#SHAREorg

SHARE

Session 13887

WebSphere MQ CHINIT Internals

Morag Hughson - hughson@uk.ibm.com

SHARE in Boston

CHINIT Internals - Agenda

N

O

T

E

S

- This presentation will explain some of the internal workings of the second address space, the CHINIT address space, used in the running of your WebSphere MQ subsystem on z/OS.
- An assumption is made that you have a working knowledge of z/OS and WebSphere MQ on z/OS for this presentation.
- We will first introduce you to the various tasks (TCBs) that run in this address space, then we will take a look at some of the system queues that are required for the CHINIT to run and then we will step through the life of a channel as it runs through these various tasks.
- The aim of this session is to give you enough understanding of the internals of the address space to help you look after your CHINIT address space and keep it running smoothly.

```
CSQX000I !MQ45 CSQXJST IBM WebSphere MQ for z/OS V7.1.0
CSQX001I !MQ45 CSQXJST Channel initiator starting
CSQY201I !MQ45 CSQXJST ARM REGISTER for element SYSMQCHMQ45 type SYSMQCH successful
CSQX030I !MQ45 CSQXJST 'GLOBAL' trace started, assigned trace number 00
+CSQX002I !MQ45 CSQXGIP Queue-sharing group is SQ23
+CSQX070I !MQ45 CSQXGIP CHINIT parameters ...
+CSQX071I !MQ45 CSQXGIP CHIADAPS=8, CHIDISPS=5, LSTRTMR=60
+CSQX072I !MQ45 CSQXGIP MAXCHL=200, ACTCHL=200
+CSQX073I !MQ45 CSQXGIP CHLEV=DISABLED, SSLEV=DISABLED
+CSQX074I !MQ45 CSQXGIP MONCHL=HIGH, MONACLS=QMGR
+CSQX075I !MQ45 CSQXGIP ADOPTMCA=ALL, ADOPTCHK=ALL
+CSQX078I !MQ45 CSQXGIP IGQ=DISABLED, CHADEXIT=
+CSQX079I !MQ45 CSQXGIP TRAXSTR=YES, TRAXTBL=2
+CSQX080I !MQ45 CSQXGIP SSLTASKS=5, SSLRKEYC=0
+CSQX081I !MQ45 CSQXGIP SSLKEYR=MQ45RING
+CSQX082I !MQ45 CSQXGIP SSLCRLNL=
+CSQX085I !MQ45 CSQXGIP LU62CHL=200, LUGROUP= , LUNAME=IYEBZ045,  LU62ARM=
+CSQX090I !MQ45 CSQXGIP TCPCHL=200, TCPKEEP=YES, TCPNAME=TCPIP
+CSQX091I !MQ45 CSQXGIP TCPSTACK=SINGLE, IPADDRV=IPV4
+CSQX092I !MQ45 CSQXGIP OPORTMIN=0, OPORTMAX=0
+CSQX093I !MQ45 CSQXGIP DNSWLM=NO, DNSGROUP=
+CSQX094I !MQ45 CSQXGIP RCVTIME=60, RCVTTYPE=ADD, RCVTMIN=0
+CSQX011I !MQ45 CSQXGIP Client Attachment feature available
+CSQX141I !MQ45 CSQXADPI 8 adapter subtasks started, 0 failed
+CSQX151I !MQ45 CSQXSSLI 5 SSL server subtasks started, 0 failed
+CSQX410I !MQ45 CSQXREPO Repository manager started
+CSQT975I !MQ45 CSQXDPSC Distributed Pub/Sub Controller has started
+CSQX015I !MQ45 CSQXSPRI 5 dispatchers started, 0 failed
+CSQX020I !MQ45 CSQXSPRI Shared channel recovery completed
+CSQX022I !MQ45 CSQXSUPR Channel initiator initialization complete
+CSQX004I !MQ45 CSQXSPRM Channel initiator is using 11 MB of local storage, 1602 MB are free
+CSQT806I !MQ45 CSQXFCTL Queued Pub/Sub Daemon started
+CSQX251I !MQ45 CSQXSTRL Listener started, TRPTYPE=TCP INDISP=QMGR
+CSQX023I !MQ45 CSQXLSTT Listener started,
 port 1545 address *
 TRPTYPE=TCP INDISP=QMGR
+ CSQU012I  CSQUTIL Initialization command handling completed
+CSQT975I !MQ45 CSQXDPSC Distributed Pub/Sub Fan Out Task has started
+CSQT975I !MQ45 CSQXDPSC Distributed Pub/Sub Command Task has started
+CSQT975I !MQ45 CSQXDPSC Distributed Pub/Sub Publish Task has started
```

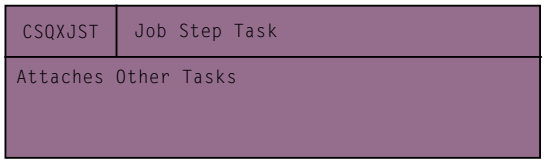
CHINIT joblog - Notes

N O T E S

- The channel initiator (CHINIT) address space is made up of a number of tasks (MVS TCBs) which are started up to serve specific purposes in the running of the CHINIT. We will start by looking at each of these tasks, it's purpose and what you need to know about it when looking after your CHINIT on z/OS.
- We will do this by walking through the messages output on the CHINIT joblog. A quick note about the messages that MQ outputs. First you will see the message number, for example CSQX000I. You can look this message number up in the Messages & Codes manual. This will give you a longer explanation of the message than just the single line. CSQ is the WebSphere MQ on z/OS message prefix. All our messages start with this. X is the component letter that represents the CHINIT, so almost all messages you will see in the CHINIT joblog are CSQX messages. Next you see the command prefix for the subsystem, in my example, my command prefix is !MQ45. Then each message will have a CSECT name which shows which part of the code is issuing this message. The various tasks we are going to look at will identify many of those CSECTs to you by the end of the presentation. After this you then see the actual message text.

The Job Step Task

- The Job Step Program
- Address space setup/cleanup
- Attaches other tasks




```
CSQX000I !MQ45 CSQXJST IBM WebSphere MQ for z/OS V7.1.0
CSQX001I !MQ45 CSQXJST Channel initiator starting
CSQY201I !MQ45 CSQXJST ARM REGISTER for element SYSMQCHMQ45 type SYSMQCH successful
CSQX030I !MQ45 CSQXJST 'GLOBAL' trace started, assigned trace number 00
```

```
//          PROC
//PROCSTEP  EXEC PGM=CSQXJST,REGION=0M
//STEPLIB   DD   DSN=MQM.SCSQANLE,DISP=SHR
//          DD   DSN=MQM.SCSQAUTH,DISP=SHR
//          DD   DSN=MQM.SCSQMVR1,DISP=SHR
```


The Job Step Task - Notes

N

O

T

E

S

- This is the task that is started when the CHINIT address space starts up. It is the PGM on the JOB STEP of the JCL for your CHINIT started task, hence it is called the Job Step Task. It is responsible for setting up what could be thought of as the frame work for the CHINIT address space including attaching a number of the other tasks that we will look at and acquiring storage for items such as the channel status table – which we will look at later.
- You will see it's prefix on some of the early messages that are output on the CHINIT joblog.
- It is also responsible for tidying up the CHINIT address space at shutdown. It detaches all the other tasks and frees off storage that was acquired.

