



**MQQSG**  
**MQ Coupling Facility Structures and CFSTRUCTs**

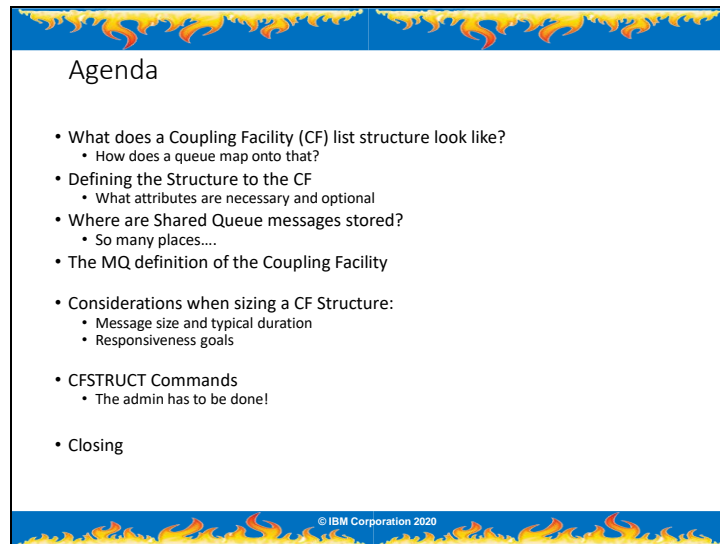
Steve Bohn – [bohns@us.ibm.com](mailto:bohns@us.ibm.com)  
Mitch Johnson – [mitchj@us.ibm.com](mailto:mitchj@us.ibm.com)  
Lyn Elkins – [elkinsc@us.ibm.com](mailto:elkinsc@us.ibm.com)



© IBM Corporation 2020



We are from IBM's Washington system center and this presentation is about the coupling facility structures – how they are defined to z/OS and to MQ.

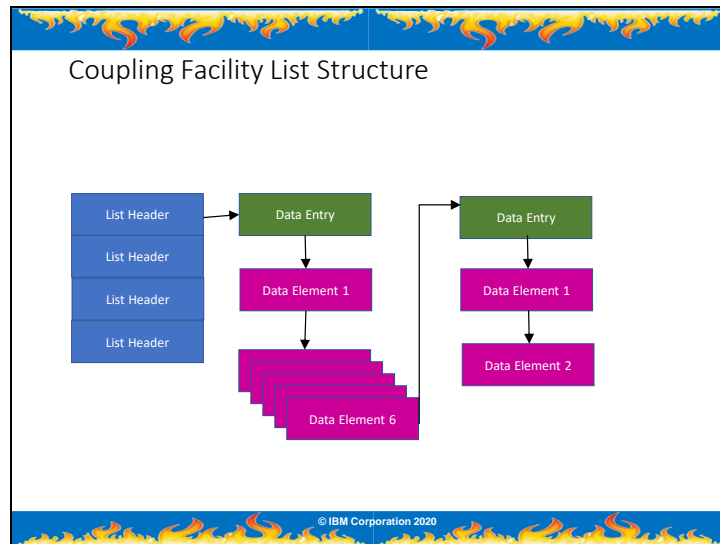


## Agenda

- What does a Coupling Facility (CF) list structure look like?
  - How does a queue map onto that?
- Defining the Structure to the CF
  - What attributes are necessary and optional
- Where are Shared Queue messages stored?
  - So many places....
- The MQ definition of the Coupling Facility
- Considerations when sizing a CF Structure:
  - Message size and typical duration
  - Responsiveness goals
- CFSTRUCT Commands
  - The admin has to be done!
- Closing

© IBM Corporation 2020

The agenda for this session.



NOTES

## Shared Message Queue Storage Using CF List Structures

What is a List structure?

- A List structure consists of a set of list headers (queues). A List header anchors the list and contains information associated with the list. The most interesting are the count of list entries (queue depth) and maximum number of list entries (maximum queue depth).
- A List or Data Entry (we will call them Data) consist of
  - .Data entry controls contain information such as an entry identifier, (maps to MsgId) and a primary and secondary key. The list can be monitored according to the contents of the primary and secondary key.
  - .Data elements contain user data which corresponds to the message data (including MQMD). The CF architecture defines a maximum number of elements associated with a data entry, and this limits the message size that may be stored in the CF.

What functions does the List structure provide?

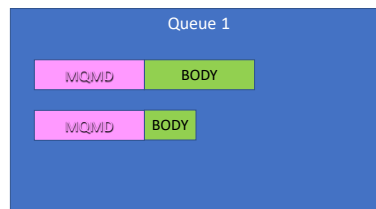
- A rich variety of access methods are provided to manipulate CF list structures.
- Basic functions and their mapping to the MQI include:
  - READ: Read a data entry; used for MQGET browse a message from a queue.
  - WRITE: Create a new data entry; MQPUT a message to a queue.
  - DELETE: Delete a data entry; Careful - MQGET is not fully deleted until committed.
  - MOVE: Move a data entry to another list; used to get a message under syncpoint, where it is invisible.
- More complex functions include
  - READ\_LIST: Read data entries with specific key; First pass in MQGET with MsgId/Correlid, list then searched.
  - READ\_LCONTROLS: Read list header controls; Used for determining queue depth.
  - MONITOR\_KEYRANGE: Notify when count of data entry with key range exists; Used for MQGET-wait, triggering.

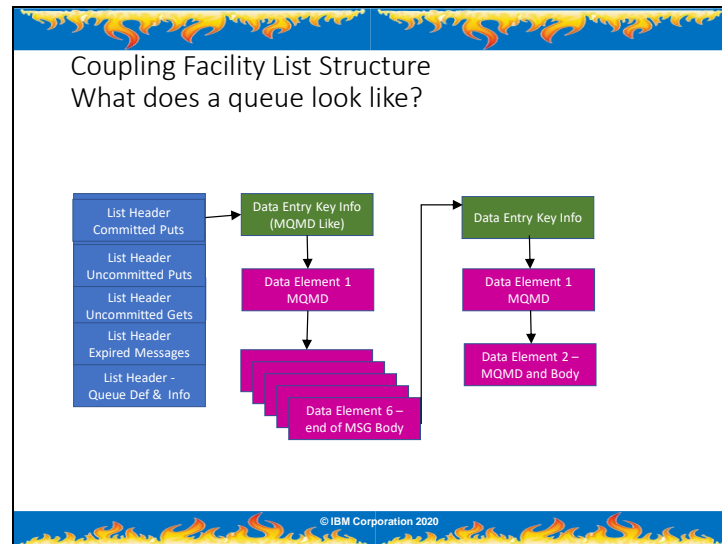
Each block represents 256 bytes of storage, so the maximum size a single message can be in the CF is 63K, 255\*256

Auxiliary storage is used for messages larger than 63K, and will be described a bit later

## The physical representation of messages

- Very simple example, a queue with 2 messages that have been put and committed.
  - The first message is fairly small, around 800 bytes, without the MQ header
  - The second message is tiny – about 100 bytes without the header





