IBM MQ

Presenting a technical solution pattern for

Enabling Auto-Segmentation of Messages That Have Properties Defined on Them

Greg Brown
CACI International Inc. (caci.com)

In This Session …

This session is for messaging application developers who wish to exploit the message auto-segmentation feature of IBM® MQ queue managers for messages on which properties have been defined.

Why????

Because, OOTB, *MQ has never been able to do this!*

Typical work-around is to increase the max message size of every relevant IBM® MQ object (up to 100MiB) and be done with it.

- Acceptable for smaller messaging infrastructures.
- Administrative footprint increases geometrically as messaging infrastructure grows.
- Prefer a solution option that has zero administrative impact and is simple to implement.

What We'll Cover …

- Overview of background history on problem discovery and interaction with L2 / L3 support.
- Overview of queue manager mediated auto-segmentation of large messages.
- A description of the fault-generating environment.
- An investigation into the fault mechanics.
- Presentation of a technical implementation pattern.
  - Code enhancements to the message putting application
  - Code enhancements to the message getting application

Some context and history …

Messaging environment:

- Messages have several properties defined on them:
  - total length of property names and their values < 2KiB.
- More than 98% of messages are less than 2KiB in size.
- Messages containing monthly, quarterly and annual aggregate data can be several tens of MiB in size.
- The default 4 MiB limit on message size for (most) IBM® MQ objects was quite acceptable:
  - simplified post-configuration of IBM® MQ deployments.
- Rely on queue manager mediated auto-segmentation of the relatively few messages > 4 MiB in size.

Some context and history (cont.) …

- November 15, 2013:
  * Opened PMR 92419.499.000 (Sev2).
- December 10, 2013:
  * APAR IC98231 created.
- February 4, 2014:
  * WebSphere MQ FP 7.5.0.3 announced; IC98231 not included!
- February 11, 2014:
  * L3 identifies possible solution; L2 proceeds with prototyping the fix.
- February 21, 2014:
  * L3 announces that no code fix will be done; will document the bug as a feature “restriction”!

From IBM L2 support 2013.11.20

N

O

T

E

S

"I reviewed the traces files and did find the first part of the segmented message was larger than the **MAXMSGL** for the SCTQ (system cluster transmission queue).

I also found that the channel agent for the cluster channel was unable to get this message because even though it did several calls to get the message from the queue with a buffer size large enough to hold it the **MQGET** call only returned 4MiB and an error that the message was truncated.

I searched but could not find a match to this problem. I have sent the PMR to our L3 team to review. Since this is a Sev2 PMR I need to ask what is the timeframe needed to resolve the issue."

From IBM L3 support 2013.12.10

N O T E S

"I have isolated the cause of the issue. When the **MCA** reads the message from the queue, the properties are prepended to the message in an **MQRFH2** header. This header causes the message length to increase beyond the **MAXMSGL**, and so the message cannot be read from the queue.

Please raise an APAR for this issue. I will need to consult development to devise the best solution for this issue, so the APAR will not be fixed quickly; the earliest date that I expect the APAR to be completed is the end of the month, 31st Dec."

From IBM L2 support 2014.02.11

N

O

T

E

S

"A possible method of fixing this issue has been agreed following consultation with Development. I intend to prototype the suggested fix and consult Development again before completing the APAR.

This will delay the completion of the APAR until L3 and Development are entirely satisfied that the proposed fix will not have unintended consequences, or cause any regressions.

Note that one outcome of the APAR is that only the documentation will be amended to cover this situation. I expect that a final decision on the resolution of this APAR will be taken by Feb 28; if a code fix is done, then it can be made available at 7.5.0.[4]."

From IBM L2/L3 support 2014.04.21-22

NOTES

"It has been decided following consultation with Development that this issue will be documented as a restriction, rather than a code fix being made." (*the next day*) "Let me explain our intentions...

The current plan for IC98231 is to close it as a documentation change, altering the InfoCenter to state that the channel's maximum message length attribute should take into account the length of any message properties which may be attached to the message.

Making the code changes which would be needed to resolve this issue have been deemed as extremely risky (due to concerns on how receiving clients may reconstruct the message) - and we don't want to risk breaking compatibility with older clients, nor regressing this important function, potentially causing serious issues for our customers.

The feeling in the lab is that the benefit of fixing the problem is significantly outweighed by the risks introduced by that fix. Given that the workaround for the issue is simple and well understood, it is felt that documenting the workaround as a restriction is the lesser of two evils."

IBM® MQ Queue Manager Mediated Auto-Segmentation of Large Messages

QMMAS has been a feature of distributed IBM® MQ since at least version 2 (twenty years ago as "MQ Series").

Technical implementation of QMMAS is fairly simple:

On the MQPUT side:

- `PMO.Options = (app-specific options)`
- `MD.MsgFlags = MQMF_SEGMENTATION_ALLOWED`
- `memcpy(MD.GroupId, MQGI_NONE, MQ_GROUP_ID_LENGTH)`

On the MQGET side:

- `GMO.Options = MQGMO_COMPLETE_MSG | (app-specific options)`

"*The queue manager splits messages into segments as necessary so that the segments (plus any required header data) fit on the queue.*"

QMASS

N O T E S

Segmentation and reassembly by the queue manager:

- This is the simplest scenario, in which one application puts a message to be retrieved by another. The message might be large: not too large for either the putting or the getting application to handle in a single buffer, but too large for the queue manager or a queue on which the message is to be put.
- The only changes necessary for these applications are for the putting application to authorize the queue manager to perform segmentation if necessary:
  - **PMO.Options** = (existing options)
  - **MD.MsgFlags** = **MQMF_SEGMENTATION_ALLOWED**
  - memcpy(**MD.GroupId**, **MQGI_NONE**, **MQ_GROUP_ID_LENGTH**)(?)
  - **MQPUT**
- And for the getting application to ask the queue manager to reassemble the message if it has been segmented:
  - **GMO.Options** = **MQGMO_COMPLETE_MSG |** (existing options)
  - **MQGET**

QMASS

N O T E S

Segmentation and reassembly by the queue manager (cont.):

- In this simplest scenario, the application must reset the **GroupId** field to **MQGI_NONE** before the **MQPUT** call, so that the queue manager can generate a unique group identifier for each message. If this is not done, unrelated messages can have the same group identifier, which might subsequently lead to incorrect processing.

But then, the documentation immediately goes on to say (rather cryptically):

- If the **MAXMSGL** attribute of a queue is to be modified *to accommodate message segmentation*, then consider: The minimum message segment supported on a local queue is 16 bytes.
- For a transmission queue, **MAXMSGL** must also include the space required for headers. Consider using a value at least 4000 bytes larger than the maximum expected length of *user data* in any message segment that could be put on a transmission queue.

I have not found anything (not saying it isn't there) in the documentation that explains what is meant by "*accommodate message segmentation*" in regards to queue manager mediate auto-segmentation of messages. Was not the word *'only'* used (item 2 in the previous slide) to circumscribe actions required to enable auto-segmentation? Refer to slide 6.5 for further (re)assurance that the queue manager takes care of everything.

QMASS

N

O

T

E

S

Segmentation and reassembly by the queue manager (cont.):

MQMD.MsgFlags: MQMF_SEGMENTATION_ALLOWED

- This option allows the message to be broken into segments by the queue manager. If specified for a message that is already a segment, this option allows the segment to be broken into [even] smaller segments. **MQMF_SEGMENTATION_ALLOWED** can be set without either **MQMF_SEGMENT** or **MQMF_LAST_SEGMENT** being set.
- When the queue manager segments a message, the queue manager turns on the **MQMF_SEGMENT** flag in the copy of the **MQMD** that is sent with each segment, but does not alter the settings of these flags in the **MQMD** provided by the application on the **MQPUT** or **MQPUT1** call. For the last segment in the logical message, the queue manager also turns on the **MQMF_LAST_SEGMENT** flag in the **MQMD** that is sent with the segment.

QMASS

N

O

T

E

S

Segmentation and reassembly by the queue manager (cont.):

Interesting…

Take care when putting messages with **MQMF_SEGMENTATION_ALLOWED** *but without* ***MQPMO_LOGICAL_ORDER*** (*MQPUT only*). If the message is:

- Not a segment, and
- Not in a group, and
- Not being forwarded,

[then] the application must reset the **GroupId** field to **MQGI_NONE** before *each* **MQPUT** or **MQPUT1** call so that the queue manager can generate a unique group identifier for each message. If this is not done, unrelated messages can have the same group identifier, which subsequently might lead to incorrect processing. See the descriptions of the **GroupId** field and the **MQPMO_LOGICAL_ORDER** option for more information about when to reset the **GroupId** field.