CHINIT Parameters Usage

- Shown in the CSQXGIP messages
- Pre-V6 -> CSQ6CHIP load module
- V6 -> ALTER QMGR command
  - Allows changes to take effect immediately – no CHINIT recycle
- CSQUTIL XPARM for conversion
  - Produces an ALTER QMGR command
- DISPLAY QMGR CHINIT – group of CHINIT parameters

CHINIT Parameters Usage - Notes

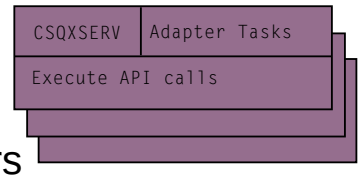
N O T E S

- The Job Step Task also gets hold of the various CHINIT parameters. These are displayed as the set of CSQXGIP messages.
- Prior to V6, these parameters were provided in a load module which was passed into the START CHINIT command (either by name or by default as CSQXPARM). In order to change any of these parameters, a macro CSQ6CHIP had to be used to rebuild the load module and the CHINIT address space had to be recycled.
- In V6 this changed. All the parameters that were provided in the CHINIT start up PARM module are now provided as attributes on the QMGR object. Thus they can be changed using ALTER QMGR and many of them can also take effect immediately. For migration purposes (for those of you still on V5.3.1) there is a CSQUTIL command, XPARM, which will build the appropriate ALTER QMGR command from an “old” parameter module.
- The command DISPLAY QMGR CHINIT will show the list of CHINIT related queue manager attributes which includes these and others that were new in V6.
- On the next page we list all the various CHINIT attributes. Most of the attributes have the same names in the ALTER QMGR command as they did in the CSQ6CHIP macro. The table shows the old attribute names in parenthesis next to the new ones for those that are different.

Queue Manager Attribute	Dynamic?	Description
CHIADAPS (ADAPS)	X	Controls the number of these tasks to start (see later)
CHIDISPS (DISPS)	X	
SSLTASKS	X	SSL configuration, use REFRESH SECURITY TYPE(SSL) to make changes take effect.
SSLRKEYC	X	
SSLKEYR	X	
SSLCRLNL	X	
RCVTTYPE + RCVTIME (RCVTIME)	√	Configures how long a channel will wait for expected data from its partner before ending.
RCVTMIN	√	
ADOPTMCA	√	Controls whether/how channels are adopted when they get orphaned by a network connection failure
ADOPTCHK	√	
MAXCHL (CURRCHL)	X	Can be reduced dynamically, but not increased. Controls size of channel status table (see later).
ACTCHL	√	Restrict number of active channels – semi-dynamic
CHISERVP (SERVICE)		For use by IBM Service
LSTRTMR	√	Interval that a listener will retry if it fails
DNSWLM	√	Controls whether and with what name the QSG group TCP listener resisters with WLM
DNSGROUP	√	
TCPCHL	√	Maximum number of TCP/IP channels – semi-dynamic
TCPKEEP	√	Use TCP KeepAlive
TCPNAME	X	Whether to use a single stack and its name, or multiple stacks and the name of the default.
TCPSTACK	X	
IPADDRV	√	Allows resolution of ambiguous IPv4 and IPv6 hostnames
OPORTMIN	√	Restriction of ports used through the firewall – less granular control than LOCLADDR on channel
OPORTMAX	√	
LU62CHL	√	Maximum number of LU6.2 channels – semi-dynamic
LUNAME	X	LU names used for outbound comms, the queue manager and group listeners
LUGROUP	√	
LU62ARM	X	APPC configuration when restarted by ARM
CHLEV	√	Controls the products of Channel and SSL event messages
SSLEV	√	
MONCHL	√	Controls collection of monitoring information for channels and auto-defined cluster channels
MONACLS	√	
IGQ	√	Controls the use of the Intra-Group Queuing Agent
CHADEXIT	√	Name of the channel auto-definition exit (Clustering)
TRAXSTR	X	Whether to automatically start trace and the size of the trace table. Use START TRACE or STOP TRACE to change
TRAXTBL	X	

Adapter Tasks

- Attached by the Job Step Task
- Issue MQ API calls on behalf of channels
  - And any other threads that run on the dispatchers
- Allows dispatcher tasks to do other work
- Set number of adapters using
  ALTER QMGR CHIADAPS(integer)
- Default is 8
- For heavy persistent workloads we recommend 30




```
+CSQX141I !MQ45 CSQXADPI 8 adapter subtasks started, 0 failed
```


Adapter tasks - Notes

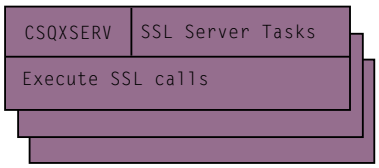
N O T E S

- Adapter subtasks are attached by the Job Step task. Their prime purpose is to service MQ API requests on behalf of channels. They are TCBs within the CHINIT address space which run the connection to MQ. Whenever a channel needs to do an MQ API call, it passes control over to an adapter task to make the call. The channel is suspended on the dispatcher until the result of the API call returns from the queue manager and the channel is resumed.
- There are several adapter tasks in the CHINIT address. You configure the number using ALTER QMGR CHIADAPS(integer).
- Each MQI call to the queue manager is independent of any other and can be made on any adapter TCB. Calls using persistent messages can take much longer than those for nonpersistent because of log I/O. Thus a channel initiator processing a large number of persistent messages across many channels may need more than the default 8 adapter TCBs for optimum performance. This is particularly so where achieved batchsize is small, because end of batch processing also requires log I/O; and where client channels are used.
- We recommend CHIADAPS(30) for such very heavy persistent message workloads. Using more than this is unlikely to give any significant extra benefit. We have seen no significant disadvantage in having CHIADAPS(30) where this is more adapter TCBs than necessary.

