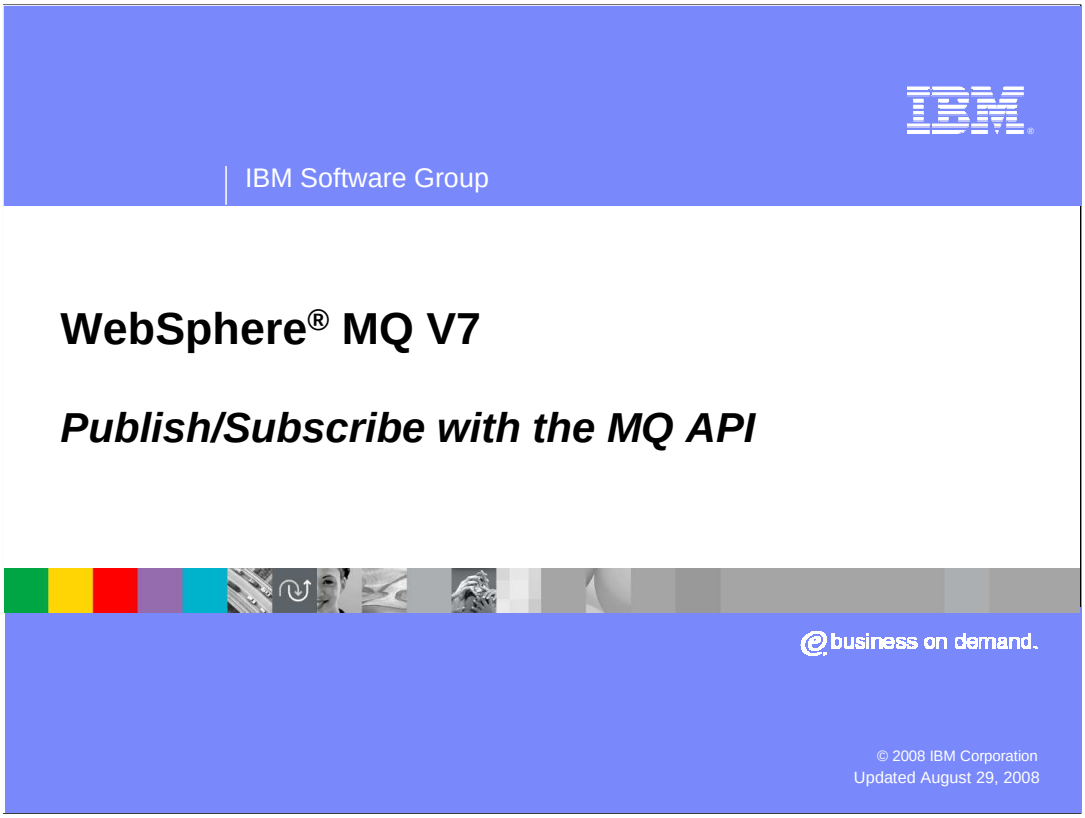


This presentation covers the MQ API (or MQI) changes in WebSphere MQ version 7.0 that cover Publish and Subscribe.

This extends the support for publish subscribe that is available in JMS to users of the MQI with straight forward verbs and options without using the broker.

These features are also available for MQ on z/OS®.

This unit assumes a reasonable understanding of the existing WebSphere MQ API for putting and getting messages to and from queues.

This unit also assumes a reasonable understanding of Pub/Sub and the administration of Topic Objects by MQ Explorer. This understanding can be gained from other presentations in this series.

Unit objectives

After you complete this unit, you should be able to:

- Publish using the MQI
- Subscribe using the MQI
- Understand the different ways to subscribe to publications
- Use the new variable length string data type in MQI applications

After completing this unit you should understand how to code applications using the MQI that will allow you to " Publish" messages on a specific topic for other applications to consume.

You will see what needs to be done in a program in order to publish messages.

You will also understand and see what a program needs to do to set up a subscription to a topic, and to consume the messages it receives as a result of that subscription.

Subscription has a much wider range of options controlling how and to what messages you subscribe to than publishing has. On completing the unit you should be able to describe what these options are.

Additionally this session discusses the new "variable length string" data type introduced in version 7.0 specifically to deal with topic strings. The JMS specification defines these as being of unlimited length, but are limited to ten thousand characters in the MQI .

And remember this session does not attempt to cover all the options in detail; you should use this presentation together with the information in the product information center.

Introduction

- In WebSphere MQ V7:
  - New verbs allow MQI applications to easily perform publish/subscribe.
  - Publications and subscriptions can be handled by both MQI and JMS applications.
  - Publish/subscribe functionality is now available on z/OS without needing Message Broker.

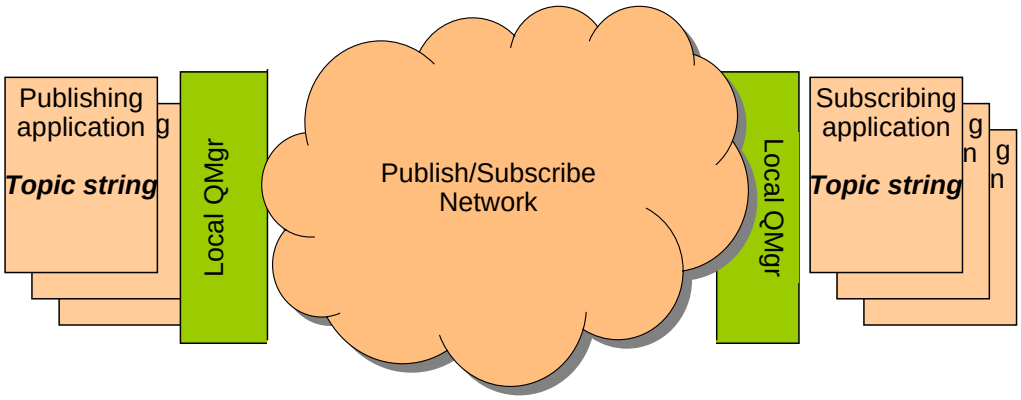
This presentation covers the MQ Application Programming Interface extensions introduced into WebSphere MQ version 7.0.