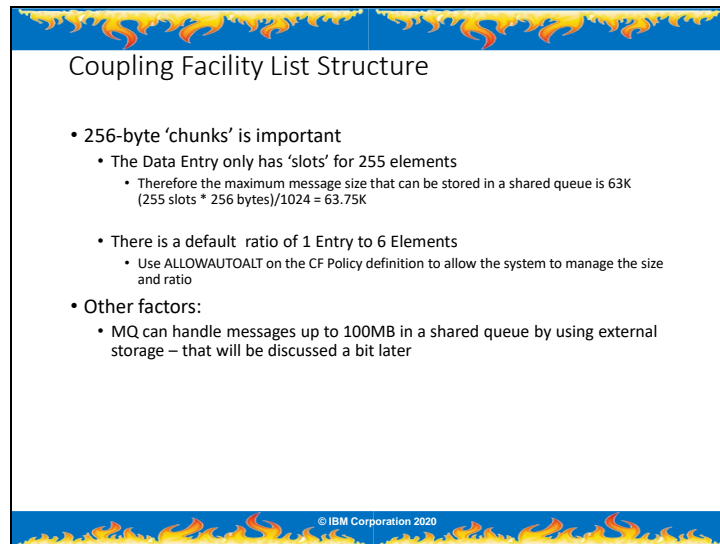
This is a visual presentation of what those messages look like when put to a shared queue. Each list header, as they're called, points to the list of messages that are in the state controlled by that header. There will be data entry headers for each queue defined on the structure. So in this representation the list header of committed puts is pointing into the data entry key information of the first message. Each entry and the associated data elements are 256 byte chunks of coupling facility memory.

The data entry contains significant information about the message, including the message key (message ID, correlation ID) if the queue is indexed. It also contains pointers to the data elements that make up the message. The elements contain the message itself, including any headers MQ has applied (MQMD and transmission headers).

The first message has the data entry, again containing information about the message, followed by the elements that make up the message including the headers. The second message is an example of the smallest messages, with one entry and two elements. For MQ V7.1 messages that were 222 bytes or less would get stored like this.

Generally speaking, using shared queues costs more in CPU than private. However the availability of messages on shared queues so far exceeds any other storage mechanism on any platform, the per message CPU costs often pale in comparison to the costs of an outage.





The slide is titled "Coupling Facility List Structure" and is framed by a decorative border with a blue background and yellow and orange flame-like patterns at the top and bottom. The content is organized into a bulleted list.

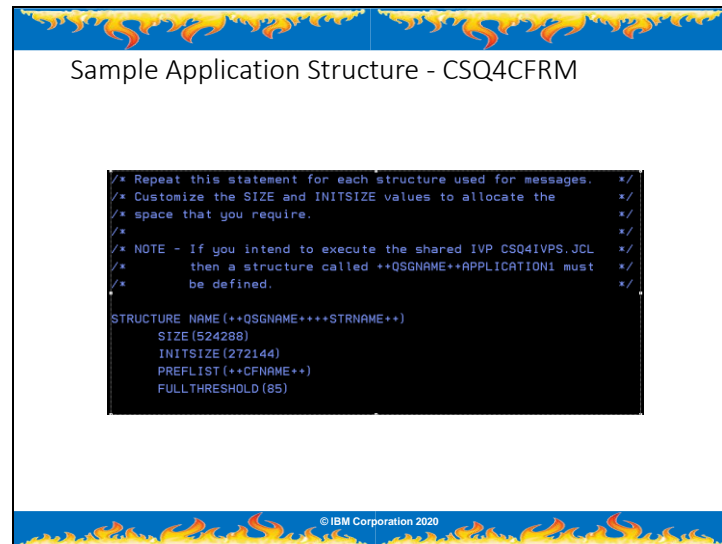
### Coupling Facility List Structure

- 256-byte 'chunks' is important
  - The Data Entry only has 'slots' for 255 elements
    - Therefore the maximum message size that can be stored in a shared queue is 63K  
(255 slots \* 256 bytes)/1024 = 63.75K
  - There is a default ratio of 1 Entry to 6 Elements
    - Use ALLOWAUTOALT on the CF Policy definition to allow the system to manage the size and ratio
- Other factors:
  - MQ can handle messages up to 100MB in a shared queue by using external storage – that will be discussed a bit later

© IBM Corporation 2020

- Calculating the maximum message size is easy, however the reason for the 63K message limitation was not clear in the first releases of the documentation
- The default ratio of 1 Entry to 6 elements can be changed, which may make the overall storage use better in the structure. It is especially helpful when the message sizes are consistent. Letting the system adjust that via the ALLOWAUTOALT attribute will be addressed in more detail a bit later.





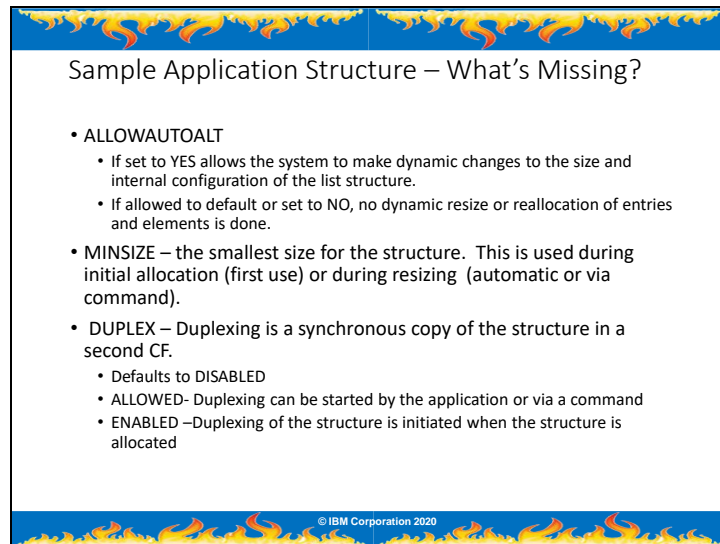
This is the sample application structure definition. It is virtually empty and missing some things that are useful in defining an application structure. I'll talk about some of those on the next slide, but for now:

- The ++QSGNAME++ is the four-character designation for the QSG. In the labs and what we use at the WSC it is QSGM
- The ++STRNAME++ is the up to eight-character name for the application structure. From a best practice perspective, this may indicate the application that uses this structure or may be more about how it is used. Our typical recommendation is to have at least 3 application structures – two for trusted applications and one for applications that may not be as stable and trusted as others. The two for trusted application may be Separated into persistent messages and nonpersistent structures. This is the CFSTRUCT, the MQ definition of the structure, set-up and administration in MQ differs for persistent messages storage. That separation is not always advisable when applications used mixed message types, as another recommendation is to have all shared queues used in a single UOW in the same structure – that reduces overhead on commits and rollbacks when only one structure is used. The best example of a single queue that hosts both persistent and nonpersistent messages is a transmission queue – those are typically not segregated.