QMASS

N O T E S

Segmentation and reassembly by the queue manager (cont.):

The queue manager splits messages into segments as necessary so that the segments (plus any required header data) fit on the queue. However, there is a lower limit for the size of a segment generated by the queue manager, and only the last segment created from a message can be smaller than this limit (the lower limit for the size of an application-generated segment is one byte). Segments generated by the queue manager might be of unequal length. The queue-manager processes the message as follows:

- User-defined formats are split on boundaries that are multiples of 16 bytes; the queue manager does not generate segments that are smaller than 16 bytes (other than the last segment).
- Built-in formats other than **MQFMT_STRING** are split at points appropriate to the nature of the data present. However, the queue manager never splits a message in the middle of an **IBM® MQ** header structure. This means that a segment containing a single MQ header structure cannot be split further by the queue manager, and as a result the minimum possible segment size for that message is greater than 16 bytes.

QMASS

N

O

T

E

S

Segmentation and reassembly by the queue manager (cont.):

The second or later segment generated by the queue manager begins with one of the following:

- An MQ header structure
- The start of the application message data
- Part of the way through the application message data

- **MQFMT_STRING** is split without regard for the nature of the data present (SBCS, DBCS, or mixed SBCS/DBCS). When the string is DBCS or mixed SBCS/DBCS, this might result in segments that cannot be converted from one character set to another. The queue manager never splits **MQFMT_STRING** messages into segments that are smaller than 16 bytes (other than the last segment).
- The queue manager sets the **Format**, **CodedCharSetId**, and **Encoding** fields in the **MQMD** of each segment to describe correctly the data present at the *start* of the segment; the format name is either the name of a built-in format, or the name of a user-defined format.

QMASS

N

O

T

E

S

Segmentation and reassembly by the queue manager (cont.):

- The **Report** field in the **MQMD** of segments with **Offset** greater than zero is modified. For each report type, if the report option is **MQRO_*_WITH_DATA**, but the segment cannot contain any of the first 100 bytes of user data (that is, the data following any **IBM® MQ** header structures that may be present), the report option is changed to **MQRO_***.

The queue manager follows the above rules, but otherwise splits messages unpredictably; do not make assumptions about where a message is split.

- For persistent messages, the queue manager can perform segmentation only within a unit of work.
- Take special care when converting data messages that might be segmented. Specifying **MQGMO_COMPLETE_MESSAGE** insures any conversion processing is performed on the entire message, not a segment.
- Note also that **MQGMO_COMPLETE_MESSAGE** is the only option that directs the queue manager to reassemble message segments.

QMASS

N O T E S

Segmentation and reassembly by the queue manager (cont.):

MQGMO.Options: MQGMO_COMPLETE_MSG:

- To use this (**MQGET**) option, the application must provide a buffer that is big enough to accommodate the complete message, or specify the **MQGMO_ACCEPT_TRUNCATED_MSG** option.
- **[[** If a message handle is specified in the **MQGMO** data structure, then on an **MQRC_TRUNCATED_MSG_FAILED** error returned on an attempted get of a too-large segmented message, the **MQGET DataLength** field is set to the length of the first segment of the message, ***NOT*** *the length of the fully reassembled message*. If **MQGMO_ACCEPT_TRUNCATED_MSG** was set, then under a regular destructive **MQGET** the message is removed from the queue, not exactly what we want. Typically, to get the size of the full message you must do an **MQGET** with browse which will provide the message's full size in the **DataLength** field whilst leaving the message on the queue. Then, having appropriately sized the message buffer, a regular destructive **MQGET** can be issued. However, none of this ultimately concerns us here because the size of the message is one of the message properties whose value we'll query and use to size our message buffer accordingly. **]]**

IBM® MQ Message Properties

Message properties are introduced in **WebSphere ® MQ** version 7.0 (late 2nd quarter 2008) along with other goodies like:

- embedded pub/sub
- shared client channels
- enhanced **JMS** implementation
- message selection (by the queue manager)

In a nutshell, using message properties eliminates the need to programmatically access the **MQMD** and **MQRFH2** headers in order to manage messages based on information contained in those headers.

[[To manage messages based on information contained in the message data payload, preferably you would use a tool like the **IBM® Integration Bus** product.]]

IBM® MQ Message Properties (cont.)

Message properties and message length:

- The total size of a property is the length of the property name in bytes plus the length of the property value in (represented) bytes, plus…
- Some control data for the set of properties after the first (or perhaps only) property is added to the message.
- Paraphrasing the documentation: "On an MQPUT, MQPUT1 or MQGET call, the length of properties defined on a message ***do not count*** toward the length of the message as far as the queue and the queue manager are concerned!" [But they likely will count if the message is segmented.]
- Set queue manager attribute *MaxPropertiesLength* (MAXPROPL) generously or, perhaps better, set it to NOLIMIT.
- Allocate the message buffer size to be at least 105-110% of the queue manager's *MaxMsgLength* (MAXMSGL) value.

IBM® MQ Message Properties

N O T E S

Message properties and message length:

- Use the queue manager attribute **MaxPropertiesLength** to control the size of the properties that can flow with any message in a **IBM® MQ** queue manager.
- *In general, when you use* **MQSETMP** *to set properties, the size of a property is the length of the property name in bytes, plus the length of the property value in bytes as passed into the* **MQSETMP** *call.* It is possible for the character set of the property name and the property value to change during transmission of the message to its destination because these can be converted into Unicode; in this case *the size of the property might change*.
- *On an* **MQPUT** *or* **MQPUT1** *call, properties of the message do not count toward the length of the message for the queue and the queue manager*, but they do count toward the length of the properties as perceived by the queue manager (whether or not they were set using the message property **MQI** calls).
- If the size of the properties exceeds the maximum properties length, the message is rejected with **MQRC_PROPERTIES_TOO_BIG**. Because the size of the properties is dependent on its representation, *you should set the maximum properties length at a gross level.*

IBM® MQ Message Properties

N O T E S

Message properties and message length (cont.):

- It is possible for an application to successfully put a message with a buffer that is larger than the value of **MaxMsgLength**, if the buffer includes properties. This is because, even when represented as **MQRFH2** elements, *message properties do not count toward the length of the message*. The **MQRFH2** header fields add to the *properties length* only if one or more folders are contained and every folder in the header contains properties.
- ***If one or more folders are contained in the **MQRFH2** header and any folder does not contain properties, [then] the **MQRFH2** header fields count toward the message length instead.*
- *On an **MQGET** call, properties of the message do not count toward the length of the message as far as the queue and the queue manager are concerned.* However, because the properties are counted separately it is possible that the buffer returned by an **MQGET** call is larger than the value of the **MaxMsgLength** attribute.

IBM® MQ Message Properties

N O T E S

Message properties and message length (cont.):

- Do not have your applications query the value of **MaxMsgLength** and then allocate a buffer of this size before calling **MQGET**; instead, allocate a buffer you consider large enough. If the **MQGET** fails, allocate a buffer guided by the size of the **DataLength** parameter.
- *The* **DataLength** *parameter of the* **MQGET** *call returns the length in bytes of the application data and any properties returned in the buffer you have provided, if a message handle* ***is not*** *specified in the* **MQGMO** *structure*.
- The **Buffer** parameter of the **MQPUT** call contains the application message data to be sent *and* any properties represented in the message data [payload].

IBM® MQ Message Properties

N O T E S

Message properties and message length (cont.):

- *When flowing to a queue manager that is earlier than Version 7.0 of the product, properties of the message*, except those in the message descriptor, *count toward the length of the message*. Therefore, you should either raise the value of the **MaxMsgLength** attribute of channels going to a system earlier than Version 7.0 as necessary to compensate for the fact that more data might be sent for each message. Alternatively, you can lower the queue or queue manager **MaxMsgLength**, so that the overall level of data being sent around the system remains the same.
- There is a length limit of 100 MiB for message properties, excluding the message descriptor or extension for each message.
- *The size of a property in its internal representation is the length of the name, plus the size of its value, plus some control data for the property. There is also some control data for the set of properties after one property is added to the message.* [The constitution of the 'control data' is ???.]

IBM® MQ Message Properties

N O T E S

Message properties and message length (cont.):

The maximum size of a message's data payload is determined by:

- The **MaxMsgLength** attribute of the queue manager.
- The **MaxMsgLength** attribute of the queue on which you are putting the message.
- The size of any message header added by **IBM® MQ** (including the dead-letter header, **MQDLH** and the distribution list header, **MQDH**)

The **MaxMsgLength** attribute of the *queue manager* holds the [maximum] size of message that the queue manager can process. This has a default value of 100 MiB for all **IBM® MQ** products at V6 or higher **(Really? See slide 8.8)**.

The **MaxMsgLength** attribute of a queue determines the maximum size of message that you can put on the queue. If you attempt to put a message with a size larger than the value of this attribute, your **MQPUT** call fails (RC2030). If you are putting a message on a remote queue, the maximum size of message that you can successfully put is determined by the **MaxMsgLength** attribute of the remote queue, [the remote queue manager], any intermediate transmission queues that the message is put on along the route to its destination, and finally of the channels used [to get there].

