

Getting the Most out of IBM MQ Publish/Subscribe

Christopher Frank

chrisfra@us.ibm.com

Purpose of this Session

- **Not an Intro to Publish/Subscribe**
 - ▶ There was one at this conference, though
- **Topics and using them wisely**
 - ▶ Things to do and things to avoid
- **Subscriptions**
 - ▶ Clever things you can do
 - ▶ Things to watch out for
- **MQI and JMS considerations**
 - ▶ And how to accommodate them
- **Where's my message?**
 - ▶ Making sure none "slip through the cracks"
- **Common mishaps and how to avoid them**
- **Performance considerations!**

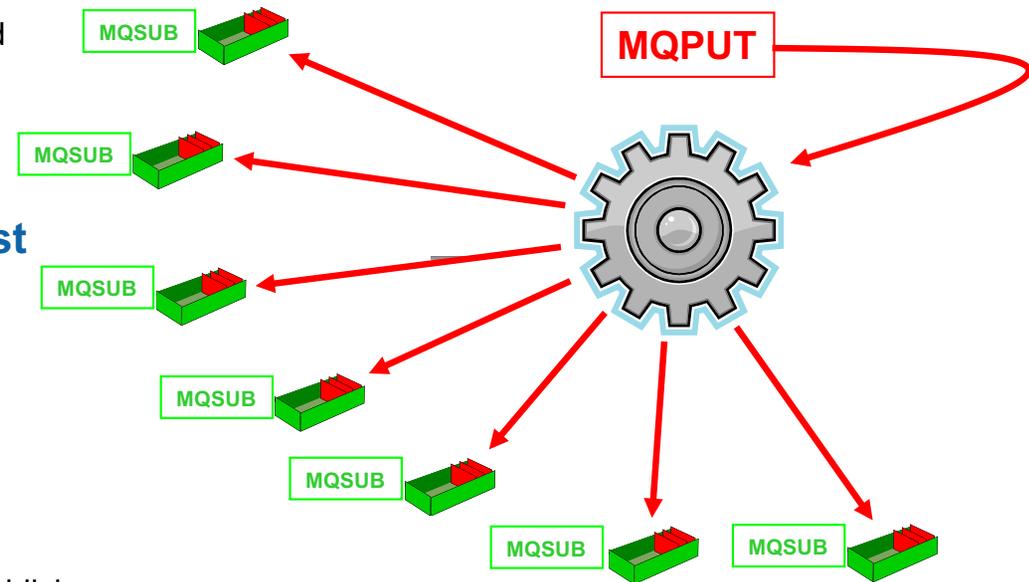
MQ Pub/Sub – The Basics in One Slide

■ Applications use MQOpen to create/open a topic

- ▶ Then use **MQPut** to publish to that topic
 - Publishers are advertising data to interested parties
 - If there are no interested parties the data is discarded
- ▶ Topics *objects* can be defined but this is not necessary
- ▶ Topics *objects* defined for administrative reasons
- ▶ Topics are not queues!

■ Applications use MQSUB to register interest

- ▶ Then use **MQGet** to consume publications
- ▶ Can subscribe to topics, topic *objects*, or both
- ▶ Publications delivered to a subscribers queue
 - Can BYOQ or have MQ provision one
 - Queue allows subscribers to be disconnected
- ▶ Publishers and subscribers are not aware of each other
 - There can be many subscribers to a topic, or none
 - These can come and go with no awareness by the publisher
- ▶ Subscribers can register interest in multiple topics
- ▶ Subscriptions can be administratively defined



MQ Pub/Sub – The Basics in One Slide

N
O
T
E
S

- What are publishers?
 - Applications use MQPUT to publish messages to Topics. Notice that on the publishing side there is no queue – publishers are essentially making data available for consumers that are interested in that data. It is perfectly allowable for no one to be interested in a particular topic, in which case the data is simply discarded. This is an important distinction of topics vs queues.
 - Topics do not need to be defined in advance – the entire topic namespace can be constructed dynamically, based solely on application actions as opposed to administrative actions. But it is possible to define topics administratively, using *Topic Objects*. This is generally done to enable administrative control over portions of the topic namespace, e.g. to secure portions of it, to assign non-default values to topic attributes, etc.
 - Topics are not queues! When messages are published to a topic, they are put directly into each matching subscribers queue. Thus messages will never “queue up” on the publishing side of things. The exception to this is when the older pub/sub API is used – prior to MQ V7, publications were put onto *stream* queues, which could hold publications from many publishers. This older model is still supported, but there are many advantages to using the native pub/sub interface rather than the older stream-based API.
- What are subscribers?
 - Subscribers use MQSUB to register interest in topics. A subscriber can register interest in a topic directly, or it can specify the name of a topic object to subscribe to (assuming one exists), or – it can use both. This special case will be discussed later in this presentation.
 - When a subscriber registers interest in a topic, a queue must be associated with the subscription. The subscriber can indicate the queue it wants publications delivered to, or it can tell MQ to dynamically create one and manage it on behalf of the subscriber.
 - Publishers and subscribers are generally not aware of each others presence. This enables any number of subscribers to connect to topics, to come and go as they please, to deregister interest, etc without the publisher needing to be involved in any way.
 - Subscribers can register interest in multiple topics, through the use of wildcards. This allows subscribers to request that categories of publications be delivered to it, without the need to specify each topic explicitly. This enables the topic space to be expanded without the application needing to change to accommodate the new names.
 - Subscribers can be administratively defined. This enables subscriptions to be created and pointed at arbitrary queues, allowing the consumer to gain access to publications on a given topic without needing to use the MQSUB verb directly.



Topics



Topics, topics – What are they?

What's in a Name?

- Topic?
- Topic string?
- Topic object?
- Topic name?
- Topic definition?

A Rose by any other Name...



- **Topic *string*:** The pattern that publications are matched against
 - ▶ e.g. Price/Fruit/Apples
- **Topic *object*:** The MQ object representing an administrative point in the topic space
 - ▶ Analogous to a Queue object
- **Topic *definition*:** The MQ definition used to create a Topic object
 - ▶ Analogous to a Queue definition
- **Topic *name*:** The name give to a Topic object (or definition)
 - ▶ Analogous to a Queue name
- **Topic:** Any of the above!

A Rose by any other Name...

N

Because publish/subscribe is a messaging pattern, different implementations will use different terms to refer to the same or similar concepts. Even within an implementation, such as IBM MQ, there can be inconsistencies with how terms are used. And administrators, architects and developers will often use different terms when referring to the same things.

O

Let's use the various permutations of topic as an example. In IBM MQ, there are several terms used when referring to "topics", including:

- Topic
- Topic *string*
- Topic *object*
- Topic *definition*
- Topic *name*

T

And in IBM MQ, these have very specific meanings:

- A Topic *string* refers to the pattern that publications are matched against - e.g. Price/Fruit/Apples
- A Topic *object* refers to the defined MQ object that represents an administrative point in the topic space - analogous to a Queue object
- A Topic *definition* refers to the MQ definition used to create a Topic object - analogous to a Queue definition
- A Topic *name* refers to the name give to a Topic object (or definition) - analogous to a Queue name
- A *Topic*: This unfortunately can be used to refer to any of the above!

E

Problems and misunderstandings can result if these terms are used too loosely.

S

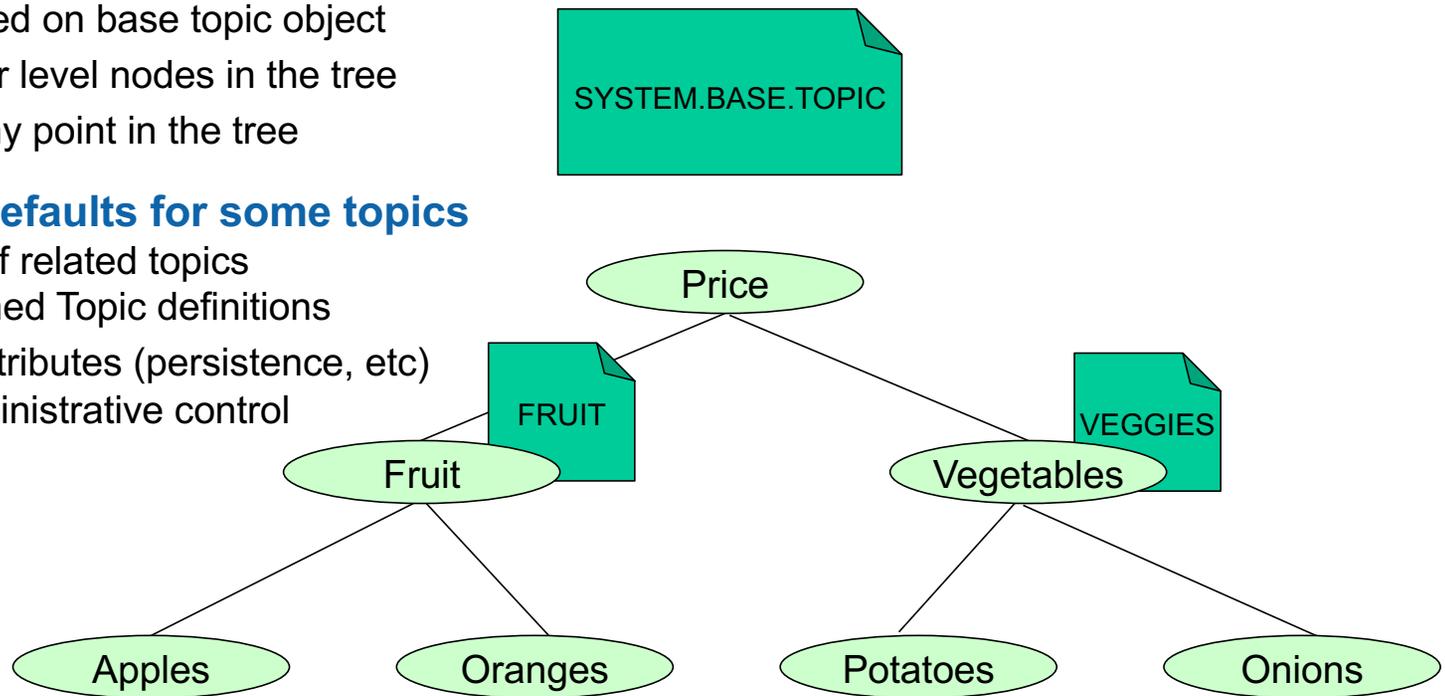
Topic Objects and the Base Topic Object

- **Topic strings enable navigation for publishers and subscribers**

- ▶ Can be fully dynamic – no definitions required
- ▶ Default attributes defined on base topic object
 - Applied to all lower level nodes in the tree
 - Can override at any point in the tree

- **May want to override defaults for some topics**

- ▶ Single topics of group of related topics
- ▶ Done using admin-defined Topic definitions
 - Override default attributes (persistence, etc)
- ▶ Provides a point of administrative control
 - Access control
 - Message delivery
 - Topic tree isolation



Topic Objects and the Base Topic Object

N

By default there is always at least one Topic Object present – this is the base topic object, that has the name SYSTEM.BASE.TOPIC. This object contains all the default settings that control the behavior of topic objects that are lower in the hierarchy. If you want your whole topic tree to behave in the same way and have no need for any other topics, you can alter this object to have the behavior you require.