The reason behind the 'lesser known app' recommendation is simple – just like an application using private queues can be subject to sympathy sickness when a bufferpool or pageset fills up, the CF storage is even more prone to storage constraints. Separating those applications that are new, those with no track record,

and those that have a known history of issues can preserve the reliability of the good actors in the environment.

- SIZE is the maximum size that a structure can grow to, if allowed. As there are no units given, this is in K (M,F and T are also allowed). The largest structure size is 1 terabyte (T). Please see:  
[https://www.ibm.com/support/knowledgecenter/SSLTBW\\_2.4.0/com.ibm.zos.v2r4.ief100/subcfrm.htm](https://www.ibm.com/support/knowledgecenter/SSLTBW_2.4.0/com.ibm.zos.v2r4.ief100/subcfrm.htm)
- INITSIZE is the initial amount of storage allocated in the CF when the structure is first used. It can be equal to SIZE or smaller. This attribute is optional.
- PREFLIST – A list of Coupling facilities that can host the structure. This is particularly important if the structure is recoverable and may need to be moved to a secondary location in the event of a CF outage.
- FULLTHRESHOLD – this percentage value tells the CF when to notify that a structure is becoming full and to initiate any automatic resizing that can be done. The next page will describe the automatic alteration capability and how it's specified. The default value is 80.  
For some high volume applications 80 might be too high, when the structure is filling rapidly that may not give enough time to respond before applications start experiencing PUT failures with the storage media full reason code ('2192').



### Sample Application Structure – What's Missing?

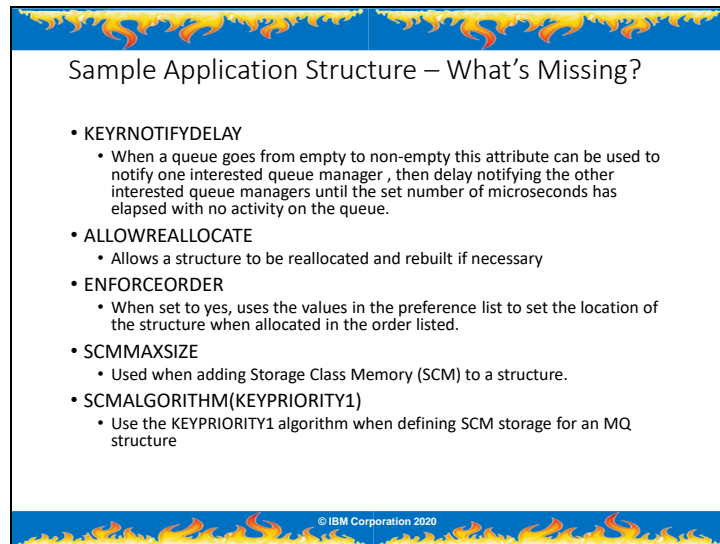
- ALLOWAUTOALT
  - If set to YES allows the system to make dynamic changes to the size and internal configuration of the list structure.
  - If allowed to default or set to NO, no dynamic resize or reallocation of entries and elements is done.
- MINSIZE – the smallest size for the structure. This is used during initial allocation (first use) or during resizing (automatic or via command).
- DUPLEX – Duplexing is a synchronous copy of the structure in a second CF.
  - Defaults to DISABLED
  - ALLOWED- Duplexing can be started by the application or via a command
  - ENABLED –Duplexing of the structure is initiated when the structure is allocated

© IBM Corporation 2020

- Allowing the systems to resize the structure when necessary can improve resilience in the event of unforeseen issues. It can provide emergency extension to storage allocations and adjust the number of entries and elements to provide better use of the storage that is allocated. Note that ALLOWAUTOALT can also allow a structure to be reduced in size, if there is unused storage and other structures in the CF need additional storage.

We recommend that this be used for application structures where resilience is the highest priority. It does not always prevent the structure from running out of storage, but it can help extend the time until putting applications start receiving '2192' return codes – an out of space condition.
- MINSIZE – often set to the INITSIZE on application structures, we recommend that this attribute be set if ALLOWAUTOALT is set to YES, to make sure the size does not get decreased to something unusable. Please see [https://www.ibm.com/support/knowledgecenter/SSLTBW\\_2.4.0/com.ibm.zos.v2r4.ieaf100/subcfrm.htm](https://www.ibm.com/support/knowledgecenter/SSLTBW_2.4.0/com.ibm.zos.v2r4.ieaf100/subcfrm.htm) for more information.
- DUPLEX – duplexing a structure is not typically recommended – and has not been since MQ V7.1, as it can add considerably to the CPU consumption and responsiveness of shared queue activity. However it can provide almost seamless failover and if there is an urgent need for extreme resiliency it can be used. Please see the High Availability WebSphere Messaging Redbook, <http://www.redbooks.ibm.com/abstracts/sg247839.html?Open> , for a detailed look at this.



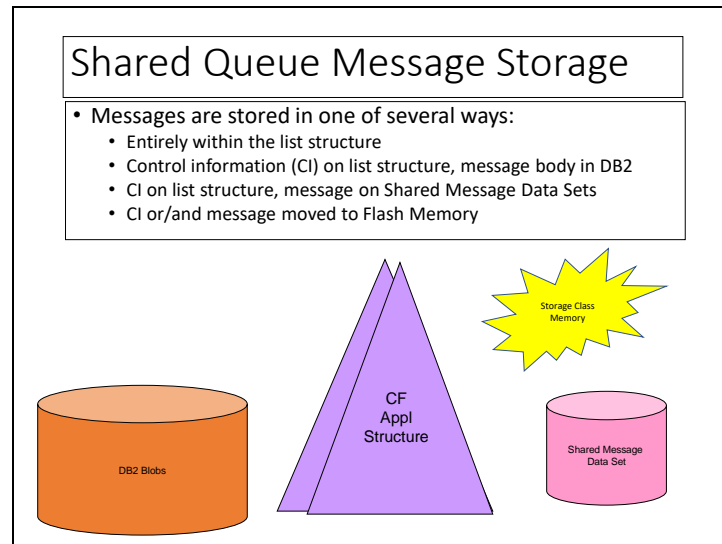


Sample Application Structure – What's Missing?