IBM® MQ Message Properties

N

O

T

E

S

Message properties and message length (cont.):

For an **MQPUT** operation, the size of the message must be smaller than or equal to the **MaxMsgLength** attribute of both the queue and the queue manager. The values of these attributes are independent, but you are advised to set the **MaxMsgLength** of the queue to a value less than or equal to that of the queue manager.

IBM® MQ adds header information to messages in the following circumstances:

- When you put a message on a remote queue, WebSphere MQ adds a transmission header structure (**MQXQH**) to the message. This structure includes the name of the destination queue and its owning queue manager.
- If **IBM® MQ** cannot deliver a message to a remote queue, it attempts to put the message on the dead-letter (undelivered-message) queue. It adds an **MQDLH** structure to the message. This structure includes the name of the destination queue and the reason that the message was put on the dead-letter queue.

IBM® MQ Message Properties

N O T E S

Message properties and message length (cont.):

- If you want to send a message to multiple destination queues, **IBM® MQ** adds an **MQDH** header to the message. This describes the data that is present in a message, belonging to a distribution list, on a transmission queue. Consider this when choosing an optimum value for the maximum message length.
- If the message is a segment or a message in a group, **IBM® MQ** might add an **MQMDE** (**but when and why is obfuscated**).

If your messages are of the maximum size allowed for these queues, the addition of these headers means that the put operation fails because the messages are now too big. To reduce the possibility of the put operations failing:

1. Make the size of your messages smaller than the **MaxMsgLength** attribute of the transmission and dead-letter queues. Allow at least the value of the **MQ_MSG_HEADER_LENGTH** constant [**4000**] (more for large distribution lists).
2. Make sure that the **MaxMsgLength** attribute of the dead-letter queue is set to the same as the **MaxMsgLength** of the queue manager that owns the dead-letter queue.

IBM® MQ Message Properties

N

O

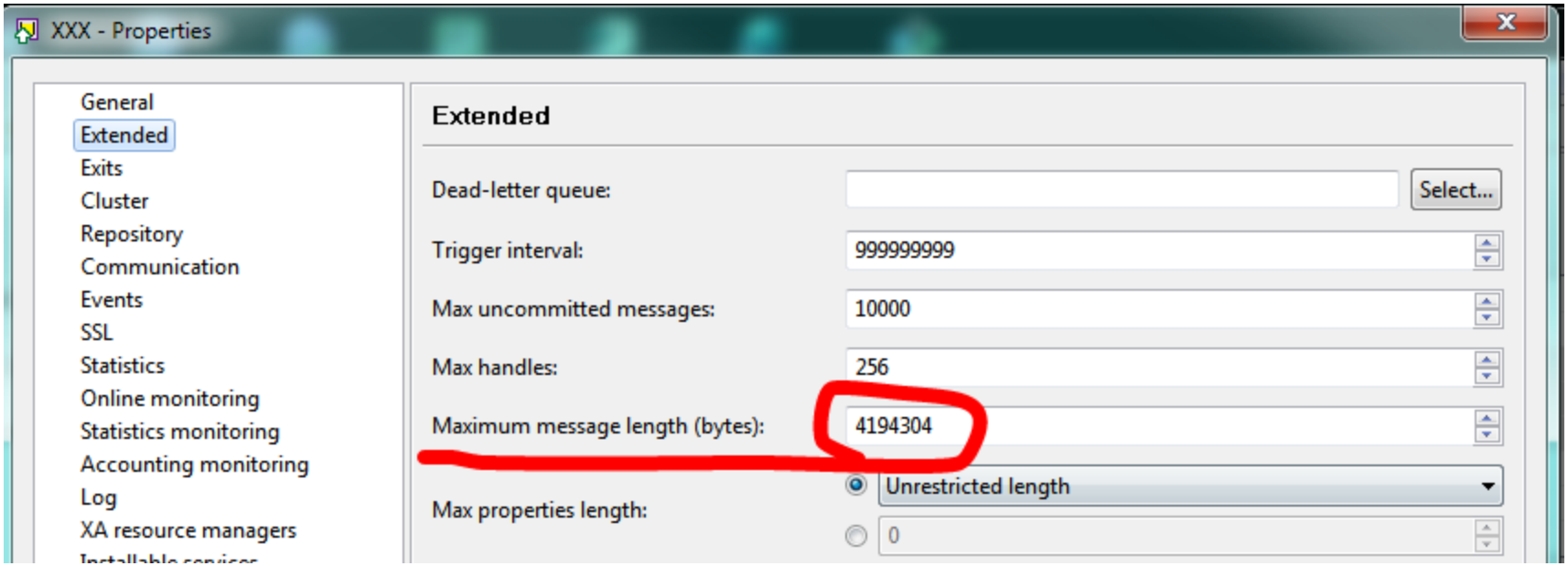
T

E

S

Message properties and message length (cont.):

The documented claim (**see slide 8.5**) that the default **MAXMSGL** value for a queue manager is 100 MiB. Really? Below is a just-created **IBM® MQ** queue manager "**XXX**". Doesn't look like 100 MiB.



IBM® MQ Message Properties (cont.)

Message property names:

- Must be a character string not exceeding MQ_MAX_PROPERTY_NAME_LENGTH (4095) characters in length.
- Must follow [naming] rules defined by the Java™ Language Specification for Java Identifiers.
- Must NOT contain:
  - Embedded nulls (although technically permitted)*
  - White space characters.
- Can be grouped using name subcomponents separated by a period (.), but…
  - The following prefixes are reserved for use by the product:
    - mcd
    - jms
    - usr
    - mq
    - sib
    - wmq
    - Root
    - Body
    - Properties
  - Use property synonyms instead.

IBM® MQ Message Properties

N O T E S

Message property names:

- A property name is a character string. Certain restrictions apply to its length and the set of characters that can be used.
- A property name is a case-sensitive character string, limited to +4095 characters unless otherwise restricted by the context. This limit is contained in the **MQ_MAX_PROPERTY_NAME_LENGTH** constant.
- If you exceed this maximum length when using a message property **MQI** call, the call fails with reason code **MQRC_PROPERTY_NAME_LENGTH_ERR**.
- Because there is no maximum property name length in **JMS**, it is possible for a **JMS** application to set a valid **JMS** property name that is not a valid **IBM® MQ** property name when stored in an **MQRFH2** structure.
- In this case, when parsed, only the first 4095 characters of the property name are used; all following characters are truncated. This could cause an application using selectors to fail to match a selection string, or to match a string when not expecting to, since more than one property might truncate to the same name. When a property name is truncated, **IBM® MQ** issues an error log message.

IBM® MQ Message Properties

N O T E S

Message property names (cont.):

- All property names must follow the rules defined by the **Java**™ Language Specification for Java Identifiers, with the exception that Unicode character U+002E (.) is permitted as part of the name - but not the start. The rules for Java Identifiers equate to those contained in the **JMS** specification for property names.
- White space characters and comparison operators are prohibited. Embedded nulls* are allowed in a property name but not recommended. If you use embedded nulls, this prevents the use of the **MQVS_NULL_TERMINATED** constant when used with the **MQCHARV** structure to specify variable length strings.
- Keep property names simple because applications can select messages based on the property names and the conversion between the character set of the name and of the selector might cause the selection to fail unexpectedly.

IBM® MQ Message Properties

N O T E S

Message property names (cont.):

- **IBM® MQ** property names use character U+002E (.) for logical grouping of properties. This divides up the namespace for properties. Properties with the following prefixes, in any mixture of lowercase or uppercase are reserved for use by the product:
  - mcd
  - jms
  - usr
  - mq
  - sib
  - wmq
  - Root
  - Body
  - Properties
- A good way to avoid name clashes is to ensure that all applications prefix their message properties with their Internet domain name. For example, if you are developing an application using domain name **ourcompany.com**, you could name all properties with the prefix **com.ourcompany**. This naming convention also allows for easy selection of properties; for example, an application can inquire on all message properties starting **com.ourcompany.%**.

IBM® MQ Message Properties

N O T E S

Property name restrictions:

- When you name a property, you must observe certain rules. The following restrictions apply to property names:
  1. A property must not begin with the following strings:
     - "**JMS**" - reserved for use by **IBM® MQ** classes for **JMS**.
     - "**usr.JMS**" - not valid.

  The only exceptions are the following properties providing synonyms for **JMS** properties:

Property	Synonym for
– JMSCorrelationID	Root .MQMD.CorrelId or jms.Cid
– JMSDeliveryMode	Root .MQMD.Persistence or jms.Dlv
– JMSDestination	jms.Dst
– JMSExpiration	Root .MQMD.Expiry or jms.Exp
– JMSMessageID	Root .MQMD.MsgId
– JMSPriority	Root .MQMD.Priority or jms.Pri
– JMSRedelivered	Root .MQMD.BackoutCount
– JMSReplyTo	Root .MQMD.ReplyToQ or Root .MQMD.ReplyToQMgr or jms.Rto
– JMSTimestamp	Root .MQMD.PutDate or Root .MQMD.PutTime or jms.Tms

IBM® MQ Message Properties

N O T E S

Property name restrictions (cont.):

Property	Synonym for
– JMSType	mcd.Type or mcd.Set or mcd.Fmt
– JMSXAppID	Root .MQMD.PutApplName
– JMSXDeliveryCount	Root .MQMD.BackoutCount
– JMSXGroupID	Root .MQMD.GroupId or jms.Gid
– JMSXGroupSeq	Root .MQMD.MsgSeqNumber or jms.Seq
– JMSXUserID	Root .MQMD.UserIdentifier

These synonyms allow an **MQI** application to access **JMS** properties in a similar fashion to **IBM® MQ** classes for **JMS** client application. Of these properties, only **JMSCorrelationID**, **JMSReplyTo**, **JMSType**, **JMSXGroupID**, and **JMSXGroupSeq** can be set using the **MQI**.