The new verbs and options allow MQI application to perform publish/subscribe operations in a simple straight forward manner, using MQ verbs to publish messages to topics and to establishing subscriptions to topics. This is in contrast to the level of indirection required in version six to set up headers and send messages to broker queues.

Publication and subscription is, broadly, functionally equivalent for JMS applications and MQI applications and the two APIs can interact directly. That is to say; messages published by JMS applications can be consumed by MQI subscribers and vice versa.

Also for the first time WebSphere MQ for z/OS has publish and subscribe capabilities without the need for WebSphere Message Broker.

Publish / subscribe applications



- Applications connect to local queue manager
- Publish/Subscribe asymmetry compared to Put/Get
- Can use JMS or any MQI language
- Can also interact with MQ V6 queued Pub/Sub command messages

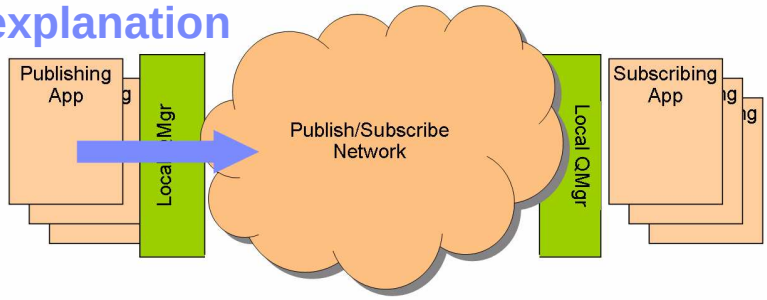
Shown here is a general publish / subscribe application.

What is shown here are one or more publishing applications which are publishing messages. These messages are published to a topic string. Also shown are one or more subscribing applications which are being sent those messages which match the topic strings they are subscribing to. Also valid is the fairly uninteresting cases where there are zero publishing applications, in which case subscribers are listening to nothing. Or the case where there is zero subscribing applications in which case publishers might be shouting into the void. Or of course the very special case of zero publishers and zero subscribers.

Look a little more closely and you will see that each publishing or subscribing application is connected to some local queue manager. These connections might be local binding, or over a network by way of the client code. The participating queue managers are connected together by the network cloud. It is an important element of publish subscribe applications that the logical connection between the programs involved is the topic strings they are publishing to and subscribing from and the nature of the network connectivity is irrelevant.

The wider application can be made up of a mix of JMS and MQI publishers and subscribers as appropriate. It might be that mainframe COBOL applications are publishing data that JMS subscribers on WebSphere Application Server are consuming. Equally older queue managers with the “queued” publish subscribe interface can also be connected to the network and freely exchange messages with V7.0 publish/subscribe applications.

Publication explanation



- Publication is MQPUT to a topic string
- Topic string can be identified in three ways
  - Specify whole topic string in application.
  - Pre-defined topic object contains the topic string.
  - Concatenation of pre-defined string from topic object plus partial topic string from application.

The first thing to see is that NO NEW VERB is introduced for publishing. Messages are published by using the existing MQPUT verb, but instead of the object descriptor in the MQPUT identifying a QUEUE the Object descriptor resolves to a topic string.

So publishing is merely putting, but to a topic string instead of a queue.

What is new and somewhat different is how the topic string is identified:

Three choices are available:

The first is to specify the whole topic string in the object descriptor. The application supplies the whole topic string.

Example: “usa/share/price/ibm” is specified as a topic string (using a new field of data type varying length string).

The second is equally simple an MQ administrative TOPIC object has been predefined, by an administrator) that contains the whole topic string. And the application specifies this administrative topic object in the same way a QUEUE is specified. The administrator supplies the whole topic string.

Example: MQSC is used to “DEFINE TOPIC(IBM_PRICE) topicstring(‘usa/share/price/ibm/’)”, and the object descriptor specifies “IBM_PRICE” as a topic object.

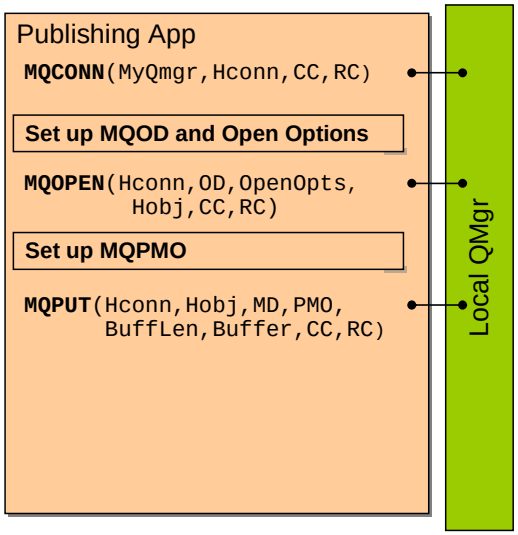
The third way is a little more interesting, and less obvious. You can create an administrative Topic Object that contains a “stem”, or beginning part, of the full topic string and supply in the object descriptor the final part of the string. The application and administrator cooperate to supply the whole string.

Example: MQSC is used to DEFINE TOPIC(PRICES) topicstring(‘usa/share/price) and the object descriptor specifies this PRICE object descriptor and the additional (partial) topic string ‘ibm’ which resolves to “usa/share/price/ibm” .

This might be useful to allow you to administratively change where in the topic hierarchy the application was operating. Or if this was a package application which needed to be configured to fit into a topic structure a particular enterprise was using.

The next slides show how this is done in code …. Or at least pseudocode.

Publishing applications



- Connect to the queue manager
- Open a topic
- Put publications to that topic

This program flow should look familiar to any MQI programmer, and essentially looks like the put to a queue.

The application first connects to a queue manager – no change here.

Setting up the Object descriptor and Open options are where the topic string being opened is identified.