- KEYRNOTIFYDELAY
  - When a queue goes from empty to non-empty this attribute can be used to notify one interested queue manager, then delay notifying the other interested queue managers until the set number of microseconds has elapsed with no activity on the queue.
- ALLOWREALLOCATE
  - Allows a structure to be reallocated and rebuilt if necessary
- ENFORCEORDER
  - When set to yes, uses the values in the preference list to set the location of the structure when allocated in the order listed.
- SCMMAXSIZE
  - Used when adding Storage Class Memory (SCM) to a structure.
- SCMALGORITHM(KEYPRIORITY1)
  - Use the KEYPRIORITY1 algorithm when defining SCM storage for an MQ structure

© IBM Corporation 2020

- KEYRNOTIFYDELAY was added to prevent substantial CPU costs when messages are trickling in rather than in substantial volumes.  
When a message arrives on a queue taking the depth from zero to non-zero, normally all interested queue managers are informed. This can lead to substantial CPU costs when there are many potential servers, but few messages coming in as all instances will 'wake up' and race to get the message. While advertised to prevent workload skewing, this does not and can actually make it worse.
- ALLOWREALLOCATE can allow or prevent a structure from being reallocated if necessary, to move it to a more optimal location – if for example it needs to be expanded in size but there is not sufficient storage on the current CF.
- ENFORCEORDER – this is often used to make sure that the CFs used are in specified places.
- SCMMAXSIZE – Adding Storage class or flash memory, will let the CF move entries from the CF onto the SCM when the structure is 90% full (that value is fixed). This is done by the CF, MQ is not involved in this decision. This will be discussed more in the discussion on 'where messages are stored'.
- SCMALGORITHM(KEYPRIORITY1) – this is used to determine how the messages should be moved from the CF structure to the SCM.



The physical storage for Shared queue messages can be hosted in multiple physical containers:

- Messages smaller than 63K might be stored completely in the coupling facility . This is the most efficient method.
- Messages larger than 63 K will have control information storage in the Coupling Facility. The rest of the message can reside on DB2 blob. This is the most costly storage mechanism. It was the original implementation for messages larger than 63K. It was intended for messages that made up less than 10% of the total number of messages used by the structure.
- Messages might also be split between the control information on the CF and the messages on shared message datasets. (SMDS) This storage mechanism was implemented in version 7.1 and is much less expensive than messages that are stored on DB2 blobs. It is the preferred method of large message storage and is supplemented by the use of 'OFFLOAD' rules that help preserve the often-critical CF space.
- Finally messages can be moved from the coupling facility structure onto storage class or flash memory if the structure gets to 90% full or more and has been defined to use this extra memory space. This method was implemented purely by the operating system and is often used as emergency storage.

Knowing the costs and responsiveness of the different storage mechanisms will help drive good decisions on how to use this storage.



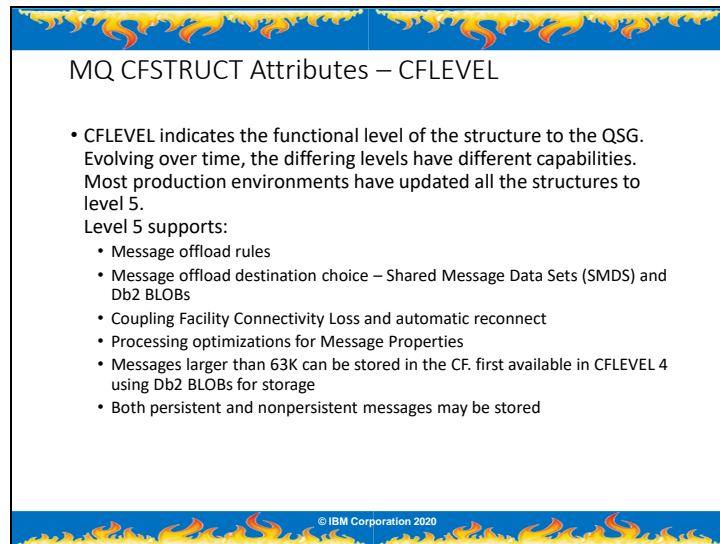
## Shared Queue Message Storage

- Shared messages are stored on one to four media
  - Control information about the message location is stored on the CF
    - One entry and two elements are reserved
    - Typically the estimate for CF storage requirements is 2000 bytes to allow for other overhead
  - Complete message is within the CF structure itself
    - All messages that are smaller than around 200 bytes
    - All messages < 63k that do not satisfy an offload rule
    - All message on a CFSTRUCT at level 3
  - Split between the CF structure and DB2 Blobs
    - All messages that are >63K and where the CFSTRUCT level is 4
    - All messages that have an offload location of DB2 and that satisfy an offload rule
  - Split between the CF structure and SMDS
    - All messages that are >63K and where the CFSTRUCT level is 5
    - All messages that have an offload location of SMDS and that satisfy an offload rule
    - Note that with SMDS the message header is stored with the message on the dataset.
  - Finally if using storage class memory (Flash)
    - Control information, and associated message data stored on the CF may be moved to the SCM when the structure reaches 90% full



## MQ CFSTRUCT definitions

- The CFSTRUCT is the MQ object definition of the physical coupling facility structure
- Like the STGCLASS for private queue, it is used to tell the queue managers in the QSG where the queues reside
- It contains several attributes that extend the way the structures and the messages they hold are treated.
- The first CFSTRUCT attribute is the name. It is different from the name of the structure (as it is defined to the CF itself).
  - The MQ CFSTRUCT definition does not include the QSG name
  - To MQ the structure name is SMDSMSGS
  - To the CF it is QSGMSMDSMSGS



The slide is titled "MQ CFSTRUCT Attributes – CFLEVEL" and is framed by a decorative border with a blue background and yellow and orange flame-like patterns at the top and bottom. The content is as follows:

MQ CFSTRUCT Attributes – CFLEVEL

- CFLEVEL indicates the functional level of the structure to the QSG. Evolving over time, the differing levels have different capabilities. Most production environments have updated all the structures to level 5.

Level 5 supports:

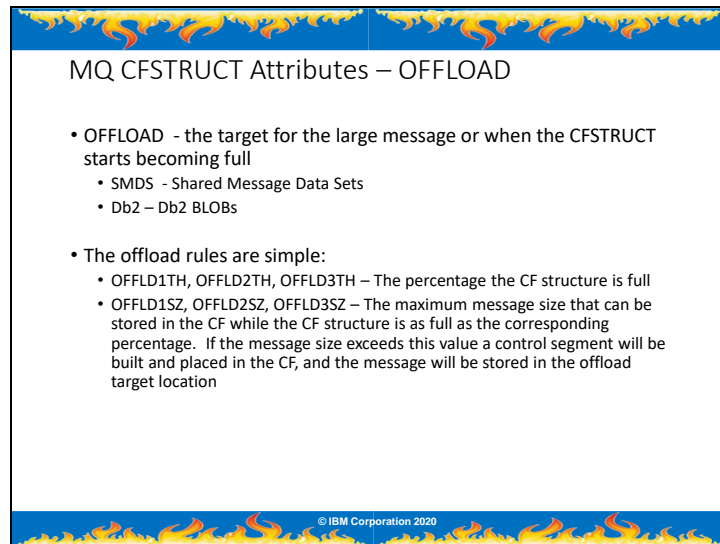
- Message offload rules
- Message offload destination choice – Shared Message Data Sets (SMDS) and Db2 BLOBs
- Coupling Facility Connectivity Loss and automatic reconnect
- Processing optimizations for Message Properties
- Messages larger than 63K can be stored in the CF, first available in CFLEVEL 4 using Db2 BLOBs for storage
- Both persistent and nonpersistent messages may be stored

© IBM Corporation 2020

The CFLEVEL should not be confused with the Coupling facility code level (typically called CFCC XX, where we'd expect the XX to be something like 21 or 22), this level controls how MQ uses the structure. The levels represent the evolution of the use of the structures.