Complete your sessions evaluation online at SHARE.org/BostonEval

SSL Server Tasks

- Attached by the Job Step Task
- Issue SSL calls on behalf of channels
  - SSL handshakes
  - Encryption / Decryption
- Allows dispatcher tasks to do other work
- Set number of SSL tasks using ALTER QMGR SSLTASKS(integer)
- Must also specify a keyring name in SSLKEYR
- Not mandatory for CHINIT operation




```
+CSQX151I !MQ45 CSQXSSLI 5 SSL server subtasks started, 0 failed
```


SSL Server tasks - Notes

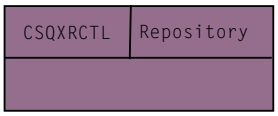
N O T E S

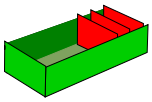
- SSL Server subtasks are attached by the Job Step task. Their prime purpose is to service SSL calls on behalf of channels. In the same way that we described in the foil about Adapter tasks, the channel is suspended on the dispatcher until the call returns.
- The calls made on these servers are those to invoke an SSL handshake call via System SSL services, and those to encrypt or decrypt a buffer of data, again via System SSL services.
- Calls made on the SSL Servers operate in a non-blocking manner as of V6. An SSL Handshake is a sequence of sends and receives and the non-blocking mechanism allows the SSL Server task to be freed up to process another SSL call while data is outstanding. At V5.3/V5.3.1 this was not the case (System SSL did not have the service yet) and SSL Channels would be serialized on these tasks. Once channels have started up, that is the SSL Handshake is done, there is no blocking concern at either version.
- 5 SSL Server tasks will be enough for several thousand channels, with the exception of the scenario where many channels have to start-up at the same time on a V5.3/V5.3.1 system. In this case, you should consider having more of these tasks. The other situation where more of these tasks are required is when Certificate Revocation Lists (CRLs) are used. These are used as part of the SSL Handshake and the LDAP server nominated is contacted in a synchronous manner so the channel start will block while this part of the handshake is done.
- If SSL Server subtasks fail to start, for example if the key-ring specified does not exist, the CHINIT address space will continue to run – SSL channels will not be available however.

Complete your sessions evaluation online at SHARE.org/BostonEval

Repository Manager Task

- Attached by the Job Step Task
- Processes cluster commands
- MQGET-wait from SYSTEM.CLUSTER.COMMAND.QUEUE
- Uses own connection to the queue manager
- Not mandatory for CHINIT operation





SYSTEM.CLUSTER.COMMAND.QUEUE

```
+CSQX410I !MQ45 CSQXREPO Repository manager started
```


Repository Manager Task

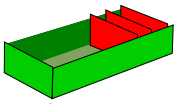
N O T E S

- This task is also attached by the Job Step Task. The Repository Task is responsible for processing clustering commands. It sits in an MQGET-wait on the SYSTEM.CLUSTER.COMMAND.QUEUE using its own connection to the queue manager – that is it does not go via the adapter tasks.
- If the repository task cannot MQOPEN the various queues used (see later) it will shutdown but the CHINIT address space will continue to run – clustering operations will not be available however.
- More details about Clustering and the queues used by the repository manager are covered in the Clustering session.

The Supervisor

- Attached by the Job Step Task
- Processes channel and listener commands, and trigger messages
- MQGET-wait from SYSTEM.CHANNEL.INITQ
- Uses own connection to the queue manager
- Invokes channel retry processing
- Mandatory task for operation of the CHINIT address space.

CSQXSUPR	Supervisor
Attach - Dispatchers - Listeners	



SYSTEM.CHANNEL.INITQ

Channel name	XmitQ	Partner	Status
QM1.TO.QM2	**QM2**	**QM2.IBM.COM**	**Retrying**
QM2.TO.QM1		**QM2.IBM.COM**	**Running**

The Supervisor - Notes

N

O

T

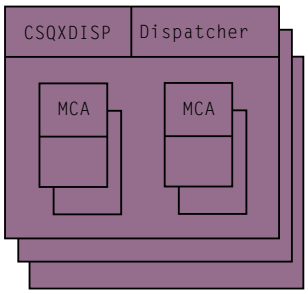
E

S

- We don't have a specific message to tell us the supervisor task has been attached, simply because, if it fails, the CHINIT will immediately end, so it's rather superfluous. We can see evidence that it has been started because we see what it does next with the next few messages in the log being identified as the supervisor.
- The supervisor task is the main control point of the CHINIT. It is attached by the Job Step task and is responsible for attaching the Dispatcher tasks. It reads from the SYSTEM.CHANNEL.INITQ, so if the CHINIT address space is running but this queue does not have IPPROCS(1) something is not working correctly.
- All channel commands are routed to this queue from the command server for the CHINIT to act upon, and all trigger messages for channels also go on this queue. We will look in more detail at these activities later. Listener commands are also routed here so the Supervisor is also responsible for attaching the Listener tasks.
- The STOP CHINIT command also goes on this queue and the supervisor kicks off shut down processing. The same processing is also invoked if you GET(DISABLE) the SYSTEM.CHANNEL.INITQ.
- This task also periodically scans the status table to find channels in RETRYING status that need to be started. We will also look at this later.

Dispatcher Tasks

- Attached by the Supervisor Task
- Worker tasks for channel code to run on
  - And other threads too
  - Co-operatively sub-dispatched
  - Many channels run on one dispatcher
  - Bear this in mind when writing exits
- Set number of dispatchers using ALTER QMGR CHIDISPS(integer)
- Loading algorithm uses MAXCHL value
  - With very small numbers of channels loading can be skewed




```
+CSQX015I !MQ45 CSQXSPRI 5 dispatchers started, 0 failed
```