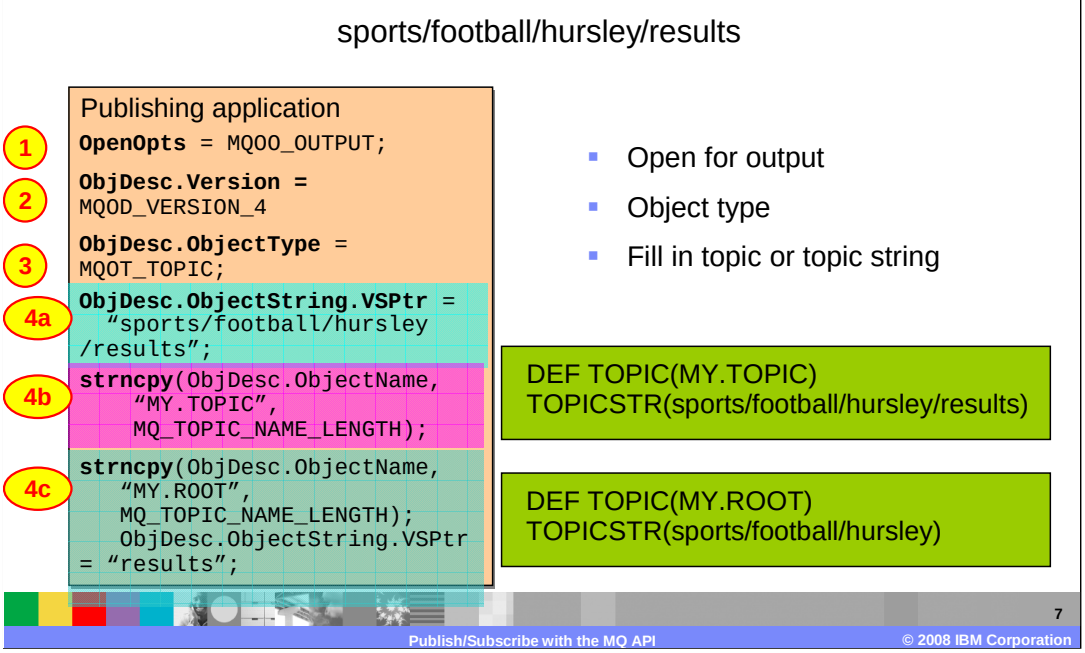
The MQOPEN itself is just as before.

Setting up the Put Message Options is shown in a later slide; the MQPMO has new options relating to publishing.

The actual MQPUT call then publishes the message to the topics string and all subscribers are delivered a copy.

Now for a look in more detail.

Set up MQOD and open options



This slide shows how to set up the object descriptor to open the 'sports/football/hursley/results/' topic string.

The first three key things (shown by bullets one to three) are:

One, the object must be opened for output when it is used for publishing.

Two, the MQOD version must be four as new MQOD fields are introduced in version 7.

Three, the Object Type is set to the new symbol for a topic, MQOT_TOPIC.

The last thing to do is to identify the topic string that publishing is to occur to – as mentioned earlier there are three alternatives shown as 4a, 4b and 4c.

4a) Specifying the whole topic string in the program, done here by setting object descriptor "ObjectString.VSPtr" to point to the string.

While NOT specifying and "ObjectName".

This results in the resolved topic string being the one supplied in the program.

Note: this is a use of the new varying length string data type details of which are discussed later.

4b) Specifying ONLY an ObjectName in the Object Descriptor.

While not specifying and ObjectString.

This results in the resolved topic string being the one specified in the administrative definition of the TOPIC Object.

In this case MQSC commands or MQ Explorer must have been used to create a topic object named "MY.TOPIC" which had the topic string

"sport/football/hursley/results" as an attribute.

4c) Here BOTH ObjectName and ObjectString are specified.

This results in the resolved string being from a concatenation of a string from and administrative topic together with the string supplied by the program.

In this case MQSC commands or MQ Explorer must have been used to create a topic object named "MY.ROOT" which had the topic string

"sport/football/hursley" as an attribute. The object string "results" specified by the program is concatenated together with a connecting slash symbol by this root to give the final resolved string "sport/football/hursley/results".

This choice of ways to identify a topic string applies to all cases where a topic sting is needed.

The next slide looks in more detail at changes to the object descriptor.