When first introduced, only nonpersistent messages less than 63K were supported in shared queues. Due to the reliability and availability of the CF structures messages are often called 'semi-persistent'. The next iteration allowed for persistent messages (CFLEVEL 3 enabled – the lowest level currently supported).

CFLEVEL 4 introduced the ability to storage messages larger than 63K, by storing the control information in the CF and the messages in DB2 BLOBs. This is still used when the percentage of large message is small, though the use is discouraged.



The slide is titled "MQ CFSTRUCT Attributes – OFFLOAD" and is framed by a decorative border with a blue background and yellow and orange flame-like patterns at the top and bottom. The content is organized into a bulleted list.

MQ CFSTRUCT Attributes – OFFLOAD

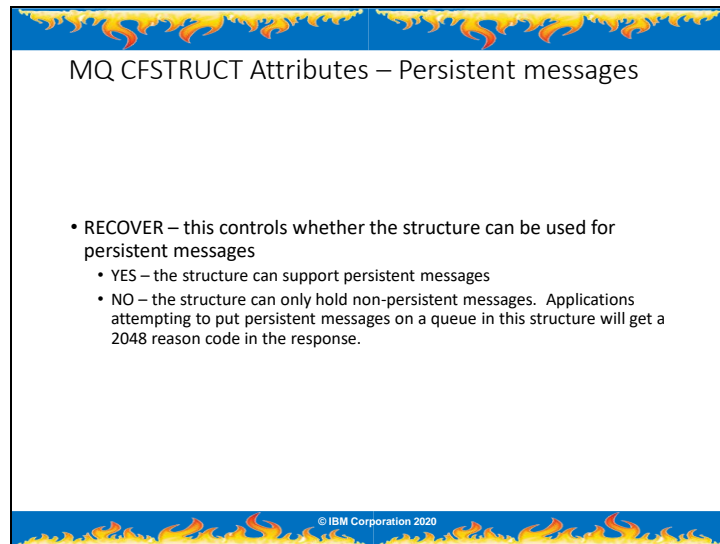
- OFFLOAD - the target for the large message or when the CFSTRUCT starts becoming full
  - SMDS - Shared Message Data Sets
  - Db2 – Db2 BLOBs
- The offload rules are simple:
  - OFFLD1TH, OFFLD2TH, OFFLD3TH – The percentage the CF structure is full
  - OFFLD1SZ, OFFLD2SZ, OFFLD3SZ – The maximum message size that can be stored in the CF while the CF structure is as full as the corresponding percentage. If the message size exceeds this value a control segment will be built and placed in the CF, and the message will be stored in the offload target location

© IBM Corporation 2020

The OFFLOAD attribute defines the storage to be used for messages larger than 63K. Db2 was the original method, but SMDS is the preferred.

The off load rules have 2 attributes, the first is the percentage full specification for the structure. That information is returned to the queue manager from the CF. The second is the maximum size of the messages that should be held in the CF.

The three pairs are designed to be used to conserve the CF storage, offloading the messages. When the structure reaches the full percentage, messages larger than the specified size are offloaded, leaving only the one element two entry structure in the CF itself. That construct contains the message key information and the location of the message itself – the RBA of the message on SMDS.

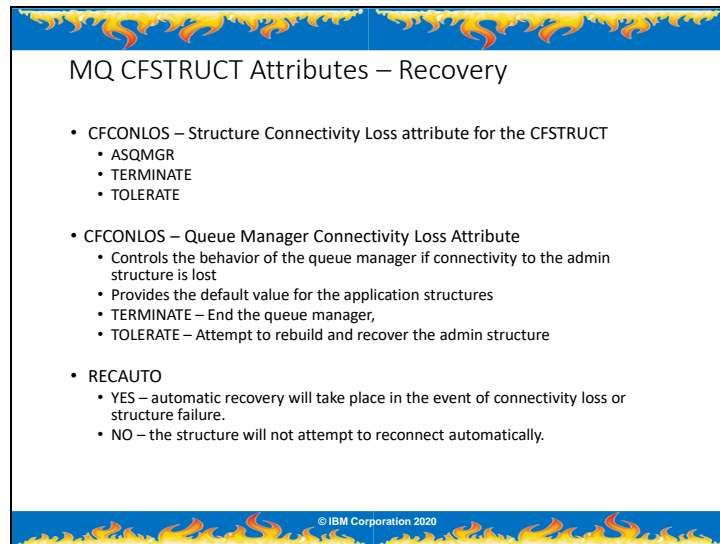


MQ CFSTRUCT Attributes – Persistent messages

- RECOVER – this controls whether the structure can be used for persistent messages
  - YES – the structure can support persistent messages
  - NO – the structure can only hold non-persistent messages. Applications attempting to put persistent messages on a queue in this structure will get a 2048 reason code in the response.

© IBM Corporation 2020

- The RECOVER attribute tells the QSG if the structure can hold persistent messages.
- When set to YES, it is important that the 'BACKUP CFSTRUCT' command be issued periodically. That command will be discussed in more detail later in the session.
- When set to NO – this has been known to cause application issues when moving from private to shared queues because the application can get this 'new' (to them at least), unrecognized reason code on a PUT.
- We are occasionally asked if this should always be set to yes to avoid any potential application problems, and I tend to say yes and make sure the settings for connectivity loss and automatic reconnect are set to yes. Setting all these properly allows the queue sharing group and the Sysplex to automatically attempt to recover in the event of the most common issues



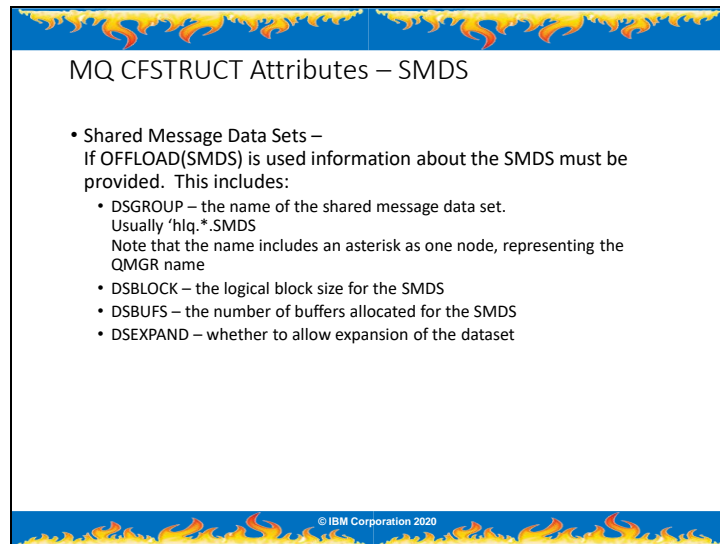
MQ CFSTRUCT Attributes – Recovery

- CFCONLOS – Structure Connectivity Loss attribute for the CFSTRUCT
  - ASQMGR
  - TERMINATE
  - TOLERATE
- CFCONLOS – Queue Manager Connectivity Loss Attribute
  - Controls the behavior of the queue manager if connectivity to the admin structure is lost
  - Provides the default value for the application structures
  - TERMINATE – End the queue manager,
  - TOLERATE – Attempt to rebuild and recover the admin structure
- RECAUTO
  - YES – automatic recovery will take place in the event of connectivity loss or structure failure.
  - NO – the structure will not attempt to reconnect automatically.