Note that the **JMS_IBM_*** properties available from within **IBM® MQ** classes for **JMS** are not available using the **MQI**. The fields that the **JMS_IBM_*** properties reference can be accessed in other ways by **MQI** applications.

2. A property must not be called, in any mixture of lower or uppercase, "**NULL**", "**TRUE**", "**FALSE**", "**NOT**", "**AND**", "**OR**", "**BETWEEN**", "**LIKE**", "**IN**", "**IS**" and "**ESCAPE**". These are the names of SQL keywords used in selection strings.
3. A property name beginning with "mq " in any mixture of lowercase or uppercase and not beginning "**mq_usr**" can contain only one "**.**" character (U+002E). Multiple "**.**" characters are not allowed in properties with those prefixes.

IBM® MQ Message Properties

N O T E S

Property name restrictions (cont.):

4. Two "**.**" characters must contain other characters in between; you cannot have an empty point in the hierarchy. Similarly a property name cannot end in a "**.**" character.
5. If an application sets the property "**a.b**" and then the property "**a.b.c**", it is unclear whether in the hierarchy "**b**" contains a value or another logical grouping. Such a hierarchy is "mixed content" and this is not supported. Setting a property that causes mixed content is not allowed.

- These restrictions are enforced by the validation mechanism as follows:
  - Property names are validated when setting a property using the **MQSETMP** – set message property call, if validation was requested when the message handle was created. If an attempt to validate a property is undertaken and fails due to an error in the specification of the property name, the completion code is **MQCC_FAILED** with reason:
    - **MQRC_PROPERTY_NAME_ERROR** for reasons 1-4 [above].
    - **MQRC_MIXED_CONTENT_NOT_ALLOWED** for reason 5.
  - The names of properties specified directly as **MQRFH2** elements are not guaranteed to be validated by the **MQPUT** call.

IBM® MQ Message Properties

N

O

T

E

S

Message descriptor fields as properties:

- Most message descriptor fields can be treated as properties. The property name is constructed by adding a prefix to the message descriptor field's name.
- If an **MQI** application wants to identify a message property contained in a message descriptor field, for example, in a selector string or using the message property APIs, use the following syntax:

Property name	Message descriptor field
– Root.MQMD.<Field>	<Field>

- Specify <Field> with the same case as for the **MQMD** structure fields in the 'C' language declaration. For example, the property name **'Root.MQMD.AccountingToken'** accesses the **'AccountingToken'** field of the message descriptor.
- The **StrucId** and **Version** fields of the message descriptor are not accessible using the syntax shown [here].

IBM® MQ Message Properties

N O T E S

Message descriptor fields as properties (cont.):

- Message descriptor fields are never represented in an **MQRFH2** header as [it is] for other properties.
- If the message data starts with an **MQMDE** that is honored by the queue manager, the **MQMDE** fields can be accessed using the **Root.MQMD.<Field>** notation described. In this case the **MQMDE** fields are treated logically as a part of the **MQMD** from a properties perspective.

IBM® MQ Message Properties (cont.)

Message property values:

- Can be a boolean, a byte string, a character string, an integer or a floating-point number.
- Must be one of the following data types:
  - MQBOOL
  - MQBYTE[ ]
  - MQCHAR[ ]
  - MQFLOAT32
  - MQFLOAT64
  - MQINT8
  - MQINT16
  - MQINT32
  - MQINT64
- Can be undefined – a NULL property.

IBM® MQ Message Properties

N O T E S

Message property data types and values:

- A property can be a boolean, a byte string, a character string, an integer or a floating-point number. The property can store any valid value in the range of the data type unless otherwise restricted by the context.
- The data type of a property value must be one of the following values:
  - MQBOOL
  - MQBYTE[ ]
  - MQCHAR[ ]
  - MQFLOAT32
  - MQFLOAT64
  - MQINT8
  - MQINT16
  - MQINT32
  - MQINT64
- A property can exist but have no defined value; it is a null property. A null property is different from a byte property (**MQBYTE[ ]**) or character string property (**MQCHAR[ ]**) in that it has a defined but empty value, that is, one with a zero-length value.
- Byte string is not a valid property data type in **JMS** or **XMS**. You are advised not to use byte string properties in the **<usr>** folder.

IBM® MQ Message Selectors

A **message selector** is a variable-length SQL query string defined within a **MQCHARV** data structure which is used by an application to register its interest (with the queue manager) in only those messages having properties that satisfy the SQL query.

- A message selector is supplied via the ***SelectionString*** field of the **MQOD** or **MQSD** data structure provided in a **MQOPEN** or **MQSUB** MQI function call.
  - MQOD ***ObjectType*** field must be **MQOT_Q**.
  - MQOPEN ***Options*** field must be either **MQOO_BROWSE** or **MQOO_INPUT_***.
- Selector string is fixed while the queue / subscription object remains open. The object handle must be closed (**MQCLOSE**) and re-opened in order to specify a new selection string.

IBM® MQ Message Selectors (cont.)

Message selector syntax and behavior can be complex depending on the query. There is no *N*th generation SQL optimization being performed by the queue manager. So, in general…

- The deeper the queue, the simpler the SQL query.
- Parenthesize expressions.
- Evaluation precedence is left to right – expressions most likely to resolve selection should lead the SQL query.

Selectors are implemented solely by the message retrieving application. Imagine… a world where selectors could be defined in the messaging infrastructure itself by having alias queue objects provide an associated selector attribute value to its base (local) queue. Intrigued??

- You are invited to re-submit RFE #41564 (declined 2015.0421) at:

https://www.ibm.com/developerworks/rfe/execute?use_case=viewRfe&CR_ID=41564

IBM® MQ Message Selectors

N O T E S

Selection behavior:

- The fields in an **MQMDE** structure are considered to be the message properties for the corresponding message descriptor properties if the **MQMD**:
  - Has format **MQFMT_MD_EXTENSION**
  - Is immediately followed by a valid **MQMDE** structure
  - Is version one or contains the default version two fields only
- It is possible for a selection string to resolve to either **TRUE** or **FALSE** before any matching against message properties takes place. For example, it might be the case if the selection string is set to "**TRUE <> FALSE**". Such early evaluation is guaranteed to take place only when there are no message property references in the selection string.
- If a selection string resolves to **TRUE** before any message properties are considered, all messages published to the topic subscribed to by the consumer are delivered. If a selection string resolves to **FALSE** before any message properties are considered, a **MQRC_SELECTOR_ALWAYS_FALSE** reason code, and completion code **MQCC_FAILED** are returned on the function call that presented the selector.

IBM® MQ Message Selectors

NOTES

Selection behavior (cont.):

- Even if a message contains no message properties (other than header properties) then it can still be eligible for selection. If a selection string references a message property that does not exist, this property is assumed to have the value of **NULL** or **'Unknown**'.
- For example, a message might still satisfy a selection string like '**Color IS NULL**', where '**Color**' does not exist as a message property in the message.
- Selection can be performed only on the properties that are associated with a message, not the message [data payload] itself, unless an extended message selection provider is available. Selection can be performed on the message payload only if an extended message selection provider is available.
- Each message property has a type associated with it. When you perform a selection, you must ensure that the values used in expressions to test message properties are of the correct type. If a type mismatch occurs, the expression in question resolves to **FALSE**.

IBM® MQ Message Selectors

N O T E S

Selection behavior (cont.):

- It is your responsibility to ensure that the selection string and message properties use compatible types.
- Selection criteria continue to be applied on behalf of inactive durable subscribers, so that only messages that match the selection string that was originally supplied are kept.
- Selection strings are non-alterable when a durable subscription is resumed with alter (**MQSO_ALTER**). If a different selection string is presented when a durable subscriber resumes activity, then **MQRC_SELECTOR_NOT_ALTERABLE** is returned to the application.
- Applications receive a return code of **MQRC_NO_MSG_AVAILABLE** if there is no message on a queue that meets the selection criteria.
- If an application has specified a selection string containing property values, then only those messages that contain matching properties are eligible for selection. For example, a subscriber specifies a selection string of "**a = 3**" and a message is published containing no properties, or properties where '**a**' does not exist or is not equal to 3. The subscriber does not receive that message to its destination queue.