MQOD C declaration with new fields

```
struct tagMQOD {
  MQCHAR4   StrucId;              /* Structure identifier */
  MQLONG    Version;              /* Structure version number */
  MQLONG    ObjectType;           /* Object type */
  MQCHAR48  ObjectName;           /* Object name */
------  Part Omitted ------
  MQCHARV   ObjectString;         /* Object long name */
  MQCHARV   SelectionString;      /* Message Selector */
  MQCHARV   ResObjectString;      /* Resolved long object name*/
  MQLONG    ResolvedType;         /* Alias queue resolved object type */
  /* Ver:4 */
```

```
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 slide illustrates two control blocks in the C programming language.

First, the new fields added to the Object Descriptor in version 7.0. These include:

ObjectString, which is used to identify the topic string to be opened.

SelectionString is used for specifying selection string – see the presentation on selectors.

ResObjectString is a returned value giving the resolved name of the object (topic) string that was actually used.

These new variables are varying length strings, of up to ten thousand characters, and are of the new MQCHARV data type.

This is illustrated in the second part of the slide.

The MQCHARV data type describes a variable length string by either a pointer, or offset into a buffer together with a length and a Coded Character Set ID.

Either a pointer or an offset can be used to specify the buffer containing the variable length string. It is expected that a pointer will be used in most cases but some languages do not support pointers in which case an offset must be used. The offset is a positive or negative number and measures the offset from the beginning of the containing structure being used as a parameter on a call. So for ObjectString, SelectionString and ResObjectString in the MQOD the offset in each case is measured from the beginning of the MQOD structure.

When an MQCHARV data type is used to return data from an MQI call (for example, ResObjectString) the length of the string is always returned in VSLength. When passing data into an MQI call (for example, ObjectString) the caller might specify the string length in VSLength, or set VSLength to MQVS_NULL_TERMINATED and of course terminate the string with a null character.

Set up MQPMO

- MQPMO_RETAIN
  - The publication being sent is to be retained by the queue manager.
  - No more than one publication is retained at each node of the topic tree.
- MQPMO_SUPPRESS_REPLYTO
  - ReplyToQ and ReplyToQMgr fields are not passed on to the subscribers.
- MQPMO_NOT_OWN_SUBS
  - Tells the queue manager that the application does not want to send any of its publications to subscriptions it owns.
- MQPMO_SCOPE_QMGR
  - Publications are not forwarded to any remote publish/subscribe queue manager.

The last thing to talk about from a publishing point of view is the new Put Message Options that are added in support of publish / subscribe.

It is possible for each unique topic string to have at most one, so either one or zero, retained publications. Retained publications are the publications that are “held” by the queue manager and delivered either immediately a new subscriber connects, or on request using the special SUBRQ Call. Setting the MQPMO_RETAIN option when putting a message will make the put message the current retained publication for that topic string.

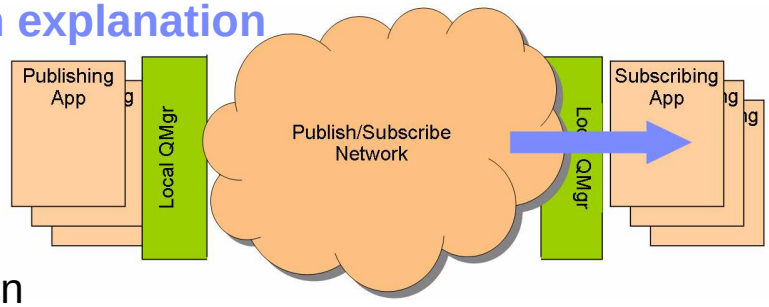
The MQPMO_SUPPRESS_REPLYTO option can be used to suppress the ReplyTo fields on messages passed to subscribers.

MQPMO_NOT_OWN_SUBS and MQPMO_SCOPE_QMGR are used to limit which subscribers might get this message. The first means that subscriptions with the same connection handle will not receive the publication. The second. that only subscribers connected to the same queue manager will receive the publication.

That completes the description of publishing using the MQI.

MQOPEN is extended to allow topic strings to be opened for output. MQPUT is extended to allow messages to be put to these topics with new options related to publishing.

Subscription explanation



- Subscription
  - A subscription involves two key elements
    1) Identifying the topic string, including wildcards, being subscribed to.
    2) Identifying the destination that qualifying messages are to be placed on.
  - Note the subscription call MQSUB does not get any messages. These must be read from the destination queue with an MQGET call.

Subscription is more of a new concept in the MQ API.

The new MQSUB call sets up, or registers, a subscription which requires two key pieces of data.

The topic string being subscribed to which makes the link to publishing applications; the subscription can be to a specific string or can involve wildcards.

Example: Subscribing to “shares/newyork/prices/#” causes messages to be delivered for any publications such as “shares/newyork/prices/IBM” and “shares/newyork/prices/Microsoft”.

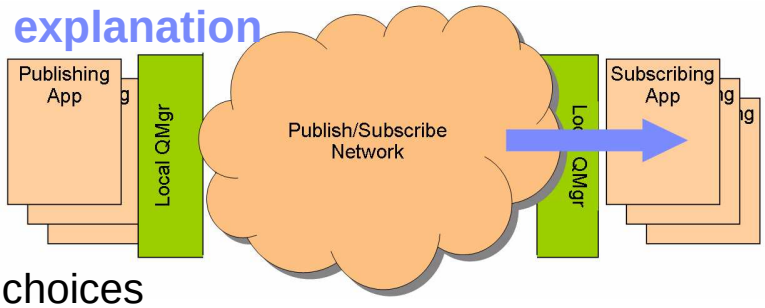
The other piece of data is the destination the subscribed messages are to be placed on. This is a queue. The queue can be specified in two ways.

The first way is that the application has already opened a queue, in which case the queue handle can be passed into the call – this is referred to as an unmanaged destination.

The alternate is to specify a null handle, in which case a queue is created by the queue manager and the handle is returned from the call – this is referred to as a managed destination.

It is worth stressing here that the MQSUB call does not “get” any messages for the application. MQSUB merely sets up a subscription in the queue manager which causes appropriate messages to be placed on the destination. These messages still need to be read by an application using an MQGET (for example).