Dispatcher Tasks - Notes

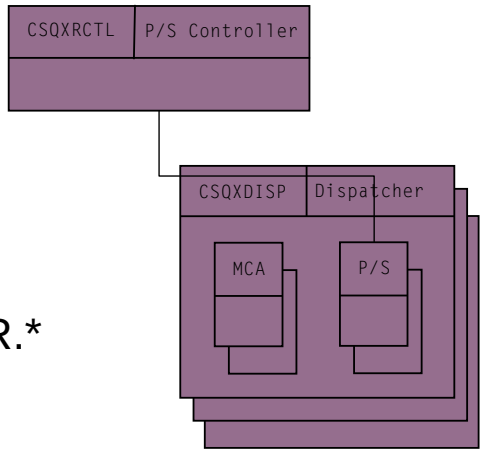
N O T E S

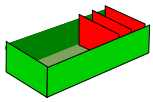
- Dispatchers are worker tasks where all the main channel activity is run. They are attached by the Supervisor task. There are several dispatcher tasks in the CHINIT address space. You configure the number using ALTER QMGR CHIDISPS (prior to version V6.0 it was a parameter on the CSQ6CHIP macro). We recommend CHIDISPS(20) for systems with more than 100 channels. We have seen no significant disadvantage in having CHIDISPS(20) where this is more dispatcher TCBs than necessary.
- Many channels share a dispatcher task where they are co-operatively sub-dispatched. That means when a channel reaches certain yield points in the code, it gives up control back to the dispatcher task, who can then schedule another channel to run. Each channel is associated with a particular dispatcher TCB at channel start and remains associated with that TCB until the channel stops. MAXCHL is used to spread channels across the available dispatcher TCBs. The first (MIN((MAXCHL / CHIDISPS) , 10) channels to start are associated with the first dispatcher TCB and so on until all dispatcher TCBs are in use. The effect of this for small numbers of channels and a large MAXCHL is that channels are NOT evenly distributed across dispatchers. We recommend setting MAXCHL to the number of channels actually to be used where this is a small fixed number.
- It is worth remembering that a channel does not have the task to itself, if for example you write channel exits. When writing channel exits you should never do anything which makes the task wait, for example going to sleep, since you will make all the other channels on that dispatcher task wait as well. Instead we recommend if you need to do this, to attach another task in your exit and use the MQXWAIT API call to wait for the completion of that task. This means that the dispatcher can allow other channels to run in the mean time.

Complete your sessions evaluation online at SHARE.org/BostonEval

Distributed Pub/Sub Tasks

- Controller attached by the Job Step Task
- Controller manages the remaining threads
  - Run on dispatchers – like channels
- Three main threads process distributed publish/subscribe operations
- MQGET-wait from SYSTEM.INTER.QMGR.* queues
  - One for each thread
- Not mandatory for CHINIT operation
  - Controlled by PSMODE





SYSTEM.INTER.QMGR.*

```
+CSQT975I !MQ45 CSQXDPSC Distributed Pub/Sub Controller has started
+CSQT975I !MQ45 CSQXDPSC Distributed Pub/Sub Fan Out Task has started
+CSQT975I !MQ45 CSQXDPSC Distributed Pub/Sub Command Task has started
+CSQT975I !MQ45 CSQXDPSC Distributed Pub/Sub Publish Task has started
```

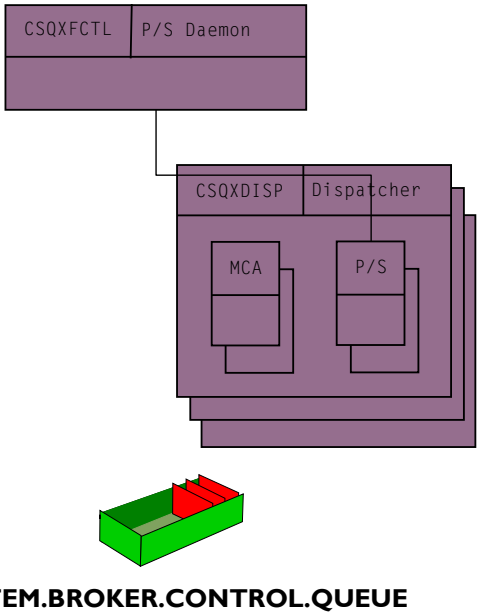

Distributed Pub/Sub Tasks - Notes

N O T E S

- This Distributed Pub/Sub Controller task is also attached by the Job Step Task. It is responsible for managing the three Distributed Pub/Sub worker threads, the Fan-Out Task, Command Task and Publish Task. These three worker threads are started as threads on the dispatchers by the Controller Task, once the Dispatchers are ready to schedule work.
- These threads are controlled by the PSMODE switch on the queue manager. If it is PSMODE(DISABLED) then these worker threads will not be started.
- The Distributed Pub/Sub Fan Out Task waits for messages on the SYSTEM.INTER.QMGR.FANREQ queue.
- The Distributed Pub/Sub Command Task waits for messages on the SYSTEM.INTER.QMGR.CONTROL queue.
- The Distributed Pub/Sub Publish Task waits for messages on the SYSTEM.INTER.QMGR.PUBS queue.

Queued Pub/Sub Daemon Task

- Attached by the Supervisor Task
- Daemon starts stream threads
  - Streams listed in SYSTEM.QPUBSUB.QUEUE.NAMELIST
  - Run on dispatchers – like channels
- Daemon MQGET-wait from SYSTEM.BROKER.CONTROL.QUEUE
- Streams MQGET-wait from stream queue
  - Named in above namelist
- Not mandatory for CHINIT operation
  - Controlled by PSMODE(COMPAT)



SYSTEM.BROKER.CONTROL.QUEUE

```
+CSQT806I !MQ45 CSQXFCTL Queued Pub/Sub Daemon started
```

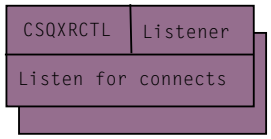

Queued Pub/Sub Daemon Task - Notes

N O T E S

- The Queued Pub/Sub Daemon task is also attached by the Supervisor Task. This task is responsible for processing the messages that arrive on the SYSTEM.BROKER.CONTROL.QUEUE (V6 compatible Pub/Sub messages in RFH1 or RFH2 format). This task will also start any streams that are required as listed in the SYSTEM.QPUBSUB.QUEUE.NAMELIST.
- This task is also controlled by the PSMODE switch on the queue manager. If it is PSMODE(COMPAT) then this task is not started because that means we are running in Compatibility mode with Message Broker which still owns the queue and is processing it, so we don't need to. If we are set to PSMODE(ENABLED) then this task will be started.

Listener Tasks

- Attached by the Supervisor Task
- One Task per TRPTYPE(TCP | LU62) and per INDISP(QMGR | GROUP)
  - Maximum four possible tasks
- Multiple address/port combinations on the TCP listener tasks
- LSTRTMR configures the retry interval after an APPC or TCP/IP failure