IBM® MQ Message Selectors

N O T E S

Messaging performance:

- Selecting messages from a queue requires **IBM® MQ** to sequentially inspect each message on the queue. Messages are inspected until a message is found that matches the selection criteria or there are no more messages to examine. Therefore, messaging performance suffers if message selection is used on deep queues.
- To optimize message selection on deep queues when selection is based on **JMSCorrelationID** or **JMSMessageID**, use a selection string of the form **JMSCorrelationID = ...** or **JMSMessageID = ...** and reference *only the one* property.
- This method offers a significant improvement in performance for selection on **JMSCorrelationID** and offers a marginal performance improvement for **JMSMessageID**.

IBM® MQ Message Selectors

N O T E S

Using complex selectors:

- Selectors can contain many components, for example:
  - a and b or c and d or e and f or g and h or i and j ... or y and z
- Use of such complex selectors can have serious performance implications and excessive resource requirements. As such, **IBM® MQ** will protect the system by failing to process overly complex selectors that could result in a system resource shortage.
- Protection can occur on selection strings that contain more than 100 tests, or when **IBM® MQ** detects that the limit on the size of the operating system stack is being approached. You should thoroughly try and test the use of selection strings with many components, on the appropriate platforms, to ensure that the protection limits are not reached.
- The performance and complexity of selectors can be improved by simplifying them using additional parenthesis to combine components. For example:
  - (a and b or c and d) or (e and f or g and h) or (i and j) ...

IBM® MQ Message Selectors

N O T E S

Selection string rules and restrictions:

- The following is a list of rules about how selection strings are interpreted:
  1. Message selection for publish/subscribe messaging occurs on the message as sent by the publisher.
  2. Equivalence is tested using a single equals character; for example, **"a = b"** is correct, whereas **"a == b"** is incorrect.
  3. An operator used by many programming languages to represent "not equal to" is **'!='**. This representation is not a valid synonym for **'<>'**; for example, **"a <> b"** is valid, whereas **"a != b"** is not valid.
  4. Single quotation marks are recognized only if the ' (U+0027) character is used. Similarly, double quotation marks, valid only when used to enclose byte strings, must use the " (U+0022) character.
  5. The symbols **'&'**, **'&&'**, **'|'** and **'||'** are not synonyms for logical conjunction / disjunction; for example, **"a && b"** must be specified as **"a AND b"**.
  6. The wildcard characters **'*'** and **'?'** are not synonyms for **'%'** and **'_'**.

IBM® MQ Message Selectors

N O T E S

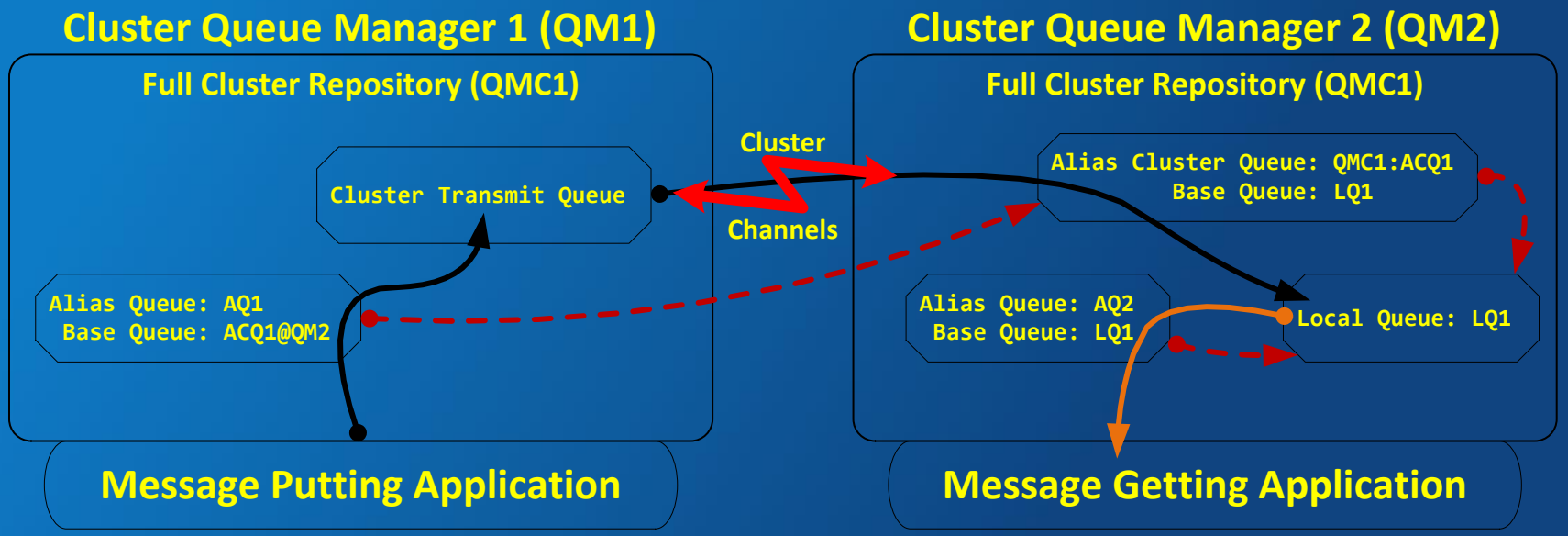
Selection string rules and restrictions (cont.):

- Rules about how selection strings are interpreted (cont.):
  7. Selectors containing compound expressions such as **"20 < b < 30"** are not valid. The parser evaluates operators that have the same precedence from left to right. The example would therefore become **"(20 < b) < 30"**, which does not make sense. Instead the expression must be written as **"(b > 20) AND (b < 30)"**.
  8. Byte strings must be enclosed in double quotation marks; *if single quotation marks are used, the byte string is taken to be a string literal*. The number of characters (not the number that the characters represent) following an **'0x'** must be a multiple of two.
  9. The keyword '**IS'** is not a synonym for the equals character. Thus the selection strings **"a IS 3"** and **"b IS 'red'"** are not valid. The '**IS'** keyword exists only to support '**IS NULL'** and '**IS NOT NULL'** cases.
- The selector string is fixed while the queue / subscription object remains open. The object handle must be closed (**MQCLOSE**) and re-opened in order to specify a new selection string. Alternatively, instantiate multiple object handles to a queue, each with its own selector string (or perhaps none).

Re-enacting the Messaging Infrastructure CS

We're not doing anything fancy here –
Just standard auto-segmentation of messages > 4 MiB.

Consider: A simple two clustered queue manager setup.



Re-enacting the Messaging Infrastructure CS (cont.)

All default MQ system objects are unaltered.
All defined MQ objects take the default maximum message length.

- The message originating queue manager MAXMSGL attribute set to 4 MiB (Dflt).
- The SYSTEM.CLUSTER.TRANSMISSION.QUEUE MAXMSGL attribute set to 4 MiB (Dflt).
- Destination queue manager MAXMSGL attribute set to 4 MiB (Dflt).
- Destination queue MAXMSGL attribute set to 4 MiB (Dflt).
- All cluster receiver / sender channels MAXMSGL attribute set to 0 (4 MiB).
- Using a message handle in place of a MQMD.

Setup of the message **putting** application…

- Initialize message property and handle objects & data structures:

```
MQHMSG     MsgHandle;
MQCMHO     CrtMsgHndOpts = { MQCMHO_DEFAULT };
MQDMHO     DltMsgHndOpts = { MQDMHO_DEFAULT };
MQSMPO     SetMsgPrpOpts = { MQSMPO_DEFAULT };
MQPD       MsgPropDesc   = { MQPD_DEFAULT };
```

Re-enacting the Messaging Infrastructure CS (cont.)

Setup of the message putting application… (cont.)

- Initialize message property and handle objects & data structures:

```
CrtMsgHndOpts.Options     = MQCMHO_DEFAULT_VALIDATION;
DltMsgHndOpts.Options     = MQDMHO_NONE;

SetMsgPrpOpts.Options     = MQSMPO_SET_FIRST;

MsgPropDesc.CopyOptions   = MQCOPY_NONE*;

MsgPropType               = MQTYPE_STRING;
PutMsgOpts.Action         = MQACTP_NEW;

PutMsgOpts.Options        = MQPMO_NEW_MSG_ID | MQPMO_NEW_CORREL_ID |
                            MQPMO_NO_SYNCPOINT | MQPMO_SYNC_RESPONSE |
                            MQPMO_FAIL_IF_QUIESCING;
```

- Instantiate object handles:
  - `MQCONN (QMgrName,   &QMgrHandle);`
  - `MQCRTMH(QMgrHandle, CrtMsgHndOpts, MsgHandle);`

Re-enacting the Messaging Infrastructure CS (cont.)

Setup of the message putting application… (cont.)