O

Note that while you can delete this default object, doing so will have no effect - the queue manager will act as if the SYSTEM.BASE.TOPIC was defined with the default attributes that come out of the box. If you delete this object and later decide that you need to change that behavior, you will first need to re-define this object again.

T

While no other Topic Objects are necessary in order to make use of Publish/Subscribe in IBM MQ, there are a number of advantages that can be had by making use of them.

The purpose of Topic Objects are to provide an administrative control point for your topic tree. Without them, all nodes in the topic tree are essentially the same, in terms of security, properties, etc.

E

Topic Objects can be used to override the default configuration attributes for a particular node in the tree. This in turn can influence the behavior of publishers and/or subscribers accessing that point in the topic tree.

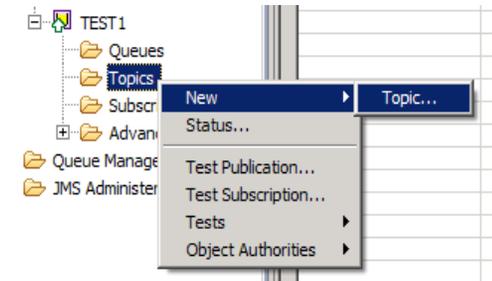
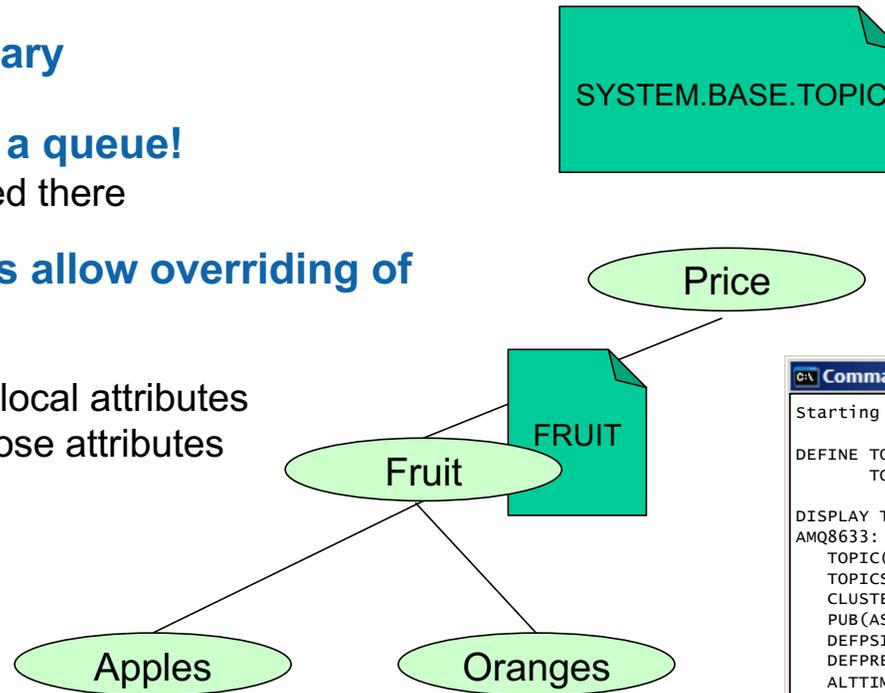
S

Topic Objects also enable you to create security profiles for particular points in the tree. You can control who can Publish, who can Subscribe, etc to particular topics in the topic tree.

A very powerful capability for which you can use Topic Objects is to provide what is called Topic Tree isolation. What this is and why you want to consider making use of it will be discussed later in this presentation.

Defining a topic object

- Useful, but not necessary
- An MQ Object, but not a queue!
 - ▶ Messages are not stored there
- Topic object definitions allow overriding of default attributed
 - ▶ DISPLAY TOPIC shows local attributes
 - ▶ ASPARENT indicates those attributes which are inherited



```
Command Prompt - runmqsc TEST1
Starting MQSC for queue manager TEST1.
DEFINE TOPIC(FRUIT)
    TOPICSTR('Price/Fruit') DURSUB(NO)

DISPLAY TOPIC(FRUIT)
AMQ8633: Display topic details.
TOPIC(FRUIT)                                TYPE(LOCAL)
TOPICSTR(Price/Fruit)                       DESCR( )
CLUSTER( )                                  DURSUB(NO)
PUB(ASPARENT)                               SUB(ASPARENT)
DEFPSIST(ASPARENT)                          DEFPRTY(ASPARENT)
DEFPRESP(ASPARENT)                          ALTDATE(2017-02-26)
ALTTIME(15.05.22)                            PMSGDLV(ASPARENT)
NPMSGDLV(ASPARENT)                          PUBSCOPE(ASPARENT)
SUBSCOPE(ASPARENT)                          PROXYSUB(FIRSTUSE)
WILDCARD(PASSTHRU)                           MDURMDL( )
MNDURMDL( )
```

- How do I determine what attributes are in effect?

Defining a topic object

N

Let's say you need to disallow the creation of durable subscriptions for one half of the topic tree. We can create one TOPIC object at the highest point where we need this behavior to start, and that behavior will be inherited by the nodes in the topic tree below that point without the need for any further TOPIC object definitions.

O

As you might expect, this new object type has DEFINE, ALTER, DELETE and DISPLAY commands. One thing to note about ALTER is that the TOPICSTR parameter of a TOPIC object cannot be altered. Think of this attribute as the other name of the TOPIC object – you cannot alter the name of an object, you must delete and redefine an object to do that.

T

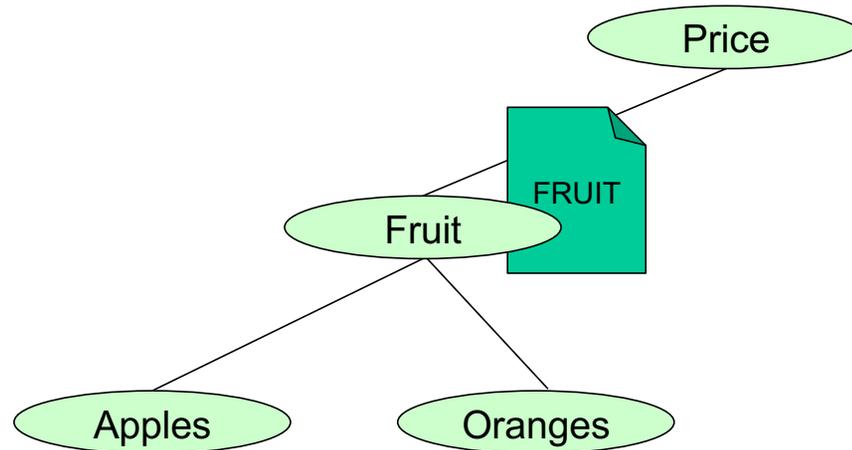
Looking at the DISPLAY output from the object we just defined, we can see that many of the attributes that we didn't specify have the value ASPARENT (or for the character strings – have blanks, which means the same thing as the ASPARENT value). ASPARENT means that the value for this attribute is taken from the next TOPIC object found by walking up the topic tree. If the next TOPIC object found also says ASPARENT for the value that is being resolved we carry on up the tree – eventually we may get to the very top and thus use the values in the SYSTEM.BASE.TOPIC.

E

S

Resolving ASPARENT

- **DISPLAY TOPIC** shows me the topic *definition*
- **DISPLAY TPSTATUS** shows attributes in effect at any node in the topic tree
 - ▶ Regardless of whether a Topic object is defined at that node



Topic status:

Topic string	Publish	Subscribe	Admin topic name	Durable subscriptions
/				
Price	Allowed	Allowed		Allowed
Fruit	Allowed	Allowed	FRUIT	Inhibited
Apples	Allowed	Allowed		Inhibited
Oranges	Allowed	Allowed		Inhibited

DISPLAY TPSTATUS

```
Command Prompt - runmqsc TEST1
Starting MQSC for queue manager TEST1.
DIS TPSTATUS('Price/Fruit')
AMQ8754: Display topic status details.
  TOPICSTR(Price/Fruit)                ADMIN(FRUIT)
  MDURMDL(SYSTEM.DURABLE.MODEL.QUEUE)
  MNDURMDL(SYSTEM.NDURABLE.MODEL.QUEUE)
  DEFPSIST(NO)                          DEFPRTY(0)
  DEFPRESP(SYNC)                        DURSUB(NO)
  PUB(ENABLED)                          SUB(ENABLED)
  PMSGDLV(ALLDUR)                       NPMSGDLV(ALLAVAIL)
  RETAINED(NO)                          PUBCOUNT(0)
  SUBCOUNT(0)                          PUBSCOPE(ALL)
  SUBSCOPE(ALL)
```

Resolving ASPARENT

N

In order to see what the real values being used for the attributes that have the value ASPARENT, you can use the DISPLAY TPSTATUS command.

O

This command takes a topic string, not a topic object as its input. This means you can find the actual values that are going to be used at any point in the topic tree – not just at those points which have defined TOPIC objects.