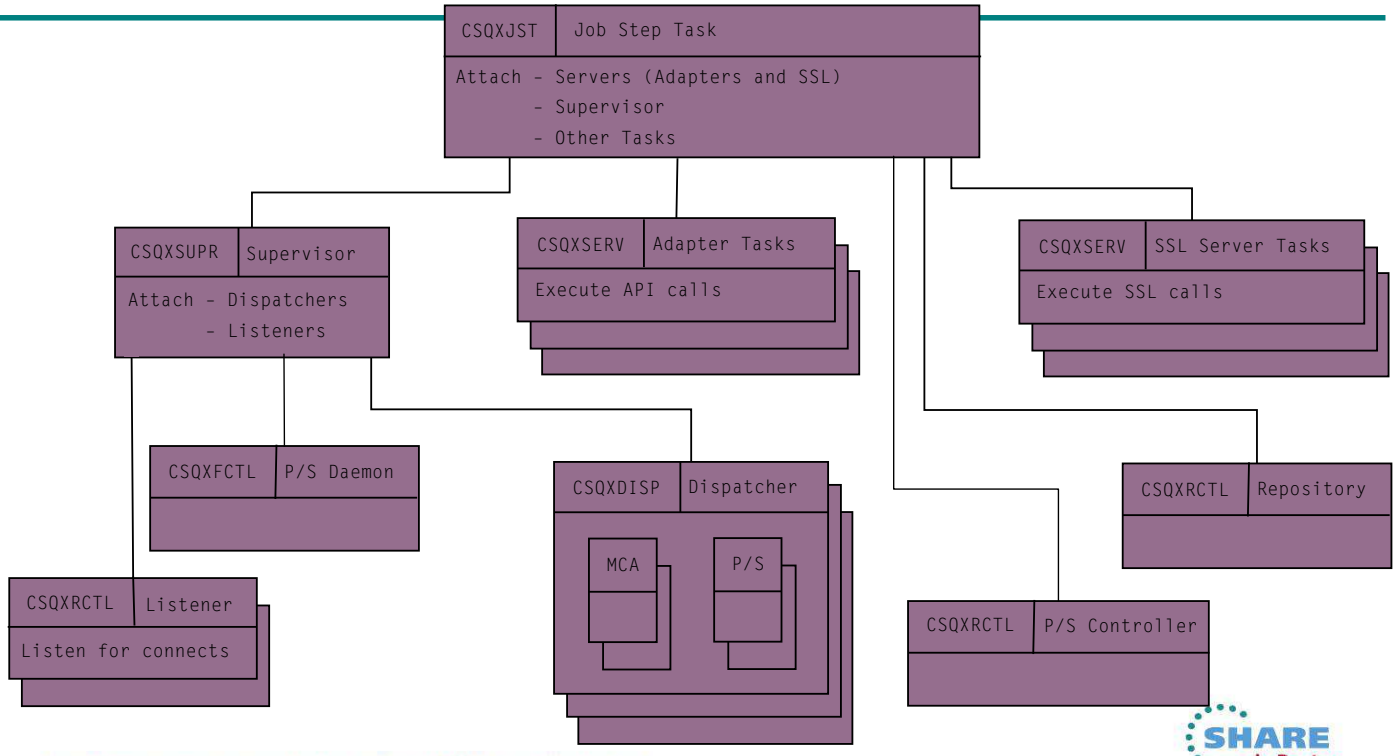

```
+CSQX251I !MQ45 CSQXSTRL Listener started, TRPTYPE=TCP INDISP=QMGR
+CSQX023I !MQ45 CSQXLSTT Listener started,
 port 1545 address *,
 TRPTYPE=TCP INDISP=QMGR
+CSQX251I !MQ45 CSQXSTRL Listener started, TRPTYPE=TCP INDISP=GROUP
+CSQX023I !MQ45 CSQXLSTT Listener started,
 port 2545 address *,
 TRPTYPE=TCP INDISP=GROUP
```


Listener Tasks - Notes

N O T E S

- These tasks are attached by the Supervisor task as a result of a START LISTENER command. There can be up to four tasks started for listener processing.
  - Queue Manager disposition TCP/IP listener
  - Group disposition TCP/IP listener
  - Queue Manager disposition LU6.2 listener
  - Group disposition LU6.2 listener
- The TCP/IP listeners can have multiple address and port combinations, all of which are handled by the single task. When the last address/port which is being listened on is stopped, the task will be ended again.
- If there is an error in the APPC or TCP/IP system, the listener tasks will retry after the number of seconds specified in the LSTRTMR attribute.

CHINIT Address space structure



CHINIT Address space structure

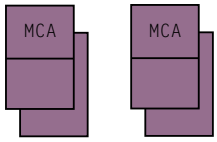
N O T E S

- We have built this picture up through the previous pages. These are all the major tasks that you will see in the CHINIT address space. There are some other ones that are not mentioned here whose purposes are very specific and the detail of which is beyond the scope of this presentation.
- The names shown in the title bar of each task on this diagram are the load modules name – the entry points that you would see for each of the TCBs if you took a dump of the address space. These sometimes match CSECT inserts in messages as we have seen and sometimes do not. A common technique is to take 7 of the 8 chars, e.g. CSQXSPR instead of CSQXSUPR (the supervisor) and add I (initialization), M (main line), T (termination) to the message insert. This gives a clue about exactly the point in processing of that task that caused an error message and is worth noting.

Complete your sessions evaluation online at SHARE.org/BostonEval

Channel capacity

- Capacity of CHINIT – number of channels
- Maximum 2GB Address Space
  - More likely no more that 1.6GB EPVT




```
//          PROC
//PROCSTEP  EXEC PGM=CSQXJST,REGION=0M
//STEPLIB   DD   DSN=MQM.SCSQANLE,DISP=SHR
//          DD   DSN=MQM.SCSQAUTH,DISP=SHR
//          DD   DSN=MQM.SCSQMVR1,DISP=SHR
```

- Calculation
  - EPVTsize(KB) /
    (max-message-size(KB) + EPVT-required-per-channel(KB))
- Upper limit around
  - 9000 non-SSL channels
  - 7500 SSL channels
- Remember message size

EPVT required per channel	
Non-SSL	SSL
140KB	170KB

```
+CSQX004I !MQ45 CSQXSPRM Channel initiator is using 11 MB of local
storage, 1602 MB are free
```

MP16 SHARE in Boston

Channel capacity - Notes

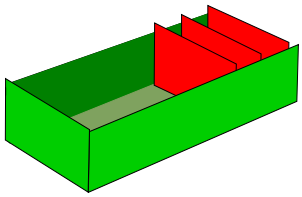
N O T E S

- The main consideration for the capacity of your CHINIT – that is how many channels it can run – is storage usage. The CHINIT address space is a 31-bit address space which means it has a theoretical limit of 2GB of addressable storage. However on most systems the extended private region (EPVT) is unlikely to exceed 1.6 GB.
- Every non-SSL channel uses about 140 KB and every SSL channel about 170KB of extended private region in the channel initiator address space. Storage is increased if messages larger than 32 KB are being transmitted. This increased storage is freed when a sending channel or client channel requires less than half the current buffer size for 10 consecutive sends or a heartbeat is sent or received.
- Message size makes big difference to this calculation. If all channels are to move 100 MB messages then only 15 channels are possible!
- The storage used by an server-connection channel is dependant on how the MQ Client application was written. For example, if the client application requests an MQGET with a 100MB buffer, then a 100MB buffer will be obtained by the server-connection channel which is using storage in your CHINIT address space. More modern versions of the MQ Client code can reduce this however. A V7+ client, will convert the application's MQGET call into an MQCB call an allow the queue manager to manage the size of the message buffer instead of automatically getting exactly the size requested on the MQGET. For this behaviour the channel must not be restricted to V6 mode, i.e. it must be using a non-zero value in SHARECNV.
- Message CSQX004I is issued periodically, once per hour, or when the memory usage changes (up or down) by more than 2%. This message was added in V7.
- Support Pac MP16 – Capacity Planning and Tuning for WebSphere MQ for z/OS contains details about this and other capacity issues for your queue manager in general.