- Four message properties are defined:
  - FileName: Name of the file whose contents comprise the data payload
  - MsgLength: Sizing buffers & segmented message group logic
  - MsgGrpId: Associate segmented message with its 'reference' message
  - MsgProps: Multiple additional message properties stand-in

```
MQSETMP(QMgrHandle, MsgHandle, SetMsgPrpOpts, FileName.Name,
        MsgPropDesc, MsgPropType, FileName.Length, FileName.Value);

MQSETMP(QMgrHandle, MsgHandle, SetMsgPrpOpts, MsgLength.Name,
        MsgPropDesc, MsgPropType, MsgLength.Length, MsgLength.Value);

MQSETMP(QMgrHandle, MsgHandle, SetMsgPrpOpts, MsgGrpId.Name,
        MsgPropDesc, MsgPropType, MsgGrpId.Length, MsgGrpId.Value);

MQSETMP(QMgrHandle, MsgHandle, SetMsgPrpOpts, MsgProps.Name,
        MsgPropDesc, MsgPropType, MsgProps.Length, MsgProps.Value);
```

- Messages are put using the MQI function call MQPUT1.**

Setup of the message putting application

N O T E S

Message property data components are defined by a data structure thusly:

```
struct tagMessageProperty {
  MQCHARV  Name;
  PMQCHAR  Value;
  MQLONG   Length;
  MQLONG   MaxLength;
};

/*************************************************************
For reference, an MQCHARV data structure is defined by MQ thusly:

typedef struct tagMQCHARV MQCHARV;

struct tagMQCHARV {
  MQPTR   VSPtr;        /* Address of variable length string */
  MQLONG  VSOffset;     /* Offset of variable length string  */
  MQLONG  VSBufSize;    /* Size of buffer                    */
  MQLONG  VSLength;     /* Length of variable length string  */
  MQLONG  VSCCSID;      /* CCSID of variable length string   */
};
*************************************************************/
```

*This is the only non-default attribute value of the message property descriptor being set; all other attribute values are defaulted (MQPD.Context = MQPD_NO_CONTEXT, MQPD.Options = MQPD_NONE, MQPD.Support = MQPD_SUPPORT_OPTIONAL).

**MQPUT1 is used to replicate the original messaging application.

Setup of the message putting application

N

O

T

E

S

An array of message property data structures would normally be used but for clarity (and since there are only four message properties), in the re-enactment we will define each message property separately, thusly:

```
typedef struct tagMessageProperty FileName = {
  { "FileName",       0,  MQVS_USE_VSLENGTH,  MQVS_NULL_TERMINATED,  MQCCSI_APPL },
  NULL,
  0,
  FileNameMaxLen
};

typedef struct tagMessageProperty MsgLength = {
  { "MessageLength",  0,  MQVS_USE_VSLENGTH,  MQVS_NULL_TERMINATED,  MQCCSI_APPL },
  NULL,
  0,
  MsgLengthMaxLen
};

typedef struct tagMessageProperty MsgGrpId = {
  { "MessageGroupId", 0,  MQVS_USE_VSLENGTH,  MQVS_NULL_TERMINATED,  MQCCSI_APPL },
  NULL,
  0,
  MsgGrpIdMaxLen
};

typedef struct tagMessageProperty MsgProps = {
  { "MessageProps",   0,  MQVS_USE_VSLENGTH,  MQVS_NULL_TERMINATED,  MQCCSI_APPL },
  NULL,
  0,
  KIB
};
```

Solution specific code continues on slide 30.1.

Re-enacting the Messaging Infrastructure CS (cont.)

Setup of the message **getting** application…

- Initialize message property and handle objects & data structures:

```
MQHMSG      MsgHandle;
MQCMHO      CrtMsgHndOpts = { MQCMHO_DEFAULT };
MQDMHO      DltMsgHndOpts = { MQDMHO_DEFAULT };
MQIMPO      InqMsgPrpOpts = { MQIMPO_DEFAULT };
MQPD        MsgPropDesc   = { MQPD_DEFAULT };
```

- The message getting application must perform message property inquiries using the **MQINQMP** MQI function call.
- Initialize message property inquiry data structures:

```
InqMsgPrpOpts.Options =  MQIMPO_CONVERT_VALUE |
                         MQIMPO_CONVERT_TYPE |
                         MQIMPO_INQ_FIRST;
```

Re-enacting the Messaging Infrastructure CS (cont.)

Setup of the message getting application… (cont.)

- Initialize message property and handle objects options:

```
CrtMsgHndOpts.Options     = MQCMHO_DEFAULT_VALIDATION;
DltMsgHndOpts.Options     = MQDMHO_NONE;

MsgPropDesc.CopyOptions   = MQCOPY_NONE;

GetMsgOpts.Options        = MQGMO_WAIT | MQGMO_CONVERT |
                            MQGMO_NO_SYNCPOINT |
                            MQGMO_PROPERTIES_IN_HANDLE |
                            MQGMO_COMPLETE_MESSAGE |
                            MQGMO_FAIL_IF_QUIESCING;

MsgPropType               = MQTYPE_STRING;
```

- Instantiate object handles:
  - `MQCONN (QMgrName,    &QMgrHandle);`
  - `MQCRTMH(QMgrHandle,  CrtMsgHndOpts, MsgHandle);`
  - `MQOPEN (QMgrHandle,  &QueObjDesc,`
    `QueOpenOpts, &QueObjHnd);`

Re-enacting the Messaging Infrastructure CS (cont.)

Running the re-enactment:

- Several text files of varying size are read by the getting application which packages the file's contents into an MQ message's data payload.
- The assembled message is put onto an alias queue whose base queue is a cluster alias queue defined on the destination queue manager. A MQMD is provided as the MQPUT's message descriptor argument.

What is observed:

1. The queue manager correctly determines the precise conditions under which a message must be segmented.
2. The queue manager converts the message's properties into a MQRFH2 data structure which it prepends onto the first message segment.
3. But-- the queue manager sizes the first message segment precisely at its MAXMSGL value of 4 MiB. Prepending the MQRFH2 data structure to the first message segment causes its size to now exceed 4 MiB!

Re-enacting the Messaging Infrastructure CS (cont.)

Running the re-enactment (cont.):

Then, when typically a MQMD is provided as the MQPUT call's message descriptor argument, the following is observed:

4. The now too-large message segment is successfully put and takes up residence on the queue manager's cluster transmission queue.
5. The destination queue manager's cluster receiver channel MCA fails in its attempt to retrieve the first message segment.
6. The cluster receiver channel ends abnormally, throwing:

   ```
   AMQ9259: Connection timed out from host '192.168.1.121'.
   ```

7. The 'fat' message segment plugs the message-originating queue manager's cluster transmission queue and no subsequently queued-up messages can be transmitted-- technically, it's a "poison" message.
8. At a minimum, the offending message segments must be manually deleted from the cluster transmission queue in order to 'free' messages queued up after the auto-segmentation event.

Running the re-enactment

N O T E S

```
Error thrown by the destination queue manager (QM2):
================================================================================
09/21/15 09:36:44 PM - Process(5014.1) User(mqm) Program(amqzxma0)
                       Host(sol10vm1) Installation(Installation1)
                       VRMF(7.5.0.4) QMgr(QM2)


The select() [TIMEOUT] 180 seconds call timed out. Check to see why data was
not received in the expected time. Correct the problem. Reconnect the channel,
or wait for a retrying channel to reconnect itself.
----- amqccita.c : 4240 ------------------------------------------------------
09/21/15 05:06:57 PM - Process(5028.51) User(mqm) Program(amqrmppa)
                       Host(sol10vm1) Installation(Installation1)
                       VRMF(7.5.0.4) QMgr(QM2)

AMQ9999: Channel 'TO_QM2' to host 'genesis (192.168.1.121)' ended abnormally.

EXPLANATION:
The channel program running under process ID 5028 for channel 'TO_QM2' ended
abnormally. The host name is 'genesis (192.168.1.121)'; in some cases the host
name cannot be determined and so is shown as '????'.
================================================================================
```

*__WARNING__: If you use **MQ Explorer** to nuke the cluster transmission queue's messages (it's a nuke because all messages get whacked, not just the offending message), it is possible for some of the poison message's legal segments to be transmitted to their destination queue where they likely will languish forever or until they are manually removed from the destination queue.

What Didn't Work (as Expected)?

1. The message-originating queue manager correctly determines that the total size of the message plus its properties will exceed the queue manager MAXMSGL attribute value. This is the only action that worked.
2. But… when the queue manager calculates the location in the message's data payload where the first segment is to be cleaved, the QM fails to take into account the length of the message's properties which are subsequently assembled into an MQRHF2 header that is prepended to the data payload of the first message segment.
3. This causes the first message segment to exceed the MAXMSGL value by an amount equal to the size of the MQRHF2 header. Amazingly, if a message header descriptor (MQMD) is provided to the MQPUT call, the message segment is successfully placed onto the transmission queue.
4. The message retrieval process executed (at the tcp/ip socket layer) by the receiver channel's MCA fails continuously and ultimately times out after three minutes (180 seconds) with the destination queue manager throwing the errors shown in the previous slide.

What Didn't Work (as Expected)? (cont.)