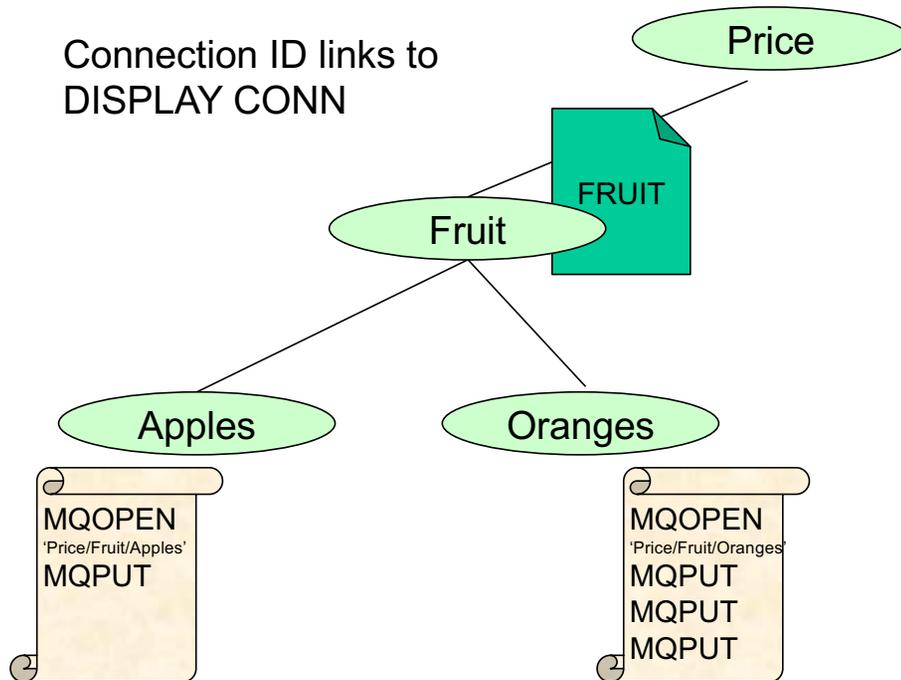
T

E

S

Publisher Topic Status

Connection ID links to
DISPLAY CONN



TOPIC attributes

DEFPRTY
DEFPSIST
DEFPRESP
PUB
PUBSCOPE
PMSGDLV
NPMSGDLV

```
Command Prompt - runmqsc TEST1

Starting MQSC for queue manager TEST1.

DIS TPSTATUS('Price/Fruit/+' ) TYPE(PUB) ALL
AMQ8754: Display topic status details.
  TOPICSTR(Price/Fruit/Oranges)      LPUBDATE(2017-02-26)
  LPUBTIME(16:50:44)
  ACTCONN(414D5143544553543120202020202020832AC44720005E02)
  NUMPUBS(3)
AMQ8754: Display topic status details.
  TOPICSTR(Price/Fruit/Apples)      LPUBDATE(2017-02-26)
  LPUBTIME(16:50:37)
  ACTCONN(414D51435445535431202020202020832AC44720007601)
  NUMPUBS(1)
```

Administration for Publishers

N

There are a few attributes on the topic object that are relevant to publishers. We will look at those here along with the topic status display that shows information about publishers.

O

There are various options on MQPUT that can be left to resolve from the object that was opened. Priority, Persistence and Asynchronous Put Response. Using MQPRI_PRIORITY_AS_TOPIC_DEF, MQPER_PERSISTENCE_AS_TOPIC_DEF and MQPMO_RESPONSE_AS_TOPIC_DEF (all of which constants have the same numeric value as their equivalent AS_Q_DEF constants) means that that the actual value is resolved from the topic object.

T

The TOPIC attribute PUB determines whether publishing is allowed at this point in the topic tree. If set to DISABLED, an MQPUT call will fail with MQRC_PUT_INHIBITED. We'll cover PMSGDLV and NPMSGDLV later in this presentation, when we talk about message delivery options and configuring behavior for publication failures.

E

As already discussed, with the resolution of ASPARENT values, the object that finally resolved the value may be further up the topic tree than the point at which you are publishing.

S

Using DISPLAY TPSTATUS TYPE(PUB) you can see the details of the current publishers on this topic string. One of the attributes returned is the Active Connection ID (ACTCONN) which links to DISPLAY CONN which shows you the details about that specific application.



Subscriptions

Subscription considerations

- Durability
- Destinations
- Administration & Monitoring
- Wildcards and Selectors
- Topic Aliasing and Topic tree isolation

Subscription Considerations

N

Subscriptions in publish/subscribe are really where the action is. Publishers make data available to consumers interested in that data. But subscribers decide what data to register interest in, and how that data will be delivered to them. Subscribers have a tremendous amount of control over what publications will be delivered to them, and by what means. This is tempered of course by security options that can be used to limit what topic(s) a subscriber can access, as well as other controls that can restrict some of the flexibility subscribers would otherwise have.

O

Subscribers can specify:

- How long-lived (or “durable”) their subscriptions will be.
- What kind of queue (“destination”) will be associated with a subscription, and who will be responsible for managing it (the queue manager, or the subscriber themselves).
- Whether to subscribe to a specific topic, or to a set of topics (wildcards)
- Whether to accept all publications on the subscribed-to topic(s), or only a subset (selectors)
- How to insulate their subscription from changes to the topic tree structure (topic tree isolation)

T

E

S

Durability of Subscriptions

- By default, subscriptions are Non-durable
 - ▶ Removed when application disconnects
- Can be made durable
 - ▶ Still exists while application is off-line
- Admin-Defined subscriptions always durable

■ Durable or Non-durable?

Application Requirement	Choose
Must not miss a beat	Durable
Can tolerate gaps in publication stream	Non-durable
Publications must be stored while app is off-line	Durable
Don't waste resources when app is off-line	Non-durable

Be Aware: Persistent messages delivered to a non-durable subscription are not recoverable!

- ▶ Still have Persistent flag turned on
- ▶ But not written to the MQ Log

Durability of Subscriptions

N

By default, subscriptions are made non-durably. This means that when the application disconnects from the queue manager, the subscription is removed and no more publications are sent to that subscription.

O

You can also make a subscription durable. This means that the subscription continues to exist even while the application is disconnected from the queue manager; publications that satisfy the subscription continue to be delivered to the subscription's destination and are stored there until the subscribing application reconnects and picks them up.

T

Whether you use a durable subscription or a non-durable subscription depends on the requirements of your application. Some of the criteria to take into account is shown on this slide.

E

Some things to note here:

- Subscriptions can be defined by the MQ administrator. This ability will be explored further in this presentation. By definition, an admin-defined subscription is always durable.
- Non-durable subscriptions can disappear at any time – and so one would expect that only non-persistent messages would be delivered to the subscription queue. It is possible for publications that were marked as persistent to be delivered to a non-durable subscriber. But be aware that in such a case, the messages, although mark as persistent, are not in fact written to the MQ recovery log, and so will not survive if they are not consumed before the queue manager is stopped and restarted. And, they will be deleted should the client that created the subscription disconnect.

S

Subscription Destinations

- **Subscriptions use a queue to hold publications that have been delivered but not yet consumed**
 - ▶ Enables publishers and subscribers to operate at different speeds
 - ▶ Also enables subscribers to disconnect and resume later (if Durable)
- **These queues can be Managed or Unmanaged**
 - ▶ “Managed” means MQ is responsible for it
 - ▶ “Unmanaged” means the application is responsible
- **Managed Destination queues cannot be accessed directly**
 - ▶ Can only be accessed via the subscription
- **Unmanaged destinations provide more options to consumers**
 - ▶ Subscriber can choose what destination to use
 - But subscriber is responsible for it
 - Making sure it exists (or is created)
 - Explicitly MQOPEN'ng it prior to the MQSUB call
 - Ensuring appropriate authorities are specified
 - Ensuring all is tidied up when the subscription is deleted.

Subscription Destinations

N
O
T
E
S

- Subscriptions use a queue to hold publications that have been delivered but not yet consumed. Using a queue as an intermediary between publishers and subscribers enables publishers and subscribers to operate at different speeds. It also enables features such as the ability for a subscriber to disconnect and resume later (if Durable).
- The queue associated with a subscription can be either “Managed” or “Unmanaged”. “Managed” means the queue manager is responsible for it – allocating it when the subscription is registered, and cleaning it up when the subscription is deregistered. Using managed destinations, the subscribing application can deal with the topic it subscribed to, and delegate the handling of the queue to MQ itself. On return from the MQSUB call your application is given two handles, an hSub and an hObj. hSub is the handle to the subscription, and the hObj is the handle which you can consume publications from using MQGET.
- An “Unmanaged” destination means the queue manager is NOT responsible for managing it on the applications behalf – the application is responsible for managing it. This allows a subscribing application to specify the queue they want the publications to be stored in for this subscription. To do this, you provide the MQSUB call with an hObj on input, so you must first MQOPEN the queue you wish to use (meaning it must already exist if not a dynamic queue), provide the resultant handle to the MQSUB and then you can consume from that queue. “Unmanaged” destinations are a bit more work for the subscribing application to manage, as it will be responsible for making sure it exists, opening it separately from the MQSUB call, and making sure the queue is disposed of properly when no longer needed (if appropriate).

Unmanaged Destinations – Things to consider

- **Unmanaged destinations allow many subscriptions to share a common queue**
 - ▶ To the same or different topics
 - ▶ Consumers effectively become QueueConsumers
 - ▶ Consumers can be a separate process from the one that created the subscription
 - Administratively defined subscriptions are an example of this
- **Security:**
 - ▶ For unmanaged destination queues, the subscriber must have **PUT** authority on the queue
 - ▶ May seem counterintuitive – why should a message consumer need PUT authority?
 - ▶ Reason is that the subscriber is requesting messages be put on its behalf
 - So it's reasonable the subscriber have authority to make such a request

Be Aware: The JMS Specification does not support the concept of Unmanaged Destinations!

- ▶ Use Admin-defined subscriptions to achieve something similar
- ▶ Consumers then are standard queue consumers

Unmanaged Destinations – Things to consider

N

- Unmanaged destinations may sound undesirable due to the additional work the application must perform, but there are scenarios where an application may want the greater degree of control over the destination that this allows. For example, unmanaged destinations allow multiple subscriptions to share a common queue. These can be subscriptions to the same or to different topics. Consumers effectively become QueueConsumers, reading messages directly off the queue and not dealing with the subscription side of things. Admin-defined subscriptions are another example where an unmanaged destination may be used.

O

- There are security considerations when using both managed and unmanaged destinations. For managed destinations, the foremost is probably that consumers cannot connect directly to the queue and consume messages – access to a managed destination is only available to the subscriber who registered the subscription, and it can only be accessed via the subscription.

T

- For unmanaged destination queues, the subscriber must have authority to PUT messages onto the queue. This may seem counterintuitive – why should a message *consumer* need PUT authority? But remember what is taking place here – the subscriber is authorizing the queue manager to put publications to the queue on its behalf. So it follows that the subscriber must have the authority to make such a request.

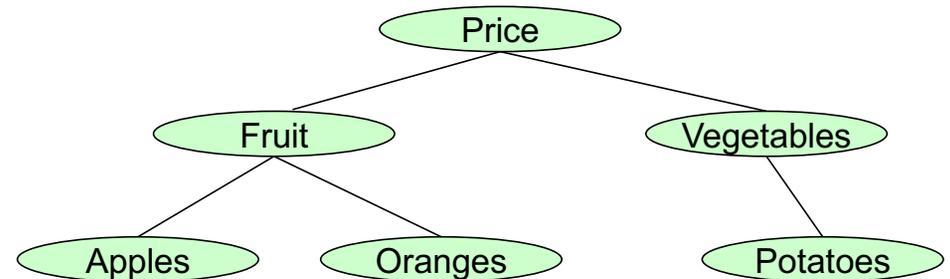
E

- For JMS subscribers, note that the JMS Specification does not support the concept of Unmanaged Destinations. In JMS, one is a QueueConsumer or a TopicConsumer; and if the latter, the consumer deals with the subscription, not directly with the destination. But while a JMS TopicConsumer cannot explicitly make use of an unmanaged destination, this can be done by creating a proxy, in the form of an Admin-defined subscription, which in turn points to an unmanaged destination. JMS QueueConsumers can now connect to this queue directly and consume messages, providing a workaround for JMS applications wanting to take advantage of using unmanaged destinations.