SHARE in Boston

System Queues

- CSQ4INSX, CSQ4INSR and CSQ4INSS
- Channel queues
  - SYSTEM.CHANNEL.INITQ
  - SYSTEM.CHANNEL.SYNCQ
    - DISPLAY CHSTATUS(*) SAVED
- Clustering queues
  - SYSTEM.CLUSTER.COMMAND.QUEUE
  - SYSTEM.CLUSTER.REPOSITORY.QUEUE
  - SYSTEM.CLUSTER.TRANSMIT.QUEUE
- Queue-sharing group queues
  - SYSTEM.QSG.CHANNEL.SYNCQ
  - SYSTEM.QSG.TRANSMIT.QUEUE



- Distributed Pub/Sub queues
  - SYSTEM.INTER.QMGR.FANREQ
  - SYSTEM.INTER.QMGR.CONTROL
  - SYSTEM.INTER.QMGR.PUBS
- Queued Pub/Sub queue
  - SYSTEM.BROKER.CONTROL. QUEUE

System Queues - Notes

N

O

T

E

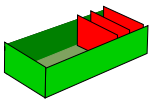
S

- The CHINIT address space uses several system queues to help in its operation. We have mentioned a couple of them already. Ensure you have the correct attributes on the definitions for these queues. The channel and cluster queue definitions are provided in sample CSQ4INSX and the additional queues required if you are using queue-sharing groups are provided in sample CSQ4INSS. The Queued Pub/Sub queues are provided in sample CSQ4INSR.
- The SYSTEM.CHANNEL.INITQ is read by the Supervisor for channel commands and trigger messages.
- The SYSTEM.CHANNEL.SYNCQ and the SYSTEM.QSG.CHANNEL.SYNCQ are used for storing channel state, for private and shared channels respectively. You can see the information stored on these queues using the DISPLAY CHSTATUS(*) SAVED command.
- The clustering queues are covered in the clustering session.
- The SYSTEM.QSG.TRANSMIT.QUEUE is read by the Intra-Group Queuing (IGQ) agent which runs in the queue manager (MSTR) address space and can be used for moving messages between members of the Queue-Sharing Group (QSG) without using channels. This is beyond the scope of this presentation.

Complete your sessions evaluation online at SHARE.org/BostonEval

In-memory Channel Status

- State about channels in the system
- In-memory for fast access and update
- Hardened to SYNCQ to be used at CHINIT restart
  - Restore STOPPED and RETRYING channels
  - Resynchronise with partner channel when INDOUBT about last batch
- MAXCHL controls the size of the status table



Channel name	XmitQ	Partner	Status
QM1.TO.QM2	QM2	QM2.IBM.COM	Retrying
QM2.TO.QM1		QM2.IBM.COM	Running
QM1.TO.QM3	QM3	QM3.IBM.COM	Stopped

In-memory Channel Status - Notes

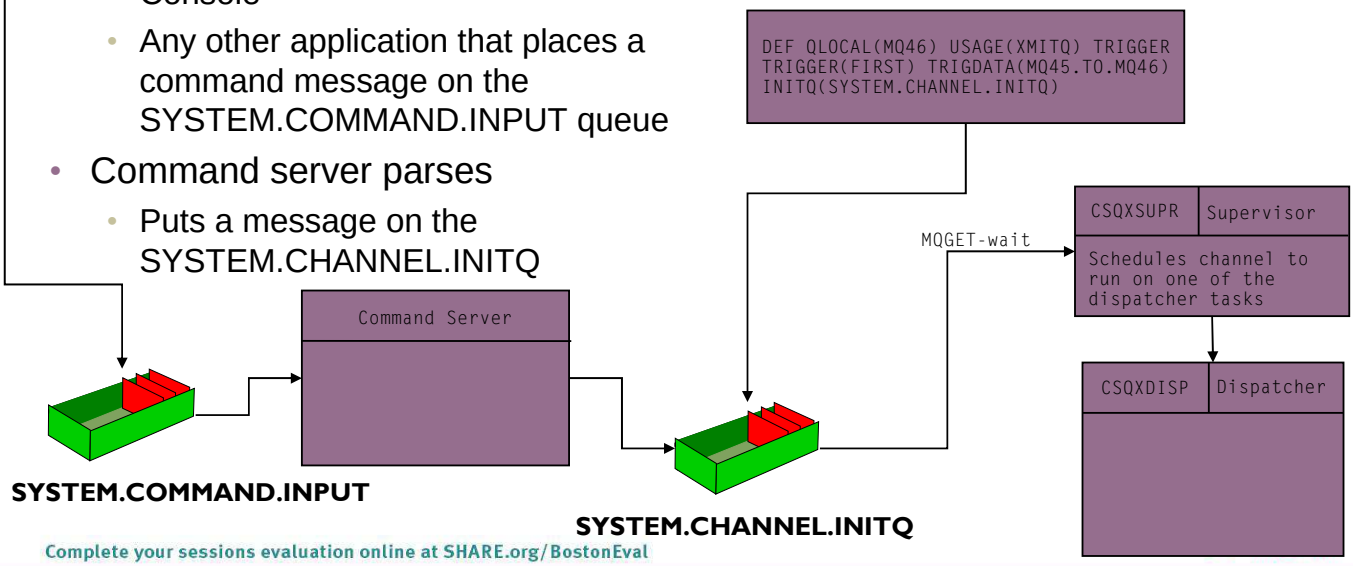
N O T E S

- The CHINIT address space keeps track of how channels are running in the Channel Status Table. There is a version of it in-memory as well as the hardened state version held on the SYSTEM.CHANNEL.SYNCQ and the SYSTEM.QSG.CHANNEL.SYNCQ, for private and shared channels respectively.
- You can see the information stored in the in-memory status table using the DISPLAY CHSTATUS(*) CURRENT command. This shows information such as the number of messages the channel has delivered; information about the batch that is in progress; its start date/time and how many retry attempts it has left if its STATUS is RETRYING.
- The Channel Status table in kept in memory for fast access and update – each time data is sent down the channel, various counters such as number of bytes and number of messages are updated. It is hardened to the appropriate SYNCQ at the end of the batch. The contents of this queue are used to rebuild the state of the channels after a CHINIT restart. Channels that you have issued a STOP CHANNEL command against, remain STOPPED; channels that were retrying know where to continue their retry from. Channels that ended INDOUBT about the state of their batch of messages have a record of how far they got and what to synchronise with their partner when connection is re-established.
- The size of the status table is determined by the value of MAXCHL – your must set this value large enough to include all channels that must have state recorded – not just those that are Running. Retrying and Stopped channels need a slot in the status table to record their state too

