Manager names**
  - Not too large, unique !!, and ideally meaningful in some way
- **Channel Name based on target Queue Manager**
  - Use generic receiver channels
- **Transmission Queue name = Target Queue Manager Name**
  - No need to invent a new name
- **Use triggering to automatically start your channels**
  - Use sensible retry values
  - Use a reasonable DISCINT value, of the order of some number of minutes
- **Use CONVERT(NO)**
- **In production make sure you have considered security**

Ideal Channel Configuration

N O T E S

There is no single configuration that will suit all IBM MQ environments. Thus, the recommendations here are very generalised and may not be applicable to all environments. However, the suggestions given here will work successfully and will enable (particularly) new users of IBM MQ to get started quickly.

The MQ default values are chosen to work well in the majority of circumstances but may require modifying in some messaging scenarios. In general the defaults are set to use less system resources at the expense of the product being more responsive. For example, high values of DISCINT and HBINT, you may wish to reduce these.

Questions & Answers