S

Administrative Subscriptions

- **Subscribers are typically applications**
 - ▶ But administrators can create them as well
- **Many use cases for creating such a proxy**
 - ▶ Feed publications to a queue consumer
 - ▶ Create a publication audit trail
 - ▶ Capture publications that matched no subscribers
 - ▶ Turn a message putter into a publisher
 - ▶ ...etc
- **Not MQ objects**
 - ▶ Different from queues, topics, etc
 - ▶ Stored in SYSTEM.DURABLE.SUBSCRIBER.QUEUE



```
Command Prompt - runmqsc TEST1
Starting MQSC for queue manager TEST1.
DEFINE SUB(GET.FRUIT.PRICE.CHANGES)
        TOPICSTR('Price/Fruit/#') DEST(FRUIT.PRICE.Q)
DISPLAY SUB(FRUIT.PRICE.CHANGES)
AMQ8096: IBM MQ subscription inquired.
SUB(FRUIT.PRICE.CHANGES) TOPICSTR(Price/Fruit/#)
DEST(FRUIT.PRICE.Q) DURABLE(YES)
SUBTYPE(ADMIN) SUBUSER(mqadmin)
```

Administrative Subscriptions

N

We've mentioned already that you can create subscriptions using `runmqsc`. Subscribers are typically applications, but there can be very good reasons for administrators to create them as well. By doing so you are effectively creating a proxy subscriber and can do so on behalf of an application that cannot do so on its own (because it is already written as a `QueueConsumer`, etc). Or, you might need to create an audit trail of publications published to a specific topic or set of topics. There are also problems that can be solved using such subscriptions: You can configure one to monitor a specific topic or set of topics for publications for which there were no other subscribers. Most of the examples in this presentation are demonstrated using admin-created subscriptions.

O

One thing to point out is that unlike Queues, Topic definitions, etc, Subscription definitions are not MQ Objects. If they are durable, they are persisted in a queue (`SYSTEM.DURABLE.SUBSCRIBER.QUEUE`) and so will survive a restart, can be exported, etc, but they are not present in the MQ Object Catalog.

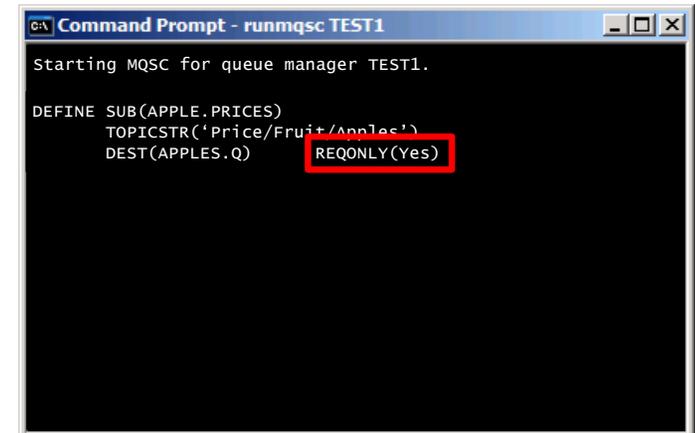
T

E

S

”Pause” a Subscription

- **Administratively-defined subscriptions decouple consumer from subscription**
 - ▶ Creation of subscription requires coordination with deployment of queue consumer
 - ▶ Problem with queue consumer means messages pile up
- **Use REQONLY(Yes) to indicate subscription will request message delivery**
 - ▶ Intended for use with Retained publications
 - ▶ Admin subscription will never issue request
 - Subscription effectively paused
 - Alter to REQONLY(No) to resume



```
Command Prompt - runmqsc TEST1
Starting MQSC for queue manager TEST1.

DEFINE SUB(APPLE.PRICES)
  TOPICSTR('Price/Fruit/Apples')
  DEST(APPLES.Q) REQONLY(Yes)
```

"Pause" a Subscription

N

With an administratively-defined subscription, you have decoupled the message consumer from the subscription. This can sometimes lead to problems, as the consumer may not be ready when you create the subscription, or may go away without your knowledge, etc, resulting in messages piling up on the queue. You may see this happening but not be sure whether this is a temporary condition or not. How can you stop delivery of messages to the subscription queue until you've sorted out what the problem is?

O

You can delete the subscription of course, but you may not want to do so. You could put-inhibit the queue, but if the queue is used by other message producers or publications, they will be affected as well. How can you just stop, or "pause", the subscription in question?

T

There is no "Pause" option, but there is a trick you can do to achieve the same thing – you can alter the subscription to receive publications only on request. This is normally an option used to retrieve retained publications, but it can be used to advantage here, as an admin-defined subscription is just an object and so will never issue any such requests – it will just sit there waiting for publication delivery to be resumed.

E

To do this, alter the subscription and specify REQONLY(YES). Publications will no longer be delivered to the queue by this subscription, but the queue can continue to be used by other producers or consumers. Once the problem is sorted out and you are ready to resume message delivery to the subscription alter the subscription once again to REQONLY(NO) to resume publication delivery.

S

Question: What if there are no Subscribers?

- **With Publish/Subscribe, standard operation is to give no indication where there are no subscribers**
 - ▶ With Point-to-Point messaging, a putter always knows there is a queue present
- **This may not be desirable in all cases**
 - ▶ An application might like to know that no one is interested in what it is publishing
 - ▶ A “safety net” to ensure publications for which there are no subscribers won’t simply disappear into the ether may be desired
 - ▶ A kind of “Dead Letter Queue” for unwanted publications
- **IBM MQ does provide options for dealing with this possibility**

Question: What if there are no Subscribers?

N

When putting a message to a queue, a successful result means the message is in the queue. When publishing to a topic, things are not quite that straightforward. With Publish/Subscribe, standard operation is to give no indication how many subscribers received the publication. That includes the situation no subscribers were present.

O

This is a reasonable behavior, given that the publisher is suppose to be decoupled from any subscribers. But from a practical standpoint, there are situations where a publisher may want to know that someone is interested in what it is publishing, and may want to take some kind of action if no subscribers are present. Similarly, perhaps there should be subscriber(s) present, but for some reason they are not receiving any publications. How can this be detected and handled?

T

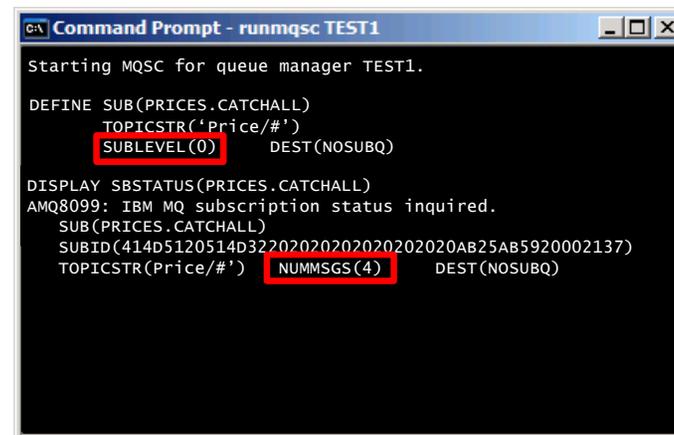
IBM MQ does provide options for dealing with this possibility.

E

S

Handling Publications for which there are no Subscribers

- A “default” subscriber can be created
 - ▶ Subscriptions with a SubLevel level of “0” will get matching publications only when no other subscribers were found
- Publishers can specify a PMO for a warning to be returned if there were no subscribers
 - ▶ pmo.Options |= MQPMO_WARN_IF_NO_SUBS_MATCHED
 - ▶ MQRC 2550 (“NO_SUBS_MATCHED”) returned if no subscribers
- **Note! For JMS applications!**
 - ▶ The JMS Specification does not support an equivalent to MQPMO_WARN_IF_NO_SUBS_MATCHED
 - ▶ So for JMS publishers, a SubLevel(0) subscription may be useful!



```
Command Prompt - runmqsc TEST1
Starting MQSC for queue manager TEST1.
DEFINE SUB(PRICES.CATCHALL)
  TOPICSTR('Price/#')
  SUBLEVEL(0) DEST(NOSUBQ)
DISPLAY SBSTATUS(PRICES.CATCHALL)
AMQ8099: IBM MQ subscription status inquired.
SUB(PRICES.CATCHALL)
SUBID(414D5120514D32202020202020202020AB25AB5920002137)
TOPICSTR(Price/#') NUMMSGS(4) DEST(NOSUBQ)
```

Handling Publications for which there are no Subscribers

N

Two possible approaches can be used if a publisher wants to know whether there are no subscribers, and/or wants to ensure that in such a situation their publications do not just disappear into the ether.

O

For MQI publishers, there is a PM option to request a warning be returned from the MQPut if there were no subscribers (MQPMO_WARN_IF_NO_SUBS_MATCHED). If this PMO is set, then MQRC 2550 ("NO_SUBS_MATCHED") will be returned if no subscribers matched the publication, and the publisher can then decide what action, if any, it wants to take.

T

As this PMO is specific to IBM MQ, JMS applications cannot set this and so cannot be made directly aware that no subscribers were interested in what is being published. But there is another approach that can be used - one that will work for any kind of publisher.

E

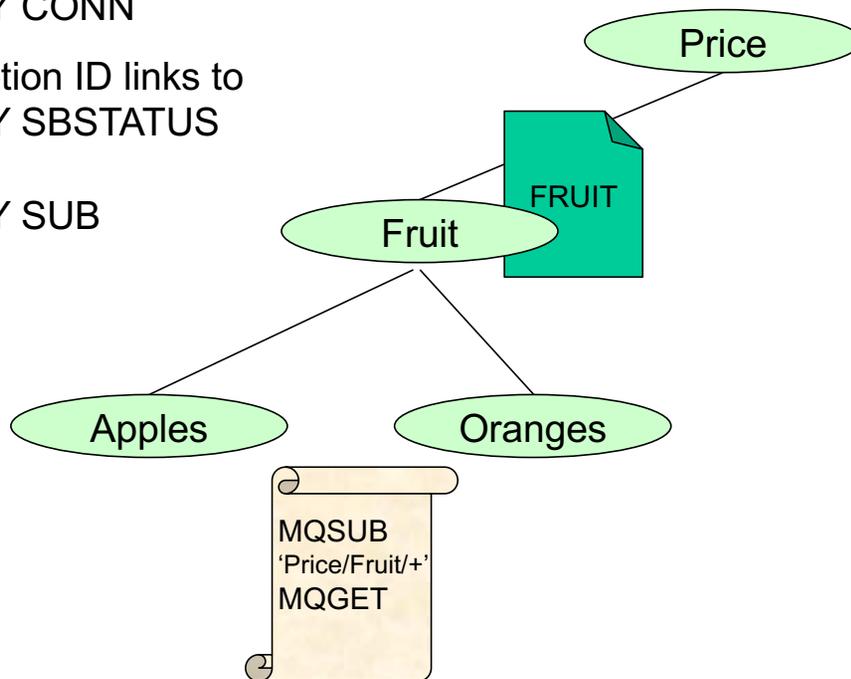
A "default" subscription can be created, either administratively, or by the application itself, to subscribe to the topic(s) in question, but setting the subscription level to zero. This is a special subscription level value (the range is usually 1 – 9). By creating a subscription with SUBLEVEL(0), publications will only be delivered to this subscriber if there were no other matching subscribers to the same topic(s). Thus this can serve as a DLQ of sorts for publications that had no interested parties subscribed.