Subscription explanation



- Subscription choices
  - Non-durable subscription - subscribe, receive pubs, and unsubscribe in a single program
  - Durable subscription – subscribe, go away, return later to retrieve publications, needs subscription name.
  - Selection Criteria – In addition to the topic string.
    - A SQL expression can be supplied that “filters” the messages based on message

The main choices for a subscription are whether the subscription is durable or non-durable.

The non-durable is the simpler case. A non durable subscription starts with the MQSUB call and ends when the application closes the handle or disconnects.

So the subscription is relatively short-lived.

A durable subscription can continue beyond the life of the application that starts the subscription. When no application is connected to the subscription, messages are still delivered to the destination specified.

An application can start a subscription; disconnect; and return later to collect all the messages delivered while it was “away”. Because of the need to reconnect all durable subscriptions must be identified by a unique subscription name.

The reconnecting “application” might be a completely different program, even running under another user ID.

Another major option for a subscription is whether you want to receive all the messages published to a topic (possibly with wildcard) or only certain messages that satisfy a selection criterion made in a selector. The selector is a SQL expression that selects using the Message Properties of a Message.

Example: Select only messages with the user property “DollarValue” Greater-or-equal-to” 10,000.

Selectors and Message Properties are covered in more detail in other presentations.

New API call for subscription registration

- MQSUB – register subscription
  - Hconn – connection handle
  - SubDesc – MQSD – subscription descriptor structure
  - Hobj – object handle from destination queue
  - Hsub – object handle returned for subscription
  - CompCode – completion code
  - Reason – reason code

Here is what an MQSUB call looks like.

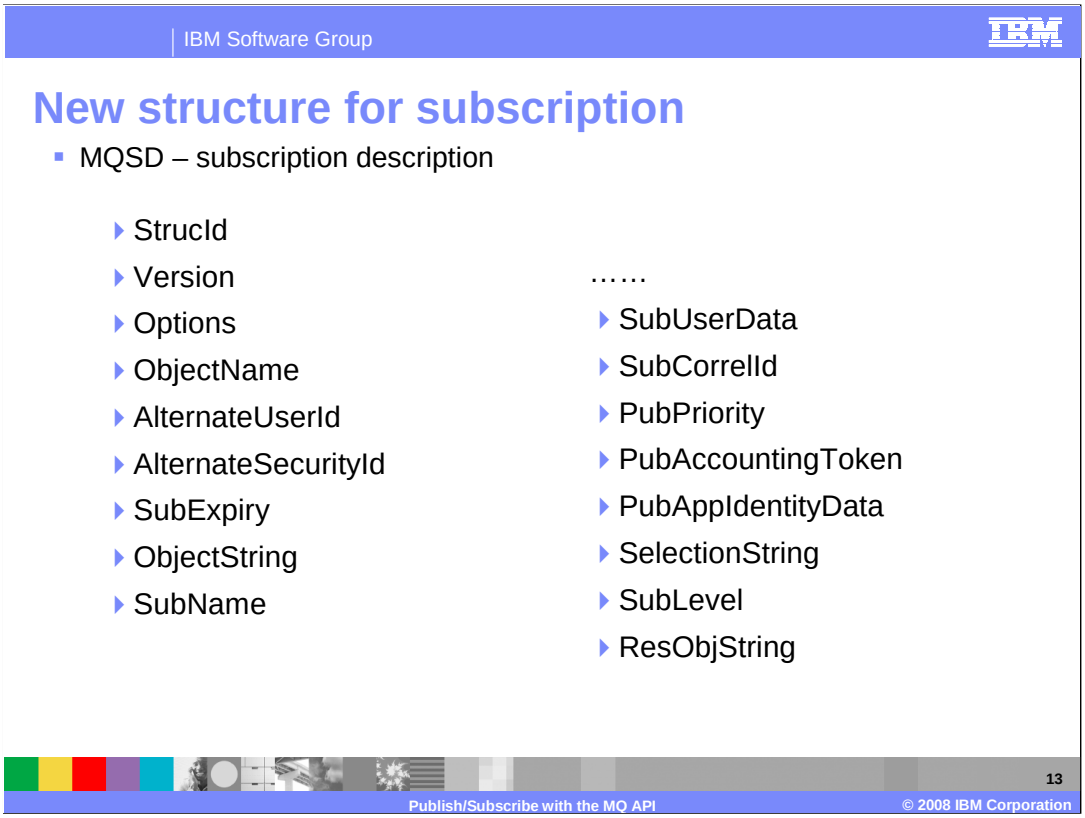
In the main it looks like most MQI calls with a Subscription Descriptor block introduced to encapsulate the options for the subscription.

The thing that makes this call somewhat different from most MQ API calls is the presence of two object handle parameters.

The first is the object handle representing the destination queue where qualifying messages are placed by the queue manager. This can be an input parameter – the unmanaged destination case – where a queue handle is supplied by the application or an output parameter – the managed case – where the queue manager creates a destination for the applications use. In either case after a successful call this parameter contains the handle of a destination that is suitable for use to read the subscribed messages from.

The second is a handle representing the subscription itself and is an output parameter.

Next the subscription descriptor is looked at in more detail.



The MQSD – Subscription descriptor is a new control block for specifying the details of a subscription. It has several fields; details of all these fields can be found in the information center.

Looking at some key fields: Objectname and ObjectString are alternate ways of specifying the topic to be subscribed to in just the same way a topic string was identified in the MQOD when publishing.

SubName is the unique name for this subscription and can be used to reconnect to a durable subscription.

SelectionString is the SQL selection string that filters the messages to be received.

Sublevel introduces the concept of subscription level. This is to allow the interception of publications by an intercept application which needs an opportunity to modify publications before passing them on to the eventual subscribers.

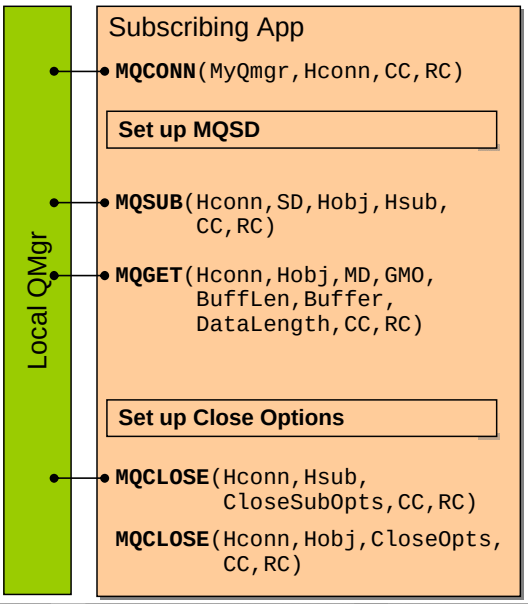
Original publishers (all standard applications will publish at publevel=9 (maximum), all final subscribers will subscribe at sublevel=1 (minimum).

An intercepting publication should subscribe at some intermediate level (say 6) and publish at that level minus one (so perhaps 5).

This is not a feature expected to be used by “application” programs but by system or utility programs.

Non-durable subscription application

- Connect to a queue manager
- Subscribe to a topic
- Get publications published to that topic
- Close subscription to unsubscribe
- Close queue



Here is the “code” for a non durable subscription.

Note the order of operations.

Like all applications the first operation is to connect.

Before subscribing a Subscription Descriptor Block specifying the topic string, selectors etcetera for the subscription must be set up.

In this case a destination object has NOT been opened and passed. So this is a managed subscription with the destination created by the queue manager.

After the MQSUB call a standard MQGET call is used to receive the subscribed messages, with whatever waits, loops and business logic you require.

At the end TWO MQCLOSE commands are required. One on the handle for the subscription itself, and one on the destination handle.

As this is a non durable subscription closing it will terminate the subscription and delete any unread messages from the managed queue.

Set up MQSD – subscription description

Topic Object: SPORTS.RESULTS

- Fill in topic object name
- Choose to Create a subscription
- Select durability
- Choose for the queue manager to create the queue
- Any other options
  - Call returns you a subscription handle.

Subscribing App