© IBM Corporation 2020

- The CFCONLOS attribute is both a queue manager and an individual structure attribute, it controls what should happen if the queue manager loses connectivity to a structure. Losing connectivity to a CF structure is more common than a structure failure – and is often a temporary situation. From a CFSTRUCT definition perspective the possible values are:  
ASQMGR – this is the default, and takes the action specified by the queue manager attribute.  
TERMINATE – end the queue manager  
TOLERATE – the queue manager will tolerate the loss of connectivity and depending on the RECAUTO setting, may attempt to recover and rebuild the structure.
- The CFCONLOS attribute for the queue manager tells the queue manager what to do when there is a connectivity loss to the admin structure.  
TERMINATE – will end the queue manager and will probably result in a QSG outage. For this reason, I recommend using TOLERATE.  
TOLERATE – attempt to recover and rebuild the admin structure.  
Note that we still see the admin structure duplexed to avoid this critical situation, even with the other structures are not duplexed.
- RECAUTO tells the queue managers what to do if the queue manager cannot communicate with the structure or if there is a structure failure reported.





The slide is titled "MQ CFSTRUCT Attributes – SMDS" and is framed by a decorative border with a blue background and yellow and orange flame-like patterns at the top and bottom. The content is as follows:

MQ CFSTRUCT Attributes – SMDS

- Shared Message Data Sets –  
If OFFLOAD(SMDS) is used information about the SMDS must be provided. This includes:
  - DSGROUP – the name of the shared message data set.  
Usually 'hlq.\*.SMDS'  
Note that the name includes an asterisk as one node, representing the QMGR name
  - DSBLOCK – the logical block size for the SMDS
  - DSBUFS – the number of buffers allocated for the SMDS
  - DSEXPAND – whether to allow expansion of the dataset

© IBM Corporation 2020

1) The DSGROUP is the name of the dataset. Each queue manager in the QSG must have its own unique dataset that is pre-allocated and formatted for use.

Member CSQ4SMDS contains the sample JCL for this. Note that when populating the DSGROUP value the asterisk is used. When the dataset is first needed the queue manager replaces the \* with the 4-character QMGR name.

2) The DSBLOCK is the blocksize used, it defaults to 256K. This is tunable, but we've not seen any examples of this being done to date. As we see more use of SMDS, we do expect to see this used more and tuned to the needs of the structure. If, for example most of the messages that are placed in the SMDS are just above 63K, it may be advantageous to set this value to 64K to use the storage more effectively. If the dataset is regularly used for smaller messages due to the offload rules firing, then a smaller value might be more effective.

3) DSBUFS – the number of buffers allocated for each queue manager for the SMDS. This number ranges from 1-9999. Again, this is not a value we have seen tuned, but as more environments make use of SMDS tuning the number of buffers may become more valuable.

4) DSEXPAND – this YES or NO value indicates whether the SMDS can expand if necessary. If SMDS is primarily used for message offloads during an emergency, then this would be set to YES to avoid (if possible) the CF filling and applications being unable to put messages.

The slide features a blue header and footer with a yellow and orange flame-like pattern. The main content area is white. The title 'Sizing the CF Structures – How many?' is in black. Below it is a bullet point: '• How many application structures are needed?'. In the center, the text '•IT DEPENDS!!!!' is displayed in large, bold, blue letters. At the bottom of the footer, the text '© IBM Corporation 2020' is visible.

Sizing the CF Structures – How many?

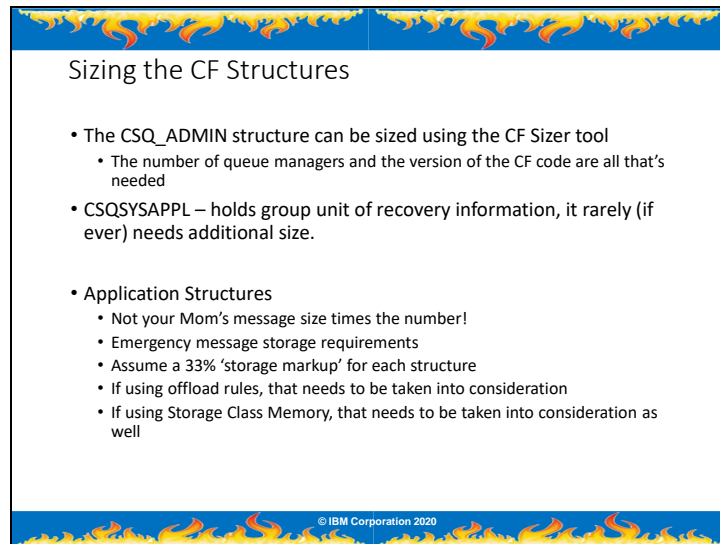
- How many application structures are needed?

**•IT DEPENDS!!!!**

© IBM Corporation 2020

- The number of application structures varies.
- Having too few, especially when co mingling well known applications with unproven applications can lead to sympathy sickness or in some cases, sympathy death, of the well behaved ones.
- Having too many can mean a lot of wasted storage, and having structures that are too small to host the workload. Its import to remember that the CF List structure always has quite a bit of storage overhead, so when 500 Mb are allocated, the actual storage for the queues is  $\frac{1}{2}$  to  $\frac{2}{3}$  of that allocated space. I have seen one environment where there were new structures added for each application, I believe they had 17 when I looked at their environment. While I am sure this made their hardware salesman happy, it was actually hurting the applications because there was not enough storage at times. I am planning on working with this customer to help reduce the number and increase the size in early 2021.
- With the fairly recent addition of the RKEYNOTIFYDELAY attribute on the CF structure, it might benefit some queues so substantially that moving them to a structure with this attribute outweighs the CPU costs of committing a UoR that crosses structures.