S

Monitoring your Subscriptions

Connection ID links to
DISPLAY CONN

Subscription ID links to
DISPLAY SBSTATUS
and
DISPLAY SUB



TOPIC attributes

DURSUB
SUB
SUBSCOPE
PROXYSUB
WILDCARD

```
Command Prompt - runmqsc TEST1
DIS TPSTATUS('Price/Fruit/+') TYPE(SUB) ALL
AMQ8754: Display topic status details.
TOPICSTR(Price/Fruit/Oranges)
SUBID(414D51205445535431202020202020202020832AC44720013D07)
SUBUSER(chrisfra) RESMDATE(2017-02-26)
RESMTIME(18:53:35) LMSGDATE(2017-02-26)
LMSGTIME(18:53:41) DURABLE(NO)
ACTCONN(414D51435445535431202020202020202020832AC44720013D05)
NUMMSGS(2) SUBTYPE(API)
AMQ8754: Display topic status details.
TOPICSTR(Price/Fruit/Apples)
SUBID(414D51205445535431202020202020202020832AC44720013D07)
SUBUSER(chrisfra) RESMDATE(2017-02-26)
RESMTIME(18:53:35) LMSGDATE(2017-02-26)
LMSGTIME(18:53:41) DURABLE(NO)
ACTCONN(414D51435445535431202020202020202020832AC44720013D05)
NUMMSGS(2) SUBTYPE(API)
```

Monitoring your Subscriptions

N

O

T

E

S

- There are a few attributes on the topic object that are relevant to subscribers. We will look at those here along with the topic status display that shows information about subscribers.
- The TOPIC attribute DURSUB determines whether the creation of durable subscriptions is allowed at this point in the topic tree. If set to NO, and MQSUB using MQSO_DURABLE will fail with MQRC_DURABILITY_NOT_ALLOWED. The attribute SUB determines whether subscribing is allowed at this point in the topic tree at all. If set to DISABLED, an MQSUB call will fail with MQRC_SUB_INHIBITED. WILDCARD is a special attribute to block the propagation of subscriptions to very generic wildcarded subscriptions, such as MQSUB('#') where you don't really want portions of your topic tree exposed to such subscribers. It doesn't have an ASPARENT value as it only applies at that specific point in the topic tree.
- As already discussed, with the resolution of ASPARENT values, the object that finally resolved the value may be further up the topic tree than the point at which you are publishing.

Subscriptions – two perspectives

N
O
T
E
S

Two different status commands can be used to see the state of subscribers from two different perspectives.

Using `DISPLAY TPSTATUS TYPE(SUB)` you can see the details of the current subscribers on this topic string. One of the attributes returned is the Active Connection ID (`ACTCONN`) which links to `DISPLAY CONN` which shows you the details about that specific application. You'll note that our single subscription to 'Price/Fruit/+' has shown up subscribers on two topic strings. This is because this display is shown from the perspective of the topic string.

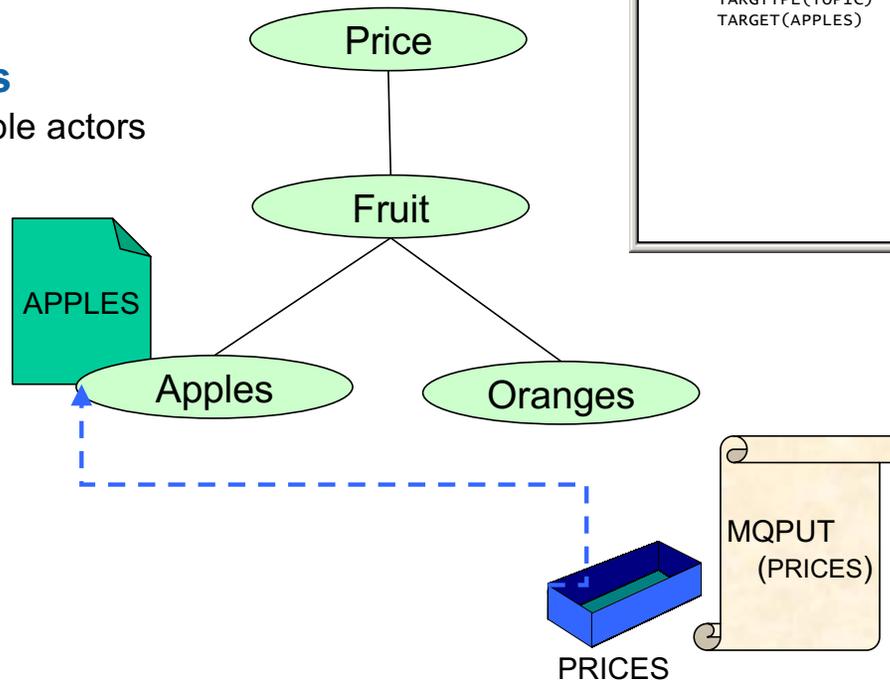
The Subscription ID (`SUBID`) value returned by `DISPLAY TPSTATUS TYPE(SUB)` can be used as input to `DISPLAY SBSTATUS` where we will see a single subscription with that ID since the perspective of that display is the subscription.



Topic Aliasing - What is it?

Queue Aliasing Topics

- Existing applications may want to leverage Pub/Sub
 - ▶ But not want to change their code
- May want to publish MQ Events
 - ▶ So they can be consumed by multiple actors
- There are restrictions:
 - ▶ Only works if producer and consumer were not using the exact same queue



```
Command Prompt - runmqsc TEST1
Starting MQSC for queue manager TEST1.
DEFINE TOPIC(APPLES)
TOPICSTR('Price/Fruit/Apples')
DEFINE QALIAS(PRICES)
TARGETTYPE(TOPIC)
TARGET(APPLES)
```

Alias Queues

N

There can be a desire to "convert" existing point-to-point applications to act as publishers or subscribers. We already discussed how to administratively create a proxy subscription to direct publications from one or more topics to a queue, from which a traditional QueueConsumer application can pull messages. What about the publishing side of things? Can an administrator achieve the same thing for a message producer?

O

This can be done by changing the queue that the putting application uses into an alias queue which points to a topic, which will result in that queue putter effectively becoming a publishing application.

Creating an administrative subscription (as we have just seen) and requesting that publication are sent to the original getting application's queue joins the two original application up again, but now via publish/subscribe.

T

One thing to note, this will only work if the point-to-point producer and point-to-point consumer were not previously using exactly the same physical queue. If they were you might first want to convert the putter to use an alias queue targeting the getters queue, and then from there convert to publish/subscribe.

E

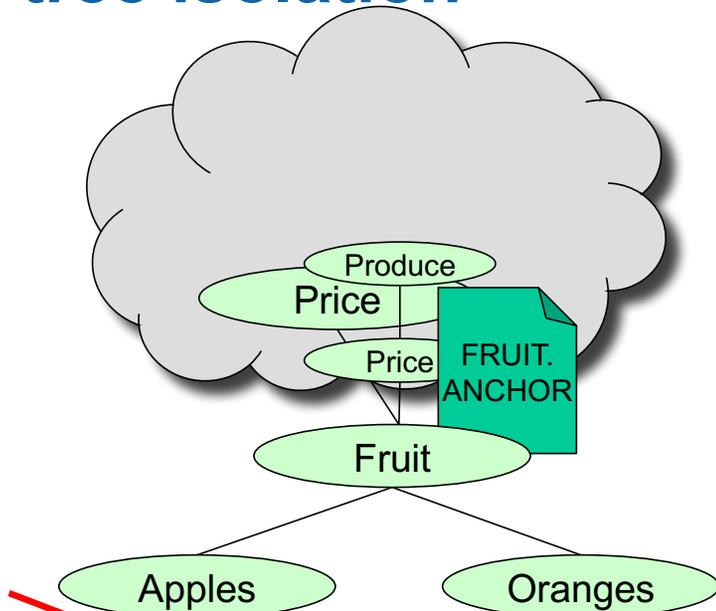
Now that you are using publish/subscribe, other interested parties can also subscribe to this topic without conflict on the getting queue or complicated logic in the putting application.

S



Topic Tree Isolation

Topic tree isolation



```
Command Prompt - runmqsc TEST1
Starting MQSC for queue manager TEST1.
DEFINE TOPIC(FRUIT.ANCHOR)
TOPICSTR('Price/Fruit')

DEFINE TOPIC(FRUIT.ANCHOR)
TOPICSTR('Produce/Price/Fruit')
```

ObjDesc.ObjectString.VSPtr
ObjDesc.ObjectString.VSLength

?

Topic tree isolation

N

When developing a Publish/subscribe application you may not yet have fully designed your topic string hierarchy structure. You may know that your application needs to deal with the price of fruit (as in our example) and that your topic strings will all end with '...Fruit/Apples' or '...Fruit/Oranges' but you do not yet know whether the full topic string will simply be 'Price/Fruit/...' or 'Produce/Price/Fruit...' or something completely different - so you wish to avoid hard-coding into your application a topic string that may change.

O

You can isolate your application from changes such as this, by providing an anchor point in the topic tree as a topic object and the portion of the topic string that your application is designing to be appended to the end. Doing this will isolate your application from future changes in the design of the full topic tree structure.