```
SubDesc.ObjectName =
 "SPORTS.RESULTS";


SubDesc.Options  = MQSO_CREATE
      | MQSO_NON_DURABLE
      | MQSO_MANAGED
      | MQSO_FAIL_IF_QUIESCING;



MQSUB(Hconn,SD,Hobj,Hsub,
      CC,RC)
```


Here is a little more detail on setting up a subscription descriptor.

In this case the topic string to be subscribed to is identified by by naming a TOPIC OBJECT (which will have the topic string as one of its properties).

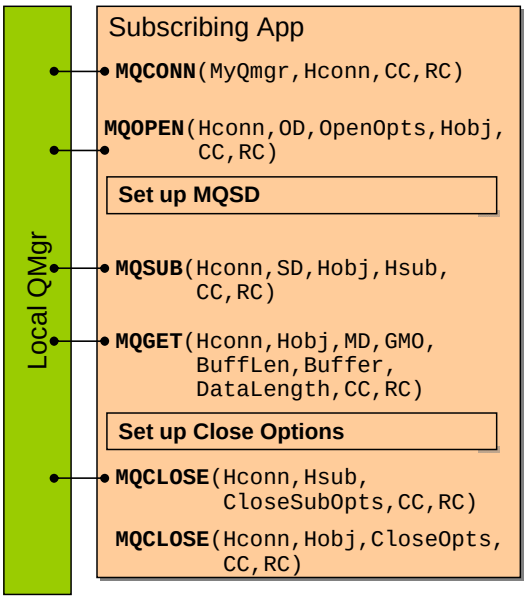
The MQSD options field is a field that is likely to need to be specified.
In this case a New; Non durable; Managed; subscription is being set up.

Additionally, the subscription should not start if the queue manager is already quiescing.

After the call completes the Hobj handle is open ready for reading messages delivered to the managed queue as a result of the subscription.

Durable subscription application

- Connect to a queue manager
- (Optionally) open a destination queue
- Subscribe to a topic
- Get publications published to that topic
- Close subscription with MQCO_KEEP_SUB
- Close queue



In this second example creation of a durable subscription is shown.

This is essentially the same as the non durable case, but note the MQCLOSE option of MQCO_KEEP_SUB which causes the subscription to be maintained after the MQCLOSE. Note that for durable subscriptions this is the default but is specified here for clarity.

Any messages published to the topic subscribed to will continue to be delivered.

Set up MQSD – subscription description

Topic String: sports/football/results/#

- Fill in topic string
- Fill in subscription name
- Choose to Create, Resume or Alter a subscription
- Select durability
- Choose not to receive retained publications
- Any other options
  - Call returns you a subscription handle.

Subscribing App

```
SubDesc.ObjectString.VSPtr =
 “sports/football/results/#”;
SubDesc.SubName.VSPtr     =
“MoragsSub”;

SubDesc.Options = MQSO_CREATE
   | MQSO_DURABLE
   | MQSO_NEW_PUBLICATIONS_ONLY
   | MQSO_FAIL_IF_QUIESCING;