Complete your sessions evaluation online at SHARE.org/BostonEval

A Day in the Life of … a Channel
Starting a Sender Channel

- START CHANNEL command
  - Ops & Control panels
  - MQ Explorer
  - Console
  - Any other application that places a command message on the SYSTEM.COMMAND.INPUT queue
- Command server parses
  - Puts a message on the SYSTEM.CHANNEL.INITQ
- Triggering
  - Queue manager generates a trigger message and places it on the specified INITQ



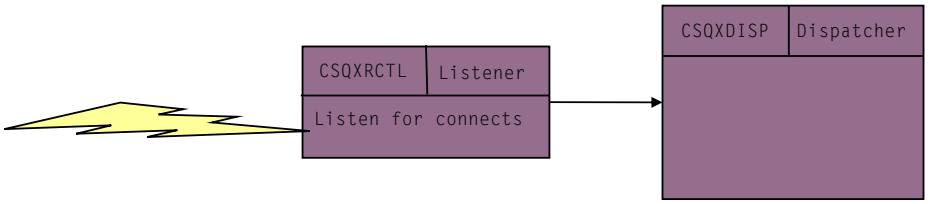
Starting a Sender Channel - Notes

N O T E S

- To start a sender channel you can issue a START CHANNEL command from the Ops and Control panels, via the MQ Explorer GUI (which can manage a z/OS queue manager), the console or any other tool that places a command message on the SYSTEM.COMMAND.INPUT queue. When you issue a start command the command is processed by the command server in the Queue Manager address space, checked for syntax errors, and results in a start channel message being put on the SYSTEM.CHANNEL.INITQ which the supervisor task reads and acts upon.
- Alternatively you can use triggering. When you set up triggering on transmission queues, the INITQ you nominate is the SYSTEM.CHANNEL.INITQ. This is because the CHINIT is the trigger monitor for channels. When a message arrives on the transmission queue, the queue manager follows all the usual triggering rules and places a trigger message on the SYSTEM.CHANNEL.INITQ which the supervisor task reads and acts upon shortly after it has been put (it sits in an MQGET-wait).
- The supervisor task schedules the sender channel to run on a dispatcher task. The supervisor then returns to monitoring the SYSTEM.CHANNEL.INITQ for more command requests. When the dispatcher next gets control it will start running the sender channel.

A Day in the Life of … a Channel
Starting a Receiver Channel

- Listener
  - Awaiting connection
- Connection arrives
- Listener task schedules a channel to run on a dispatcher task



Starting a Receiver Channel - Notes

N O T E S

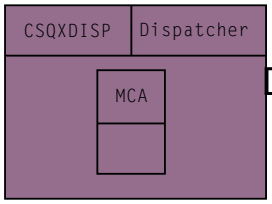
- In order to start a receiver channel, you must have a listener listening on a TCP/IP port or an APPC LU name. When a network request arrives at the listener, it schedules the receiver channel to run on a dispatcher task. The listener then returns to waiting on the network for the next connection request. When the dispatcher next gets control it will start running the receiver channel.