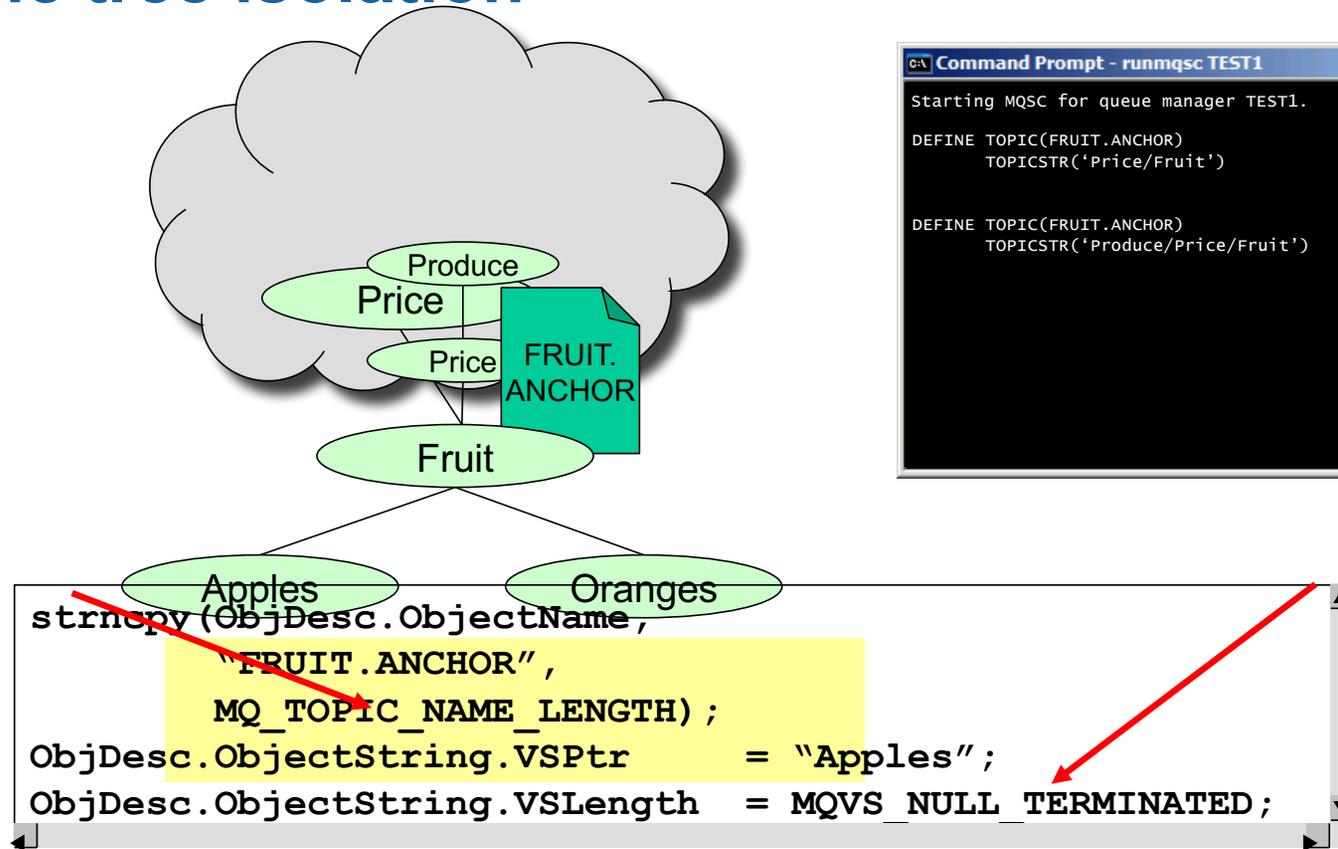
T

We do this by putting the anchor point topic object name in the MQOD.ObjectName and putting the portion of the topic string that goes on the end in the MQOD.ObjectString field. By doing this the queue manager will look up the topic object to find the associated topic string and then concatenate the two parts together to form your full topic string.

E

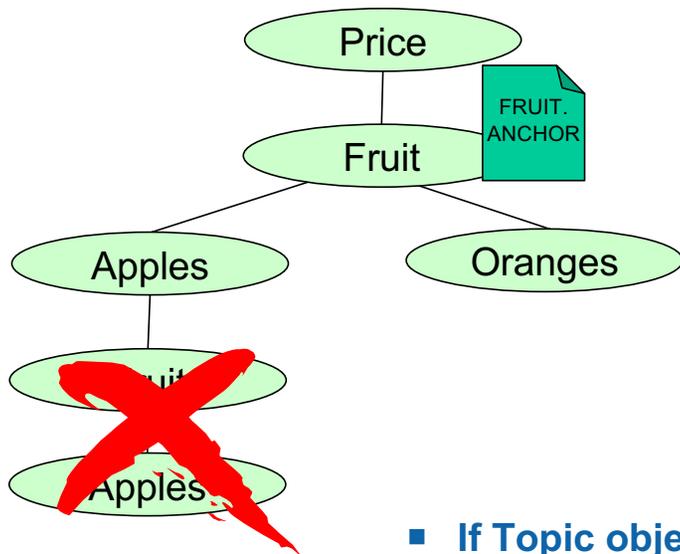
S

Topic tree isolation



```
Command Prompt - runmqsc TEST1  
Starting MQSC for queue manager TEST1.  
DEFINE TOPIC(FRUIT.ANCHOR)  
    TOPICSTR('Price/Fruit')  
  
DEFINE TOPIC(FRUIT.ANCHOR)  
    TOPICSTR('Produce/Price/Fruit')
```

Topic tree isolation – What can go wrong?



```
Command Prompt - runmqsc TEST1
Starting MQSC for queue manager TEST1.
DEFINE TOPIC(FRUIT.ANCHOR)
  TOPICSTR('Price/Fruit')

DEFINE SUB(APPLE.PRICES)
  TOPICOBJ(FRUIT.ANCHOR)
  TOPICSTR('Price/Fruit/Apples')

DISPLAY SUB(APPLE.PRICES)
AMQ8096: IBM MQ subscription inquired.
SUB(APPLE.PRICES) TOPICOBJ(FRUIT.ANCHOR)
TOPICSTR(Price/Fruit/Price/Fruit/Apples)
```

- If Topic object specified on subscription, Topic string always relative to it
- For topic tree isolation:
 - ▶ Specify Topic object only
 - or
 - ▶ Specify Topic object + subtopic from that point in the topic tree

Topic tree isolation – What can go wrong?

N

When doing this, it is important to correctly specify the topic string correctly. A common mistake that is easy to make is show here.

Define the topic object, specifying the topic string the object is to represent:

```
DEFINE TOPIC(FRUIT.ANCHOR) TOPICSTR('Price/Fruit')
```

O

Now the subscription is created, either programmatically (by putting the anchor point topic object name in MQOD.ObjectName and the topic string in MQOD.ObjectString), or as shown here in an Admin-defined subscription:

```
DEFINE SUB(APPLE.PRICES) TOPICOBJ(FRUIT.ANCHOR) TOPICSTR('Price/Fruit/Apples')
```

T

This would seem correct, and would not raise an error. However, it will not receive any messages published to “Price/Fruit/Apples”. If you display the subscription, the error becomes apparent:

```
DISPLAY SUB(APPLE.PRICES)
```

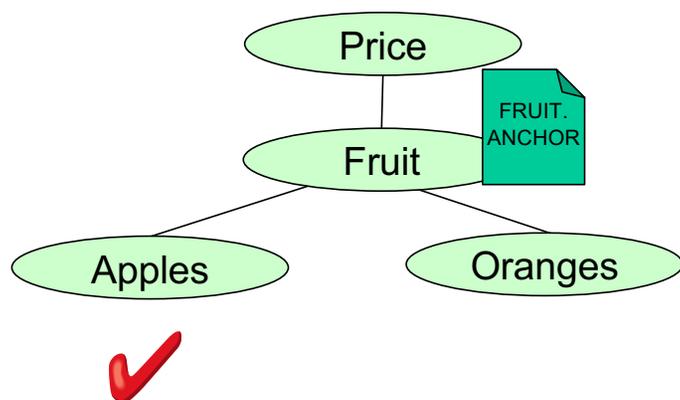
```
SUB(APPLE.PRICES) TOPICOBJ(FRUIT.ANCHOR) TOPICSTR(Price/Fruit/Price/Fruit/Apples)
```

E

You obviously did not intend for the subscription to use a topic string of “Price/Fruit/Price/Fruit/Apples”! So take care to remember that when using this technique for topic tree isolation, the subscription specifies either *only* the anchor point topic object name, or the anchor point topic object name and then *only* the portion of the topic string that extends beyond the anchor point (in this case “Apples”).

S

Topic tree isolation – Getting it right



```
Command Prompt - runmqsc TEST1
Starting MQSC for queue manager TEST1.
DEFINE TOPIC(FRUIT.ANCHOR)
  TOPICSTR('Price/Fruit')

DEFINE SUB(APPLE.PRICES)
  TOPICOBJ(FRUIT.ANCHOR)
  TOPICSTR('Apples')

DISPLAY SUB(APPLE.PRICES)
AMQ8096: IBM MQ subscription inquired.
SUB(APPLE.PRICES) TOPICOBJ(FRUIT.ANCHOR)
TOPICSTR(Price/Fruit/Apples)
```

- If Topic object specified, Topic string always relative to it
- For topic tree isolation:
 - ▶ Specify Topic object only
 - or
 - ▶ Specify Topic object + subtopic from that point in the topic tree

Topic tree isolation – Getting it right

N

O

T

E

S

This example shows the subscription using the anchor point topic object name, and then only the portion of the topic string that extends beyond the anchor point (in this case “Apples”).

JMS and Topic Tree Isolation

- The JMS specification does not have the concept of topic tree isolation
- IBM MQ does provide a means of using this for JMS applications
- Leverages existing properties for stream queues

► For Destinations

Destinations					
Filter: Standard for JMS Destination					
Name	Class name	Messaging provi...	Topic	Publication stream	Broker version
Fruit.Anchor	MQTopic	WebSphere MQ ...	Apples	FRUIT.ANCHOR	v2

► For Connection Factories

Connection Factories					
Filter: Standard for JMS Connection Factory					
Name	Class name	Messaging provider	Provider version	Publication stream	Broker queue manager
All.Fruit.Prices	MQConnectionFactory	WebSphere MQ	8	FRUIT.ANCHOR	QM2

JMS and Topic Tree Isolation

N

Unfortunately, Topic objects (or Topic object definitions) are not the same thing that the JMS specification refers to when it uses the term “Topic object”. In JMS, there is no concept of topic anchors, and no concept of two-factor topic names. So how would a JMS application take advantage of Topic tree isolation in IBM MQ?

O

It is possible to take advantage of this in IBM MQ, using a ConnectionFactory or Destination property that was available in older versions of MQ to support the implementation of publish/subscribe *streams*. Since MQ V7, the native publish/subscribe implementation in MQ no longer uses streams (although they are still supported for backward compatibility). But if you define an anchor point Topic object, and then specify that object name (“FRUIT.ANCHOR” in our example) as the “Publication Stream” value in either the ConnectionFactory or Destination definition (property name BROKERPUBQ), then MQ will behind-the-scenes create the topic name for use by the JMS application.