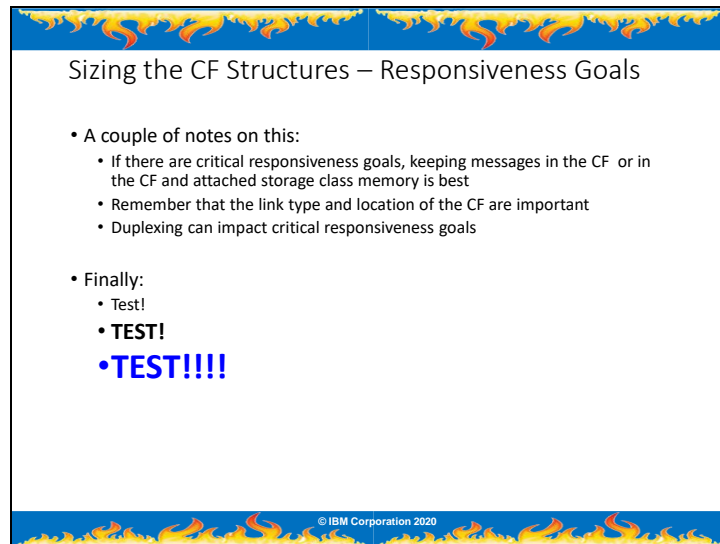
### Sizing the CF Structures

- The CSQ\_ADMIN structure can be sized using the CF Sizer tool
  - The number of queue managers and the version of the CF code are all that's needed
- CSQSYSAPPL – holds group unit of recovery information, it rarely (if ever) needs additional size.
- Application Structures
  - Not your Mom's message size times the number!
  - Emergency message storage requirements
  - Assume a 33% 'storage markup' for each structure
  - If using offload rules, that needs to be taken into consideration
  - If using Storage Class Memory, that needs to be taken into consideration as well

© IBM Corporation 2020

- The CSQ\_ADMIN structure sizing is straightforward. Please note as hardware and software is upgraded or as more queue managers are added, this does need to be checked. Upgrades to the CF code levels have been known to increase the storage requirements for this structure substantially. There has been some growth in requirements in recent upgrades, but not to the levels in the past, but once burned....
- The CSQSYSAPPL structure is typically left at the default size.
- Application structures
  - Knowing the message sizes and the length of time they 'live' on a queue is possibly more important when sizing for a Cf structure than for bufferpools and pagesets
  - An often-overlooked consideration is how many messages might have to be stored in the event of an emergency . Some organizations have externally imposed rules, like they must keep 4 hours of messages (or longer) when there is an outage.
  - When sizing application structures, because of the way storage is used within a structure there is typically around 32-33% that is used for the management of the structure. That might be lower, you might get better use of the storage, if the messages are of a consistent size and the structure can reset the entry to element ratio by taking advantage of the ALLOWAUTOALT feature. Sounds like a lot, does it?
  - We developed a sample sizing spreadsheet that included the various storage 'controls', but it lacks the complexity of the 3 offload rules (just used one).

Happy to share if requests, and most importantly if we can get feedback as to accuracy.



### Sizing the CF Structures – Responsiveness Goals

- A couple of notes on this:
  - If there are critical responsiveness goals, keeping messages in the CF or in the CF and attached storage class memory is best
  - Remember that the link type and location of the CF are important
  - Duplexing can impact critical responsiveness goals
- Finally:
  - Test!
  - **TEST!**
  - **TEST!!!!**

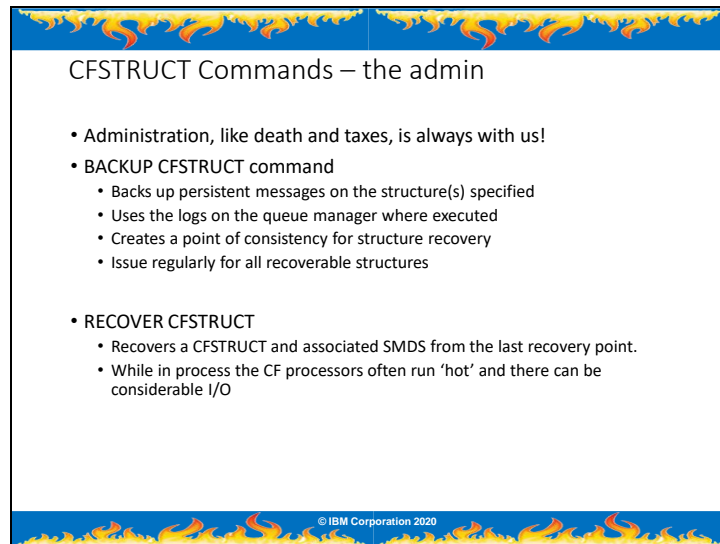
© IBM Corporation 2020

Responsiveness goals should drive where your messages are physically stored. It's always better to have messages that are completely within the coupling facility or in the coupling facility an attached storage class memory to help meet those goals. Anytime messages have to be divided across 2 storage locations that is control information in the coupling facility and the message itself in auxiliary storage like DB2 blobs or shared message datasets, that will slow down processing and cost more in CPU. That doesn't mean you shouldn't use off load rules for these structures what it does mean is that you need to think about what those goals are and use the auxiliary storage judiciously . Also keep in mind that the link type that is the physical links from the L part where your applications run an the coupling facility structure itself , and the physical location of the coupling facility are also critically important to responsiveness. We were asked recently about Huawei requests that were being processed by one queue manager on a particular L par took so much longer then other requests running in another alpar even though the first alpar was more powerful it had more dedicated ingens it had fewer resources being consumed by other applications. It was a simple matter of the physical location of the coupling facility which happened to be Co located on the same keck as the ALP are hosting the queue manager that was meeting the goals.

System manage duplexing for an application structure is sometimes used as a way to make certain that there is little interruption to messaging in the event of some sort of coupling facility problem. However this duplexing has caused just as many problems I believe as it's relieved. In one particularly critical example the duplexing was done between coupling facilities that were at the far ends of the maximum distance you could have for a sysplex environment an when the customer turned on duplexing the applications suddenly started failing becausw structures were getting full

responsiveness was not being met there were just a huge number of issues that were introduced because of the time it took for the physical duplexing to take place for every request that was made against that particular structure.

Finally to summarize there are a lot of moving parts here and it's really important that you test what's being done in your environment. We can offer all kinds of advice and scenarios on how to size your structures what kind of responsiveness you will probably get based on the type of links that you have and the physical distance etc but nothing beats actual testing. Please keep this in mind and remember that it's always advisable to create a series of simple reproducible tests that you can use in your environment to verify that things are working the way you think they should that your responsiveness goals are going to be met and in the event of some sort of unforeseen critical problem your entire environment can respond correctly.



CFSTRUCT Commands – the admin

- Administration, like death and taxes, is always with us!
- BACKUP CFSTRUCT command
  - Backs up persistent messages on the structure(s) specified
  - Uses the logs on the queue manager where executed
  - Creates a point of consistency for structure recovery
  - Issue regularly for all recoverable structures
- RECOVER CFSTRUCT
  - Recovers a CFSTRUCT and associated SMDS from the last recovery point.
  - While in process the CF processors often run 'hot' and there can be considerable I/O

© IBM Corporation 2020

So the first command I'm going to talk about here is the backup CF struct command. This command backs up persisted messages that are on these specified structures so can only be used against a structure that is defined as recoverable. The messages are read from the structure and written to the queue manager log on the queue manager where the command is executed. This not only backs up the messages, it creates a point of consistency for the structure and gives the QSG a place to start if structure recovery is necessary. Structure recovery will start at that consistency point and roll thru the logs for all the QSG queue managers to restore the structure contents. **IT IS VERY IMPORTANT** that this command be issued regularly.