A Day in the Life of … a Channel
Normal Channel Operation

MQGET via Adapter Tasks

Encrypt via SSL Server Task

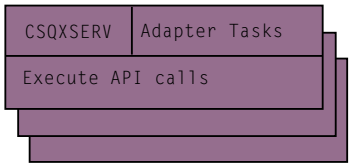
Send to the Comms subsystem

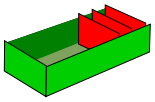


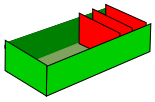
Receive from the Comms subsystem

Decrypt via SSL Server Task

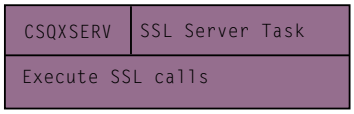
MQPUT via Adapter Tasks

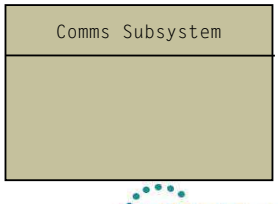






Transmission Queue **Target Queue**



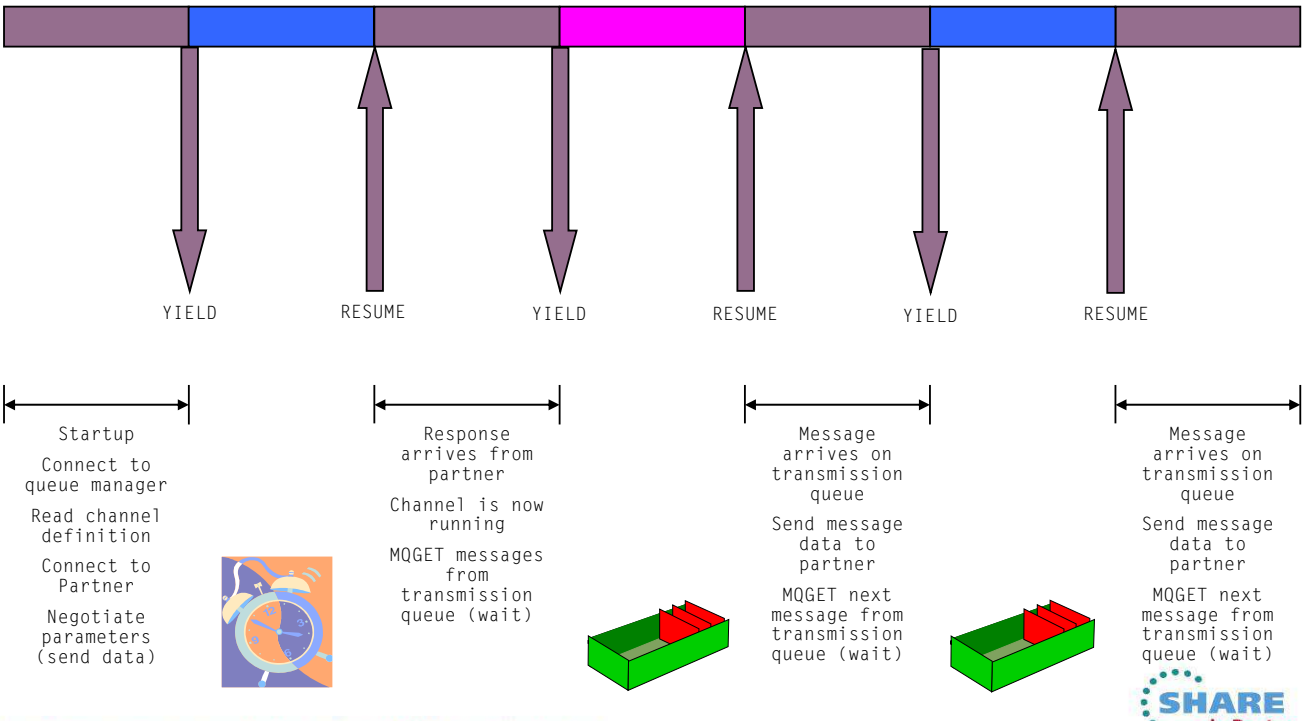


Normal Channel Operation - Notes

N O T E S

- Once a channel is running on a dispatcher task, its normal operation will include interaction with the queue manager via the adapter tasks (to MQGET from or MQPUT to queues), interaction with a TCP/IP or APPC subsystem to receive or send data on the network, and if SSL encryption is required, interaction with the SSL server subtasks to have the encrypt and decrypt processing done.
- An instance of a channel will always run on the same dispatcher while it remains in RUNNING state. If it goes into RETRYING state and is subsequently successfully started it may be on a different dispatcher. When a channel requests an API be issued on its behalf by an adapter, it will use the first one that is free – it has no tie to any adapter. When a channel requests the use of System SSL facilities via the SSL Server subtask, it is tied to the first one it chose – this affinity remains for the duration of running channel instance.
- Each time a channel makes use of one of these other tasks in the CHINIT address space it yields control of the dispatcher allowing another channel to run. We will look at this behaviour next.

A Day in the Life of … a Channel
Cooperating with other channels



Cooperating Channels - Notes

N O T E S

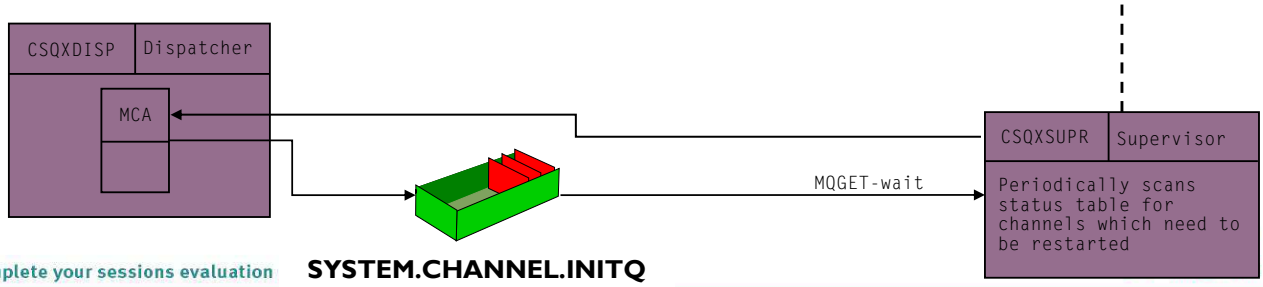
- The channel reads its definition from the Queue Manager and determines its vital parameters. A sender channel needs to know some vital things about itself. Name (used to get all information about itself from the queue manager), Protocol (this is either TCP/IP or APPC), Partner's location (for TCP/IP this is a network address and port; for APPC this is an LU name), The transmission queue that it is going to serve This is where it is going to get its messages from. The destination queue is contained in the message.
- Once it has determined that everything is OK on the local side, it will try to start a conversation with its partner receiver channel on the remote system. Once the conversation has been established, the sender and receiver channels negotiate various session parameters such as batch size and maximum message size.
- After successful negotiation, the sender channel waits for the Queue Manager to inform it when messages arrive on the transmission queue it is servicing. On the remote system, the receiver channel waits for communication from the sender channel.
- Many channels run on each dispatcher task (we discussed the loading algorithm and the recommended number of tasks needed earlier). Channels run in a non pre-emptive, cooperative, sub-dispatching task. This means that a channel will yield control back to the dispatcher task when they need to wait for something to happen. The dispatcher task can then allow another channel to run on the task for a little while – until it needs to wait.
- Channels actually spend a lot of time waiting! Sender channels wait for messages to arrive on transmission queues and receiver channels wait for data to arrive on the network.
- This shows why you shouldn't sleep or wait the whole task in a channel exit, because while in the exit the channel cannot yield control, unless you do as recommended and use the MQXWAIT call in your exit.

A Day in the Life of … a Channel
Channel Retry

- Channel detects an error
- Channel stops running
  - Completion message written to SYSTEM.CHANNEL.INITQ
  - Status marked as RETRYING
- Supervisor monitors Status Table
  - Restarts channel after retry interval has expired

```
CSQX208E !MQ45 CSQXRCTL Error receiving data,
channel MQ45.TO.MQ46,
connection localhost (127.0.0.1)
(queue manager MQ46)
TRPTYPE=TCP RC=00000461 reason=76650446
CSQX599E !MQ45 CSQXRCTL Channel MQ45.TO.MQ46 ended abnormally
```

Channel name	XmitQ	Partner	Status
MQ45.TO.MQ46	MQ46	MQ46.IBM.COM	Retrying
MQ46.TO.MQ45		MQ46.IBM.COM	Running



Channel Retry - Notes

N O T E S

- Channel retry is the ability of the channel initiator to help channels to overcome transient errors. There are both long and short retry times and long and short retry counts. Retry occurs when a channel encounters a transient error, such as a communications failure. It will then be retried.
- When the error occurs, a message is written to the system log, the channel's status is set to RETRY.
- The channel writes a completion message to the SYSTEM.CHANNEL.INITQ indicating failure with retry, and the channel code stops running. As the supervisor is continually monitoring the SYSTEM.CHANNEL.INITQ. It will be alerted when a channel ends, normally or abnormally. As part of its regular wake-up cycle, the Supervisor scans the channel status table and restarts any channel that is in retry state and has exceeded its retry interval since it was last started. The supervisor used the shortest time to retry for all the channels as its GET_WAIT interval to the SYSTEM.CHANNEL.INITQ.
- The channel code starts running again and may or may not succeed. Note: When the channel exceeds both its short and long retry counts, it becomes DISABLED.