T

E

S



Wildcard Best Practices

Wildcards

- MQ supports two wildcard schemes

- Character based

- ▶ '*' – matches many characters

Price/Fruit/* Matches on all fruit

Price/Fruit/Ap* Matches on Apples, Apricots, etc

Price/Fruit/*berries Matches on Blueberries, Strawberries, etc

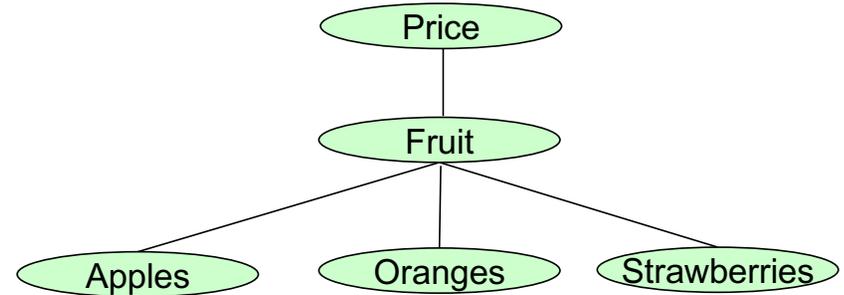
- ▶ '?' – matches single character

Price/Fruit/????berries Matches on Blueberries and Raspberries

But not on Blackberries or Strawberries

- Character-matching allows patterns outside the topic hierarchy

- ▶ But can be slower, since matching is done at the character level



Wildcards

N
O
T
E
S

IBM MQ supports two different wildcard schemes for use by subscribers (publishers do not use wildcards).

The first, and oldest, are “character-based” wildcards. This was the scheme used in MQ publish/subscribe prior to MQ V7, and continues to be supported today for backward compatibility. Character-based wildcard schemes are common with other products (databases, etc) and provide a great deal of flexibility when creating masks to apply against a source string, or in our case, the topic string. The supporting slide shows some examples of how character-based wildcards might be used on some sample topic strings.

Character-based wildcards allow masks to be created that apply to all characters in the topic string – they are not limited to just wildcarding the at the topic hierarchy. This means they are slower to evaluate than topic element-based wildcards (which we’ll discuss next), since every character in the topic string must be evaluated.

Wildcards

■ Topic element based

- ▶ '#' – matches one or more levels in the hierarchy

Price/Fruit/# Matches on all fruit

Price/Fruit/Ap* But no equivalent to this

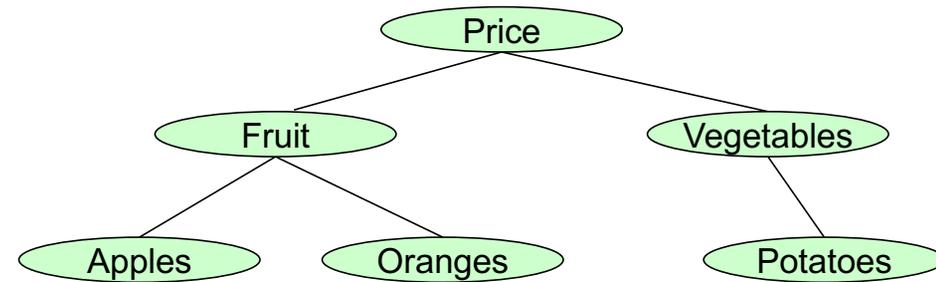
Price/Fruit/*berries Or to this

- ▶ '+' – matches a single level in the hierarchy

Price/+/# Matches on all Fruits, Vegetables, etc

■ Topic element matching only allows filtering on the topic hierarchy

- ▶ But can be much faster, as the topic string is not treated as a character string for matching purposes



Wildcards

N
O
T
E
S

Topic element-based wildcards are often seen in publish/subscribe implementations, and are preferable to character-based wildcards for performance reasons. But there are things that can be done using character-based wildcards that are not possible with Topic element-based wildcards (see examples on slide). For this reason, you will want to design your topic tree structure to minimize or eliminate the need to use character-based wildcards – not that their use is forbidden, but because you will realize a performance benefit by using Topic element-based wildcards whenever possible.

Wildcards – How implemented

- **Use of wildcards has performance implications**

- ▶ Resolved when subscription created
- ▶ Each matching node in tree effectively has a subscription

Price/Fruit/Apples

Price/Fruit/#

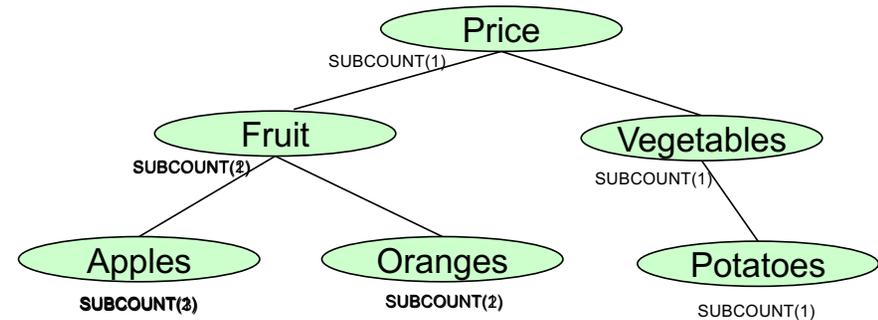
Price/#

- **Other implications of using wildcards:**

- ▶ Can result in topic tree nodes being held even if seldom used
 - Since the SUBCOUNT on all matching nodes will be > 0
 - Can increase publication time

- **Use but use wisely**

- ▶ Monitor using DISPLAY TPSTATUS
- ▶ Can be blocked using DEFINE TOPIC(...) WILDCARD(BLOCK)
 - But only applies to subscriptions created after the change!



Wildcards – How implemented

N

There are performance considerations to take into account with the use of wildcards. Consider these examples:

Price/#
Price/Fruit/#
Price/Fruit/Apples

O

One could assume there are subscriptions at three points in the topic tree: At “Price”, “Price/Fruit” and “Price/Fruit/Apples”. But there are actually ten subscriptions: Three at “Price/Fruit/Apples”, two each at “Price/Fruit” and “Price/Fruit/Oranges”, and one at each of the remaining topic tree nodes. You can see this by using DISPLAY TPSTATUS(‘Price/#’) and looking at the SUBCOUNT value for each returned topic tree node under “Price”.

T

Why is this of significance? First, with a very wide topic tree (e.g. thousands of leaf nodes), this illustrates the reason why registering a subscription using a wildcard might take more time than expected, But also, it shows the potential impact of a large number of wildcarded subscriptions, especially on a very deep or very wide topic tree.

E

The queue manager will try and free the storage occupied by topic tree nodes that have been idle for some period of time. But the use of wildcards in subscriptions can hamper this, as topic tree nodes with a SUBCOUNT > 0 will be seen as active, even if nothing is even published to some of them!

S

This is not saying that the use of wildcards should be avoided. The ability to select on a category of topics is a very powerful capability. But it can have implications, especially on a very deep or very wide topic tree. So make use of them, use them wisely.

Wildcards – Be Careful

- **Scheme specified when subscription created**
 - ▶ Must be sure topic string matches scheme specified
- **Wildcard scheme of CHAR does not recognize '+' and '#' as wildcard characters**
 - ▶ Treats these as actually part of the topic string
- **Best to use only one wildcard scheme**
 - ▶ Preferably topic-level wildcards for performance reasons
 - ▶ Must be sure topic string matches scheme specified

```
Command Prompt - runmqsc TEST1
Starting MQSC for queue manager TEST1.
DEFINE SUB(ALL.FRUIT.PRICES)
        TOPICSTR('Price/Fruit+') WSCHEMA(CHAR)

DISPLAY SUB(ALL.FRUIT.PRICES)
AMQ8096: IBM MQ subscription inquired.
        SUB(ALL.FRUIT.PRICES) TOPICSTR(Price/Fruit+)
        WSCHEMA(CHAR)

DISPLAY TPSTATUS('Price/Fruit/#')
AMQ8754: Display topic status details.
        TOPICSTR(Price/Fruit+)
        SUBCOUNT(1)
```

QM2 - Topic Status

Queue Manager: QM2

Topic status:

Topic string	Publish	Subscribe
[Empty]	Allowed	Allowed
Price	Allowed	Allowed
Fruit	Allowed	Allowed
+	Allowed	Allowed

Wildcards – Be Careful

N

As mentioned, IBM MQ supports two different wildcard schemes. But you must choose which scheme you will use when you create your subscription. You cannot use a mixture of the two schemes. So you must take care that you specify the wildcard scheme (WSHEMA in the example shown) that matches your actual wildcard usage.

O

Why is this important? Because the queue manager will interpret your topic string based on the scheme specified. If, as shown in this example, you specified WSCHEMA(Char) but then specified a topic string like "Price/Fruit/+", the subscription will register successfully, but publications will not be delivered to it, and it will not be obvious what went wrong. If you use DISPLAY TPSTATUS you will see what went wrong, if you have a sharp eye – MQExplorer will display Topic Status graphically and so makes the mistake easier to spot.

T

You can minimize the likelihood of such a mistake by only using Topic element level wildcards, and get a bit better performance to boot – but that is not always possible. So the best advice is probably to be vigilant,, and when creating subscriptions, whether programmatically or using DEFINE SUB, take care to be sure the wildcard characters used match the scheme specified.

E

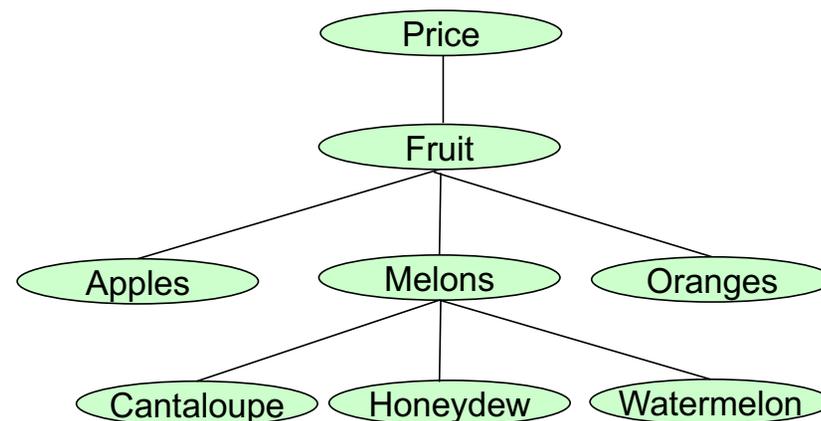
S



Using Message Selectors

Why Message Selectors?

- **Enables filtering at the publication level**
 - ▶ Filtering is done on message properties
 - Or message payload if using IIB
 - ▶ Reduces size and complexity of topic tree
 - ▶ Provide more granular filtering
 - ▶ Eliminate delivery of messages that would just be discarded
- **Performance considerations**
 - ▶ Selectors evaluated at publication time
 - Will impact MQPut performance



```
GA\ Command Prompt - runmqsc TEST1
Starting MQSC for queue manager TEST1.
DEFINE SUB(ALL.MELONS)
TOPICSTR('Price/Fruit/Melons/+')
DEFINE SUB(ONLY.WATERMELON)
TOPICSTR('Price/Fruit/Melons')
SELECTOR('Kind='watermelon'')
DISPLAY SUB(ONLY.WATERMELON)
AMQ8096: IBM MQ subscription inquired.
SUB(ONLY.WATERMELON) TOPICSTR(Price/Fruit/Melons)
SELECTOR(kind='watermelon') SELTYPE(STANDARD)
```

Message Selection

N

So far we have only looked at application's subscribing to all the messages on a particular topic string, or group of topic strings (via wildcards). But your application may wish to subscribe more specifically than that. Let's continue our example of fruit prices – imagine wanting to include different kinds of melons under the Fruit topic. You may decide that would make that level of the topic tree too wide (after all, there are also different kinds of Apples, Oranges, etc), and so decide to add another level to the topic tree, with Cantaloupe, Honeydew, Watermelon, etc under the "Melon" topic.