Most customers automate this command and select the queue manager queue managers where they want it to run where the I/O to the queue manager logs will not be disruptive.

The general recommendation from the development lab is to run this at least once an hour and back up all the structures in the QSG. My recommendations tend to focus on your return to service objectives, and include knowing the MQ log turnover rate. Simply the recovery will be much faster if the last recovery point is on the current active log of one of the queue managers in the QSG and if all of the subsequent log entries for all the queue managers for this structure are as well. If the log entries required to rebuild the structure have been archived – especially if they have been archived to tape, recovery will be slower. I generally recommend it be run every 15 minutes on a busy QSG, maybe more frequently for a really busy high persistent message volume environment. If this command is not issued regularly if there is a problem with the structure, chances are very good that the QSG will attempt to read MQ logs back to the beginning of the

logs since the queue manager was created. This almost invariably occurs during a time of stress.

The BACKUP CFSTRUCT command has

The next command is to recover the CF structure. It is most often used as part of a disaster recovery test or a real disaster recovery scenario. It may occasionally have to be used especially if you do not have automatic recovery set on a particular structure. Some things to note:

- The command will fail unless one or both the structure or the SMDS are in a failed state.
- If the SMDS is in a failed state, but the structure isn't – the structure is set to failed and all the contents are deleted from both storage areas. In practical terms this means that the structure is not usable until the recovery is complete and nonpersistent messages are deleted.
- A recover command can be issued with the purge type, that will completely empty the structure and associated SMDS. This is often used when for one reason or another the latest backup is not available – or there never was one taken. This will cause data loss.

### CFSTRUCT Commands – BACKUP CFSTRUCT

- If CFSTRUCT BACKUP is not being done:

```

SDSF OUTPUT DISPLAY ZMQ1MSTR STC02637 DSID 2 LINE CHARS 'BACKUP' FOUND
COMMAND INPUT ==>                                SCROLL ==> CSR
15.52.21 STC02637 CSQE041E ZMQ1 CSQECFBI Structure CSQSYSAPPL backup is 912
          912 more than a day old
    
```

- BACKUP CFSTRUCT example:  
 ZMQ1 BACKUP CFSTRUCT(\*) CMDSCOPE(ZMQ3) EXCLINT(40)  
 Where:
  - The \* directs that all recoverable structures be backed up
  - The CMDSCOPE gives the execution queue manager
  - EXCLINT(40) is the exclusion interval

© IBM Corporation 2020


- If a BACKUP CFSTRUCT command is not issued for a recoverable resource in over 24 hours, message CSQE041E is emitted on all the QMGRs in the QSG as shown. There is a less urgent message CSQE040I, that indicates that a structure has not been backed up for 2 hours as well. Some customers have set up automation to run the BACKUP CFSTRUCT command when they receive the informational message. Personally, I think that is too long because any recovery would be from at least 2 hours ago and have to read the logs across all the queue managers in the QSG.
- The example – an instruction from our environment
  - In this case the asterisk will cause all the recoverable structures to be backed up. Specific structure names or wildcarded partial names are also acceptable. This parameter is required.
  - The CMDSCOPE, which is optional, in this case tells the queue manager the command is to be executed on queue manager ZMQ3, another member of the QSG
  - The EXCLINT is the exclusion interval, and directs that messages put in the last few seconds, in our example 40, are to be excluded from the back-up. This can help limit the impact of the backup. As delivered the default is 30 seconds – which is the minimum.

## CFSTRUCT Commands – BACKUP CFSTRUCT

- The results of the command:

```
17.17.51 STC02139 CSQE120I ZMQ3 CSQELBK1 Backup of structure LARGMSG 993
993 started at RBA=000000000048F622
17.17.51 STC02139 CSQE120I ZMQ3 CSQELBK1 Backup of structure CSQSYSAPPL 994
994 started at RBA=000000000048F66A
17.17.51 STC02139 CSQE121I ZMQ3 CSQELBK1 Backup of structure CSQSYSAPPL 995
995 completed at RBA=000000000048F6B2, size 0 MB
17.17.51 STC02139 CSQE121I ZMQ3 CSQELBK1 Backup of structure LARGMSG 996
996 completed at RBA=000000000048F6FA, size 0 MB
```






### CFSTRUCT Commands – RECOVER CFSTRUCT

- RECOVER CFSTRUCT example:  
ZMQ1 RECOVER CFSTRUCT(CSQSYSAPPL) CMDSCOPE(ZMQ3)  
TYPE(NORMAL)

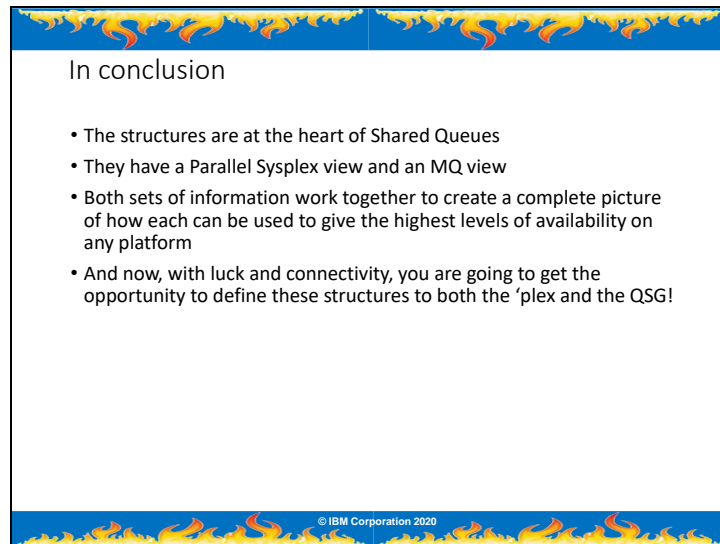
Where:

- Up to 63 structure names can be specified.
- The CMDSCOPE gives the execution queue manager
- TYPE(NORMAL) is the default, and will recover a failed structure and SMDS, restoring persistent messages from the logs
- TYPE(PURGE) may also be used for application structures, and all messages will be removed
  - If attempted against the CSQSYSAPPL command it will fail

© IBM Corporation 2020



- The RECOVER CFSTRUCT command recovers a structure and SMDS if one, the other or both are in a failed state
- The structure name must be specified. If multiple structures need to be recovered, it is more efficient to do them all at once.
- TYPE(PURGE) will clear a failed structure and SMDS.



### In conclusion

- The structures are at the heart of Shared Queues
- They have a Parallel Sysplex view and an MQ view
- Both sets of information work together to create a complete picture of how each can be used to give the highest levels of availability on any platform
- And now, with luck and connectivity, you are going to get the opportunity to define these structures to both the 'plex and the QSG!

© IBM Corporation 2020