MQSUB(Hconn,SD,Hobj,Hsub,
      CC,RC)
```


This time the subscription descriptor has different options.

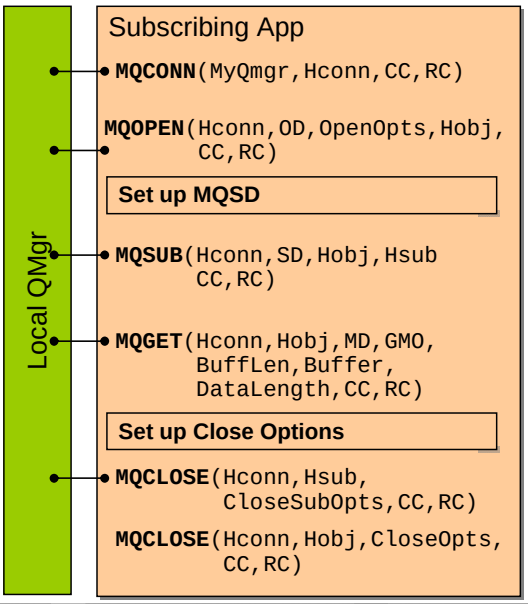
In this case the topic string to be subscribed to is identified by by naming the topic sting directly using the MQSD.ObjectString field, which is an instance of the new MQ varying length string data type.

Because this is a durable subscription and might be reconnected to it must be given a (unique) name.

Also slightly different options are chosen for this subscription.

Return to the durable subscription

- Connect to a queue manager
- (Optionally) open a destination queue
- Subscribe to same topic with MQSO_RESUME
- Close subscription with MQCO_REMOVE_SUB
- Close queue



Returning to a durable subscription, the code looks just like creating a subscription.

Differences are in the options.

You subscribe using the MQSO_RESUME option in order to reconnect to an existing subscription.

And you might optionally close with the MQCO_REMOVE_SUB option which causes the durable subscription to be ended permanently.

MQSD -- subscription options 1

- Mandatory choice
  - MQSO_CREATE for a new subscription
  - MQSO_RESUME to continue existing subscription by name
  - MQSO_ALTER to change existing subscription by name
- You can combine options
  - MQSO_CREATE + MQSO_RESUME
    - Create if it doesn't exist
    - Resume if it does exist
  - MQSO_CREATE + MQSO_ALTER
    - Create if it doesn't exist
    - Alter if it does exist

Looking a little more closely at the subscription options that can be set; the first options choice that must be made concerns the "newness" of a subscription.

MQSO_CREATE indicates that creating a new subscription is allowed.

MQSO_RESUME indicates that resuming an existing (named) subscription is allowed

MQSO_ALTER indicates that an existing subscription can have its properties altered.

These options can be combined so CREATE and RESUME means use existing if it exists and create if it does not. Without the CREATE option the MQSD call fails if the named subscription did not already exist.

MQSD -- subscription options 2

- Durability options
  - MQSO_DURABLE for a subscription that will last beyond the end of the application connection
  - MQSO_NON_DURABLE for a subscription that ends with the close of the subscription
- Destination options
  - MQSO_MANAGED if you want the queue manager to create and manage the queue where the publications are stored

The options are also where the durability is specified.

The subscription is either DURABLE – can continue after subscription is MQCLOSE'd (with the right close options) or it is not.

Additionally the choice of managed or unmanaged destination is made here. MQSO_MANAGED asks the queue manager to create a destination, otherwise the application must supply a queue handle.

MQSD -- subscription options 3

- MQSO_SCOPE_QMGR – this subscription is made only on the local queue manager
- MQSO_NEW_PUBLICATIONS_ONLY – only receive new publications, not previously created retained publications
- MQSO_PUBLICATIONS_ON_REQUEST – only receive retained publications using MQSUBRQ, no new publications

Additional subscription options include; the MQSO_SCOPE_QMGR subscription is made only on the local queue manager, and is not propagated to any adjacent queue managers. Only messages published on this queue manager delivered. The default is to receive all publications from clustered or hierarchically connected queue managers.

Setting MQSO_NEW_PUBLICATIONS_ONLY means that no currently retained publications are to be sent, when this subscription is created, only new publications. This option only applies when MQSO_CREATE is specified.

The MQSO_PUBLICATIONS_ON_REQUEST option radically changes the nature of the subscription. Setting this option indicates that no messages are to be placed on the destination as a result of this MQSUB call. The MQSUB call identifies the destination and what messages qualify but no messages are delivered UNTIL an MQSUBRQ call is issued. The MQSUBRQ call causes all the appropriate retained publications to be delivered to the destination.

The MQSO_PUBLICATIONS_ON_REQUEST and MQSO_NEW_PUBLICATIONS_ONLY cannot both be specified.

MQSD -- subscription options 4

- MQSO_WILDCARD_CHAR – wildcards are specified with characters without reference to the topic hierarchy
  - / No significance, just another character
  - * Wildcard, zero or more characters
  - ? Wildcard, exactly one character
  - % Escape character
    Example: “%*’ stands for and asterisk in a topic
    Example: the string “hot%?cold%?” matches only with the nine character string h-o-t-?-c-o-l-d-?.

The final subscription options to be considered is the alternate wildcard matching.

MQSO_WILDCARD_CHAR
This option specifies that the alternate (compatible with the older MQRFH1 not MQRFH2 headers) wildcard matching for topic strings should apply.
Wildcards only operate on characters without reference to the topic hierarchy within the topic string.

The / (slash) has no significance, it is just another character.
The * (asterisk) is a wildcard standing for zero or more characters.
The ? (question mark) is a wildcard standing for exactly one character.
The % (percent symbol) is an escape character. When used before a slash, asterisk, or question mark, it means that the character is literal.

Compare this to the default behavior specified by MQSO_WILDCARD_TOPIC, where wildcards only operate on topic elements within the topic string.

The / (slash) is the separator between levels in the hierarchy.