5. The error reported in the destination queue manager's (DQM) log is vague.* No FDC files are generated by either the originating or destination queue managers. Running a trace is the only way to expose the true nature of the problem.
6. The most notorious failure in this scenario is the message-originating queue manager's failure to recognize the message segment is too large and putting it on the transmission queue anyway. This behavior may be ameliorated by setting the MQPUT message descriptor parameter to NULL.
7. Failure of the DQM SCRC's MCA to retrieve the first message segment from the message-originating queue manager's SCTQ results in the DQM's SCRC being stopped. Furthermore, it is not possible to restart the channel until the offending message segment is removed from the SCTQ.
8. If a null message descriptor is provided to the MQPUT call (instead of a MQMD), then the message put does fail, throwing either a RC2030 error (**message too large for queue**) or a RC2031 error (**message too large for queue manager**) *even though MQMF_SEGMENTATION_ALLOWED was specified*!

Running the re-enactment

N

O

T

E

S

*Error thrown by the originating queue manager (QM1) is even more vague:

```
================================================================================
9/21/2015 17:12:44 - Process(13952.11) User(GregBrown) Program(amqrmppa.exe)
                     Host(GENESIS) Installation(Installation1)
                     VRMF(7.5.0.4) QMgr(QM1)

AMQ9206: Error sending data to host 192.168.1.172 (192.168.1.172)(1422).

EXPLANATION:
An error occurred sending data over TCP/IP to 192.168.1.172
(192.168.1.172)(1422). This may be due to a communications failure.
ACTION:
The return code from the TCP/IP(send) call was 10053 X('2745'). Record these
values and tell your systems administrator.
----- amqccita.c : 3024 ------------------------------------------------------
9/21/2015 17:12:44 - Process(13952.11) User(GregBrown) Program(amqrmppa.exe)
                     Host(GENESIS) Installation(Installation1)
                     VRMF(7.5.0.4) QMgr(QM1)

AMQ9999: Channel 'TO_QM2' to host '192.168.1.172 (1422)' ended abnormally.

EXPLANATION:
The channel program running under process ID 13952(12748) for channel 'TO_QM2'
ended abnormally. The host name is '192.168.1.172 (1422)'; in some cases the
host name cannot be determined and so is shown as '????'.
================================================================================
```

Yes, the **SCTQ's **USEDLQ** attribute value is set to **'YES'** (the default).

What Other Scenarios Were Tried?

1. Set the originating queue manager's (OQM) defined cluster sender channel (DCSC) MAXMSGL and the destination queue manager's (DQM) defined cluster receiver channel (DCRC) MAXMSGL to 4200000:

   Result: Failure [This was the official 'work-around' given by IBM.]

2. Set the OQM SCTQ MAXMSGL to 4200000:

   Result: Failure

3. Set the DQM MAXMSGL to 4200000:

   Result: Failure

4. Reset all of the altered MAXMSGL attributes back to 4194304 and set the OQM MAXMSGL to 4100000:

   Result: Failure

5. Set the OQM MAXMSGL at default 4194304:
   i. Set the OQM SCTQ MAXMSGL to 4200000
   ii. Set the OQM DCSC MAXMSGL to 4200000
   iii. Set the DQM DCRC MAXMSGL to 4200000
   iv. Set the DQM and destination queue MAXMSGL to 4200000

   Result: Failure!

Zombie Messages Live to Raise Hell!

Let's go up to the lab and check out what's on the slab.

A Technical Implementation Solution Pattern

Problem Statement: Given that queue manager mediated auto-segmentation of large messages does not work with messages that have properties defined on them, what solution options are there?

1. Promulgate an IT governance directive prohibiting the use of the **IBM® MQ** message property feature anywhere in the application domain.
2. Promulgate an IT governance directive requiring the **MAXMSGL** attribute value for all **IBM® MQ** objects involved in enabling messaging services in the application domain be set to **104857600** bytes.
3. Promulgate an IT governance directive requiring developers of message enabled applications to adopt a technical implementation convention whose goal is to render large messages with properties defined on them transparent to a regrettably fickle messaging infrastructure.

The choice naturally will be directed by any number of factors. Typically the goal is to provide IT the most flexible options with minimal organizational impact. The solution presented here ultimately is just another option that, like all other options, has its own advantages and disadvantages.

A Technical Implementation Solution Pattern (cont.)

Main Advantages of the solution pattern:

1. Solution implementation is (almost) completely transparent to IT operations and administrative domains.
2. Solution implementation is fairly straight-forward for the developer.
3. Disadvantage #2 can be easily ameliorated by lowering the solution's message segmentation logic threshold.

Main Disadvantage of the solution pattern:

1. Theoretically, in order to assure that the solution would work with any configuration of an **IBM® MQ** messaging infrastructure, the MAXMSGL attribute value of every participating **IBM® MQ** object in the message transmission path would have to be known in advance by the putting application. This is, at best, a difficult proposition. *Consequently…*
2. A significant assumption is made: The message segmentation threshold in the putting application that triggers the solution logic is not larger than the smallest MAXMSGL value of any **IBM® MQ** object encountered in the course of message transmission.

A Technical Implementation Solution Pattern (cont.)

Keep in mind:

- A corollary assumption *of acceptably low risk* is that all **IBM® MQ** objects comprising a messaging infrastructure have a MAXMSGL value no lower than their default value [this may call for its own IT governance directive]. To the extent this assumption is (quite likely) true, then 4194304 bytes (4 MiB) is the magic threshold value above which the solution should not go.
- The solution logic threshold value is calculated as the sum of the length of the message data payload ***plus*** the collective length of the message's properties. If this value exceeds 4 MiB, then the solution logic is invoked.
- The solution logic is functionally benign when invoked in cases where auto-segmentation of a message does not in fact occur. This will happen on occasion where the total length of any particular message's properties is provided by a universally applicable maximum properties length value but that, when added to the length of a message's data payload, results in a calculated threshold value that is greater than 4MiB when in fact the true threshold value is less than 4 MiB (the true collective length of a message's properties must always be <= the maximum properties length value).

A Technical Implementation Solution Pattern (cont.)

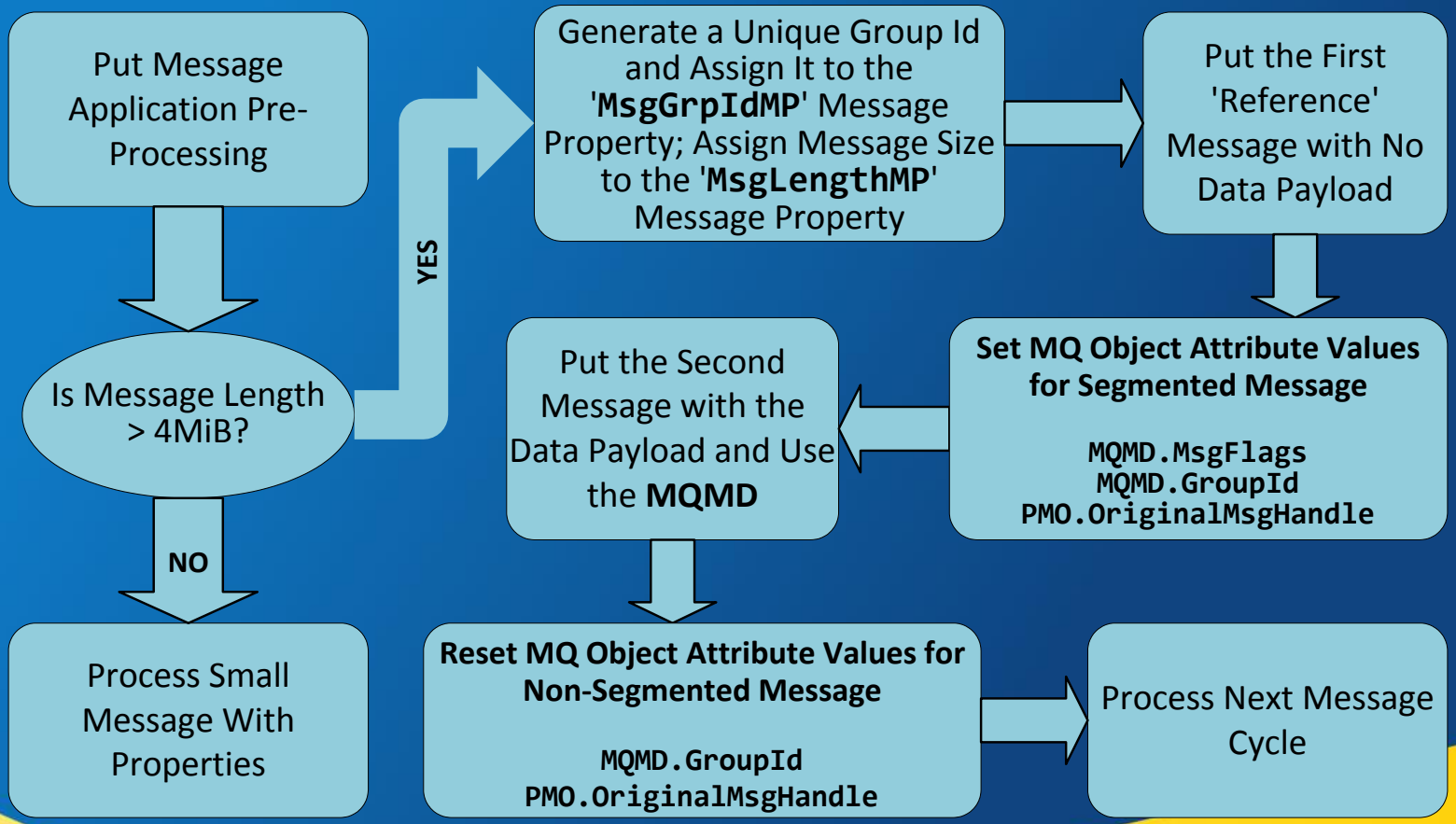
Message putting application pattern structure:

For the purpose of illustrating the technical implementation pattern, the demonstration application employs a simple process loop pattern:

- Application initialization
- Loop until exit condition:
  - Receive a file name
    - Acquire file size
    - Allocate message buffer memory
  - Assemble and set the MessageSize and FileName message properties
  - If the message data payload length (the file size) plus the max properties length value does not exceed 4 MiB, then
    - Put the message
  - Else
    1) Assemble and set the MessageGroupId message property
    2) Set variables supporting putting of a message that will be auto-segmented
    3) Put the (reference) message
    4) Put the (data) message
    5) Reset variables set in step 2
  - Continue

A Technical Implementation Solution Pattern (cont.)

Solution Enhancements to the Message Putting Application



A Technical Implementation Solution Pattern (cont.)

Code enhancements to the message putting application:

- Two message properties that absolutely must be included in a segmented message's reference message properties pool are:
  1. The total size of the segmented message to come (MP likely already there)
  2. The segmented message's assigned MQMD.GroupId (MP likely not already there)
- You are strongly encouraged to use strings to represent message property values. This will eliminate potential conversion issues.
- A more precise calculation of the length of a message's properties:
  - Calculate a constant whose value is the size of a declared but uninitialized message property data structure times the number of occurrences, and add to it the sum of the lengths of all message property names.
  - For every message, sum the length of all message property values and add that to the constant previously calculated. Then consider guidance provided in slides 6.2, 8.4 and 8.7 (among others), and add an appropriate fudge factor.
  - Be aware of potential issues with DBCS representation.

A Technical Implementation Solution Pattern (cont.)

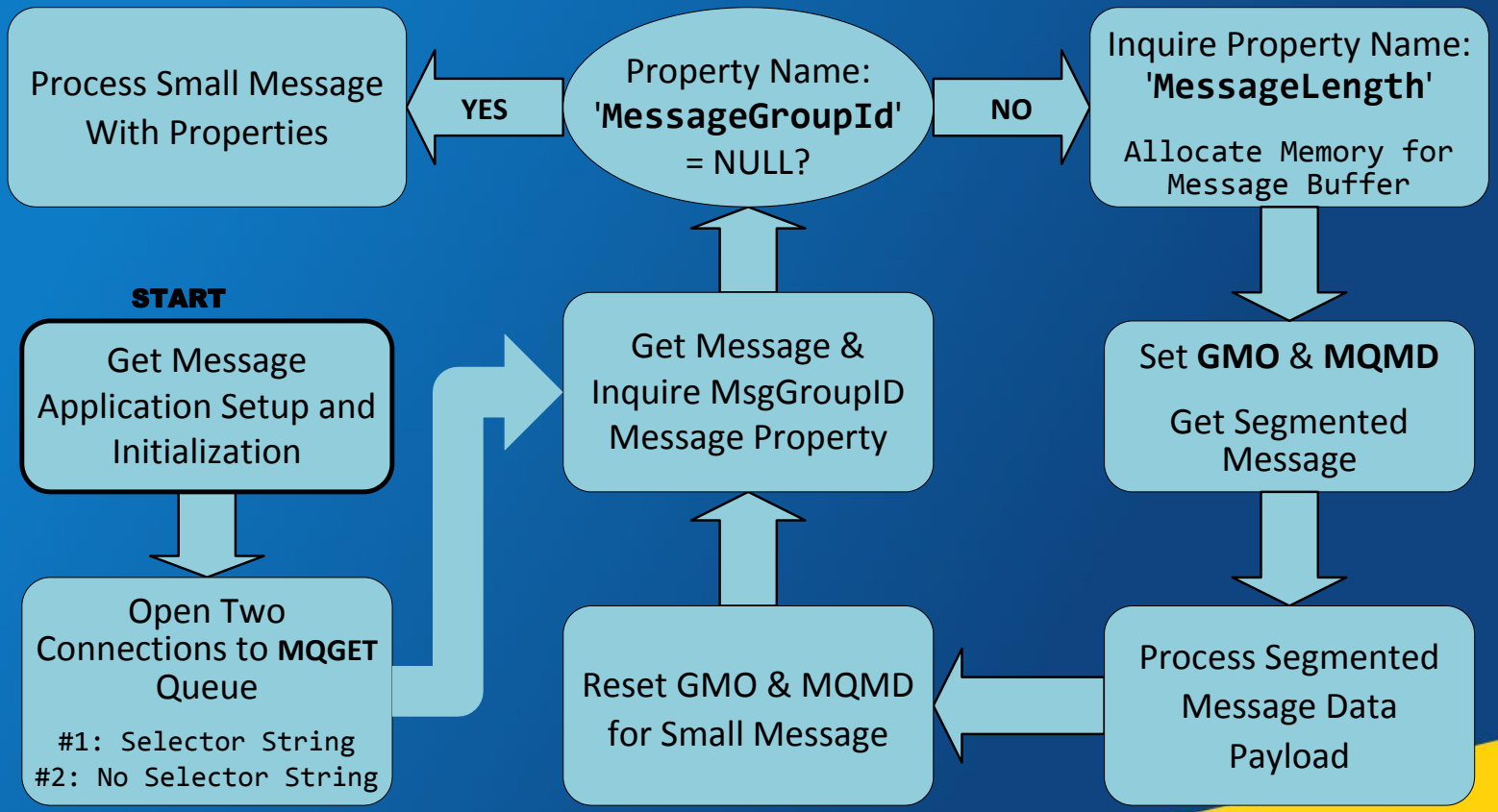
Message getting application pattern structure:

For the purpose of illustrating the technical implementation pattern, the demonstration application employs a simple process loop pattern:

- Application initialization
  - Instantiate two object handles to the get queue:
    1) The first object handle has no selector string defined to it
    2) The second object handle has an application-specific selector string defined to it
- Loop until exit condition:
  - Get a message (using first object handle)
  - Inquire MP: 'MessageGroupId' == NULL?
  - **YES:** Process message
  - **NO:** This is a reference message for a segmented message
    - Inquire MP: 'MessageSize' ; allocate message buffer memory
    - Initialize MQMD and GMO data structures for getting the segmented message
    - Get the segmented message (using second object handle)
    - Process message data payload
    - Initialize MQMD and GMO data structures for getting the next (regular) message
  - Continue

A Technical Implementation Solution Pattern (cont.)

Solution Enhancements to the Message Getting Application



A Technical Implementation Solution Pattern (cont.)

Code enhancements to the message getting application:

- The salient functional enhancements which enable the message getting application to process the 'reference' and 'data' messages that comprise a segmented message group consist of:

1. Instantiating two object handles to the MQGET queue:
   i. The first queue object handle will have no selector and will be used to get auto-segmented data payload messages.
   ii. The second queue object handle will have a selector string defined on it and will be used to get messages (< 4MiB) that have properties defined on them.
2. Incorporating two additional solution-specific message properties:
   i. The first message property will supply the size of the message data payload.
   ii. The second message property will supply the message group ID that is assigned by the message putting application to the segmented data payload message's **MQMD.GroupId** property.

Solution Logic

N

O

T

E

S

Please refer to the 'putwmp.c' and 'getwmp.c' source files which are authoritative exemplars for code enhancements required to implement the solution.

Note that the 'putwmp.c' application incorporates additional logic required to demonstrate failure of queue manager mediated auto-segmentation of messages that have properties defined on them.

Some may have preferred such demonstration be presented via a separate process in order to keep solution-specific logic as transparent as possible. In hindsight I wish I had done just that. In particular the MQPUT1 logic is obtusely dense. If additional clarity is desired, please feel free to email me with any questions you may have (grebrown@caci.com).

What We Covered:

- Learned the background history on problem discovery and the subsequent interaction with L2 / L3 support.
- Reviewed queue manager mediated auto-segmentation of large messages.
- Reviewed message properties (and selectors).
- Presented a description of the fault-generating environment.
- Investigated the underlying mechanics of the problem.
- Considered a technical implementation solution pattern.
  - Code enhancements to the message putting application
  - Code enhancements to the message getting application

Exit Stage Right

THANK YOU!

Questions & Answers