The # (pound sign or hash mark) stands for zero or more levels in the hierarchy.
The + (plus sign) stands for exactly one level in the hierarchy.
Note that this use of wildcards supplies exactly the meaning provided in WebSphere Message Broker V6 when using MQRFH2 formatted messages for Publish/Subscribe.

This is not a complete list of the available options, but only the key ones for understanding MQ Pub/Sub. You should refer to the information center for complete information on all options.

Set up close options

- MQCO_KEEP_SUB
  - Subscription is kept
  - Only valid for durable
  - Default for durable
- MQCO_REMOVE_SUB
  - Subscription is closed
  - Any publications not yet consumed from a managed destination are removed when the subscription is closed.
  - Default for non-durable

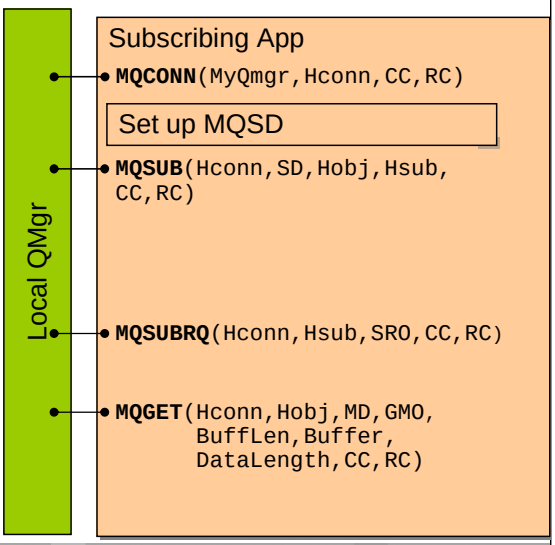
Close options are fairly straight forward.

MQCO_KEEP_SUB only applies to durable subscriptions and is then the default. It means that after the close messages continue to be delivered to the destination ready for the time when the subscription is resumed.

On the other hand The MQCO_REMOVE_SUB permanently closes the subscription. Any managed destination is deleted along with any unread publications on that destination.

Publications on request only

- Connect to the queue manager
- Subscribe to a topic using MQSO_PUBLICATIONS_ON_REQUEST
- **No MQGETs after**
- Request the retained publications
- Get publications published to that topic



The MQSO_PUBLICATIONS_ON_REQUEST option and the MQSUBRQ call were mentioned earlier in the subscription options section.

Here you see how it might be coded.

The MQSUB call sets up the parameters of the subscription – topics – destinations – but does not begin the placing of messages on the destination, so no MQGET

At some time later the application issues an MQSUBRQ, at which point the appropriate retained publications are delivered to the destination and can be got by the application.

This option allows the retained publications to be delivered on demand.

New API call for subscription requests

- MQSUBRQ – subscription request options
  - Hconn – connection handle
  - Hsub – subscription handle
  - Action – action requested on this subscription
    - MQSR_ACTION_PUBLICATION
  - SubRqOpts – MQSRO – subscription request options
  - CompCode – completion code
  - Reason – reason code

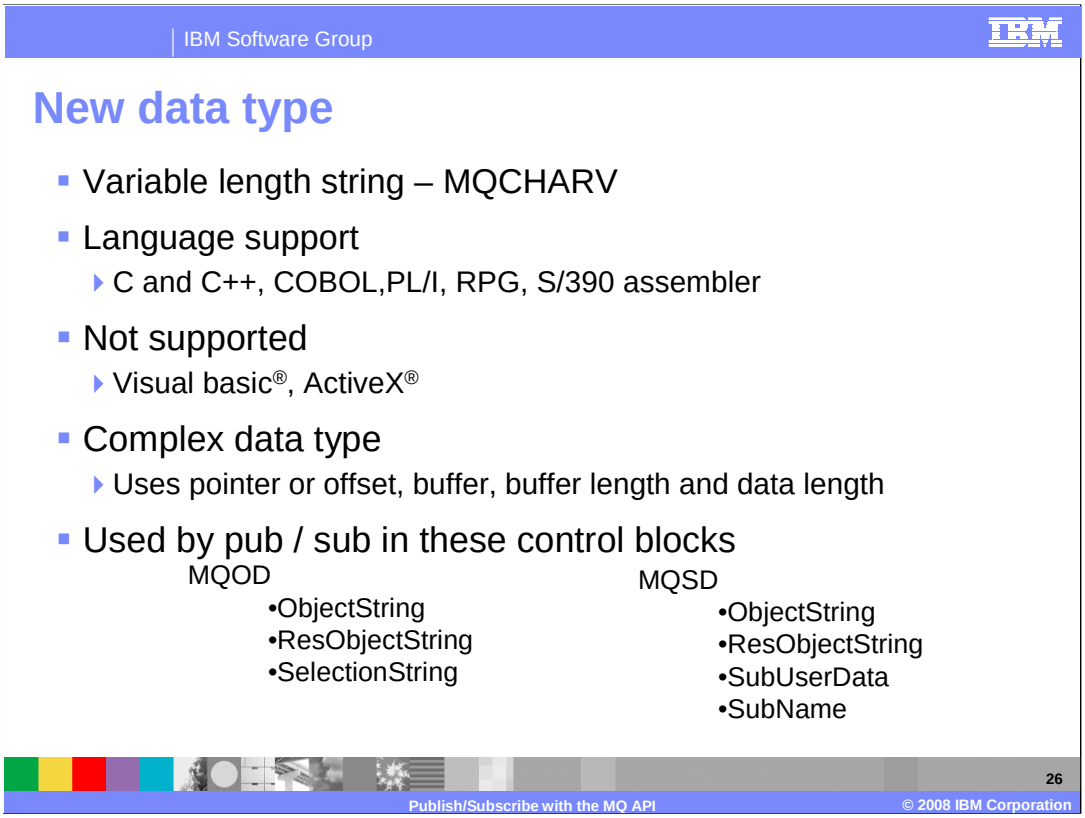
Having set up the subscription with the MQSO_PUBLICATIONS_ON_REQUEST option the MQSUBRQ call causes the qualifying messages to be delivered.

It is a very simple call passing in the subscription handle and the only valid action of MQSR_ACTION_PUBLICATION.

Although a MQSRO – Subscription on Request Options field is passed a very limited set of options can be specified.

In fact the only choice is whether to code the MQSRO_FAIL_IF_QUIESCING option.

Note: Since the topic in the existing subscription represented by the Hsub parameter might contain wildcards, the subscriber might receive multiple retained publications.



One last item before leaving the publish/subscribe API.

The new MQ data type MQCHARV to handle varying length strings was illustrated earlier.

Here is a recap.

It is supported in most MQI languages, but not Visual Basic and ActiveX where the APIs have not been extended in V7.

It is a complex data type using a pointer or offset to locate a varying length string in a buffer.

And is used for topic strings, selector strings, subscription names and for the user data passed in a subscription.

Publish/subscribe using MQI - summary

- The verbs used are:-
  - MQOPEN
  - MQPUT
  - MQSUB
  - MQGET
  - MQSUBRQ
  - MQCLOSE
- New structures to accompany new verbs
  - MQSUB – MQSD – Subscription Descriptor
  - MQSUBRQ – MQSRO – Subscription Request Options

In summary, MQOpen and MQClose are extended.

MQPut is used to publish by putting to a topic.

MQSUB verb is new and used to specify a subscription. Messages are actually got using a standard MQGET call.

New call MQSUBRQ can be used to get all the retained publications available at a point in time.

New structures and options are introduced.

And to repeat that this is very much an overview – detail on the MQI options and coding can be found in the information center.

Unit summary

Now that you have completed this unit, you should be able to:

- Publish using the MQI
- Subscribe using the MQI
- Understand the different ways to subscribe to publications
- Use the new variable string data type in MQI applications

Now that you have completed this unit, you should understand how to: Publish using the MQI, Subscribe using the MQI, Understand the different ways to subscribe to publications and use the new variable string data type in MQI applications.

Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send e-mail feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_iea_310_wmqv7_API_1_PubSub.ppt

This module is also available in PDF format at: ../iea_310_wmqv7_API_1_PubSub.pdf

You can help improve the quality of IBM Education Assistant content by providing feedback.

Trademarks, copyrights, and disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM WebSphere z/OS

A current list of other IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

ActiveX, and Visual Basic are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements or changes in the products or programs described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (for example, IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.