O

But wait! Since there are also different kinds of Apples, Oranges, and so on, doing this for all the different variants of each type of fruit will make the topic tree quite busy. So you may decide to keep your topic tree structure simpler, referring only to categories of Fruit, Vegetables, etc, with the different kinds of different fruits being identified by a message property specified by the publisher. Subscribers can then use that to filter on publications to be delivered to them by topic as well as by selecting a specific kind (or kinds) of melon, if they need to be that specific. If they do not, they can just omit the selector when creating the subscription, and consume all publications to "Price/Fruit/Melons".

T

What are the trade-offs of using this approach? By allowing filtering of individual publications, the subscriber can have the queue manager suppress delivery of messages that the application would just throw away anyway. This reduces network bandwidth and makes the subscribing application simpler to write. The trade-off is that the publisher must make the message properties available for selecting against (a bit more work for them), and the queue manager is taking on some of the burden of the subscribing application, since the message selectors are essentially a bit of application code that is embedded in the subscription.

E

The next few slides will explore ways to minimize the performance impact of this trade-off.

S

Message Selection – Performance

- **Selector statements executed at publication time**
 - ▶ If more than one subscriber has specified a selector, these are executed serially
- **This can have an impact on publisher (MQPut) performance**
 - ▶ Complexity of selectors also a factor
 - Consider these examples:

```
SELECTOR(sendingApp='TEST' AND eventReasonCode='NA')
```

```
SELECTOR(sendingApp='TEST' AND sendingInterface='ADT' AND msgTriggerEvent IN  
( 'A01','A02','A03','A04','A05','A06','A07','A08','A09','A10','A11','A12','A13','A38','A40','A45') AND sendingFacility IN  
( 'ALCLR','ALHRD','CHT1','CHT10','CHT12','CHT13','CHT14','CHT15','CHT2','CHT3','CHT4','CHT6','CHT7','CHT8','CHT9','  
CTROC','CTROE','CTROG','CTROJ','CTROM','DEWCD','EULHR','FAFHD','GI35','IRR01','IRR0M','IRRG0','IRRMb','IRROG  
'; 'IRROJ','KAKH','LASKR','MAMHR','MPMC','NFRTR','ONOCD','OpT1','OpT3','OpT9','OWOCD','ROALGI','ROBA','ROCCL  
'; 'ROCO','ROCOCRS9','RODA','ROEILD','ROEIOR','ROFF','ROFM','ROGO02','ROGO07','ROGO15','ROGOG12','ROGOG1  
9','ROGU','ROHI','ROHPC','ROHRS','ROMAGO','ROMBGI','ROMBOR','RONE','RONX','ROOP','ROPL','RORMC','ROSE','R  
OSI','ROSMC','ROSU','ROSX','SU01','SU02','SU03','SU04','SU05','SU06','SU07','SU08','SU09','SU10','SU11','SU12','SU13'  
, 'SU14','SU15','SU16','SU17','SU18','SU19','SU20','SU21','SU22','SU23','SU24','SU25','SU26','SU27','SU28','SU29','SU30','  
SU31','SU32','SU33','SU34','SU35','SU36','SU37','SU38','SU39','SU40','SU41','SU42','SU43','SU44','SU45','SU46','SU47','S  
U48','SU49','SU50','SU51','SU52','SU53','SU54','SU55','SU56','SU57','SU58','SU59','SU60','SU61','SU62','WBSED' ) )
```

Message Selection

N

Clearly, one thing to consider is the complexity of the message selector. Consider the two examples on the slide – there will obviously be a higher cost incurred executing the more complex selector. But if the subscribing application would just have to do the same sort of filtering anyway, isn't the cost moot?

Actually, it isn't. Message selectors are evaluated at the time of publication – not when the subscriber attempts to consume the message. That is a more efficient approach – why put messages to subscriber queues (logging them if they are persistent) only to discard them later? So it makes sense to evaluate selectors at publication time, and only put messages to the subscriber queue if they are selected for.

O

But consider the effect of having a non-trivial number of subscribers, each using a selector. When a message is published on a topic, each subscriber to that topic (whether explicit or wildcarded) must have its selector evaluated, before the MQPut can return control back to the publishing application. Normally, an MQPut, even of a persistent message, executes very quickly. But in the case of a publisher, the MQPut is really resulting in potentially many puts (to subscriber queues) and executing the selector for each subscriber that specified one. The result, as the number of subscribers to a topic grows, and/or the complexity of the selectors increases, can be increasingly long MQPut times when putting to a topic with such subscribers.

T

In a later slide we will cover things to take into account when using selectors, especially if you are experiencing long MQPut times you believe are due to the use of complex selectors.

E

S

Admin Subscriptions and Selectors

Be Careful!

- **API-defined subscription selectors validated against MQ or “extended” selection engine**
 - ▶ IBM Integration Bus (IIB) is the only supported extended selection engine
- **Admin-defined subscriptions get minimal selector validation**
 - ▶ If MQ rejects, subscription flagged as needed “extended validation”
 - ▶ But IIB not invoked to validate the selector
- **What happens at publication time?**
 - ▶ Publication fails because this is not a selector the MQ selection engine can digest

```
Command Prompt - runmqsc TEST1
Starting MQSC for queue manager TEST1.

DEFINE SUB(ONLY.WATERMELON)
  TOPICSTR('Price/Fruit/Melons')
  SELECTOR('kind='watermelon'')

AMQ8094: IBM MQ subscription created.

DISPLAY SUB(ONLY.WATERMELON)
AMQ8096: IBM MQ subscription inquired.
SUB(ONLY.WATERMELON) TOPICSTR(Price/Fruit/Melons)
SELECTOR(Kind='watermelon') SELTYPE(Standard)
```

```
Command Prompt - runmqsc TEST1
Starting MQSC for queue manager TEST1.

DEFINE SUB(ONLY.WATERMELON)
  TOPICSTR('Price/Fruit/Melons')
  SELECTOR('kind="watermelon"')

AMQ8094: IBM MQ subscription created.

DISPLAY SUB(ONLY.WATERMELON)
AMQ8096: IBM MQ subscription inquired.
SUB(ONLY.WATERMELON) TOPICSTR(Price/Fruit/Melons)
SELECTOR(Kind="watermelon") SELTYPE(EXTENDED)
```

Admin Subscriptions and Selectors

N

When using administratively-created subscriptions, there are some things to be careful of. One involves a difference between API-defined subscriptions and administratively-created subscriptions – specifically, how selectors are validated.

With API-defined subscriptions, selectors are evaluated for correctness at the time the subscription is created, and if the selector is not known to be valid by the selection engine in use (either MQ or IIB), then the MQSub request will fail.

O

But with administratively-created subscriptions, the same degree of validation is not performed. There is some basic validation performed (for matching quotes, parentheses, etc), and, if it passes that, then some simple tests are made, and MQ decides whether the selector will be executed by the “Standard” selection engine (the queue managers), or by the “Extended” selection engine (i.e. IIB). The potential problem here is that the queue manager will decide that IIB is to be used for evaluating message selectors even if IIB is not in use by this queue manager, or IIBs Extended Message Selection (EMS) feature is not enabled.

T

The example shows one possible scenario where this can occur – if the selector makes use of only single quotes, the MQ selection engine will be chosen, but if there is any use of double-quotes, the IIB selection engine will be assumed.

What happens at publication time if the Extended selection engine is to be used, but IIB is not present, or not configured for EMS? The publisher’s MQPut will fail, because this is not a selector the MQ selection engine can digest.

E

What does this mean for the MQ Admin? When defining a subscription that will include a selector, they must make sure, if MQ assigns a SELTYPE value of Extended, that IIB is actually present, that it is configured for EMS, and that the selector is tested to ensure that IIB can actually execute the selector, since it will not be validated when the subscription is created.

S

Selectors – Specify with Care

Don't use IIB for selection unnecessarily

- Selecting against message header fields (MQMD, RFH2, etc)
- Root.MQMD.* handled by MQ selection engine
- Root.<anything else> always assumes extended selection engine
- Can be controlled via qm.ini property:
TuningParameters:
ExtendedSelectionPrecedence=[0|1|2]

Where:
0 = Pass to IIB if required
1 = Never pass selection to IIB
2 = Always pass selection to IIB

```
Command Prompt - runmqsc TEST1
Starting MQSC for queue manager TEST1.
DEFINE SUB(TEST1)
  TOPICSTR('Price/Fruit/Apples')
  SELECTOR('Root.MQMD.ApplOriginData='Mcintosh'')
AMQ8094: IBM MQ subscription created.

DISPLAY SUB(TEST1)
AMQ8096: IBM MQ subscription inquired.
SUB(TEST1) TOPICSTR(Price/Fruit/Apples)
SELECTOR(Root.MQMD.ApplOriginData='Mcintosh'')
SELTYPE(STANDARD)
```

```
Command Prompt - runmqsc TEST1
Starting MQSC for queue manager TEST1.
DEFINE SUB(TEST2)
  TOPICSTR('Price/Fruit/Apples')
  SELECTOR('kind='Mcintosh'')
AMQ8094: IBM MQ subscription created.

DISPLAY SUB(TEST2)
AMQ8096: IBM MQ subscription inquired.
SUB(TEST1) TOPICSTR(Price/Fruit/Apples)
SELECTOR(kind='Mcintosh')
SELTYPE(Standard)
```

Selectors – Specify with Care

N

When using administratively-created subscriptions, you must also take care when selecting against message header properties.

If you are selecting against an MQMD property, a selector constructed like “Root.MQMD.<property>” will be handled by the queue managers selection engine. But if your selector tests against a property in any other MQ header (for example, ‘Root.MQRFH2.usr.kind=“Mcintosh”’) then the queue manager will always assume the Extended selection engine (IIB) is to be used. We already discussed what will happen if IIB is not actually in use, or configured for EMS. But here is a situation where IIB is present and is being used for extended message selection. IIB would be able to handle such a selector, but the concern in this case is that there is a higher cost associated with using IIB for message selection versus the queue manager - three to four (or more) times as much overhead. So it is preferable to use the queue managers selection engine whenever possible.

O

There is nothing wrong with using IIB for message selection – it is a great selection engine. But it not necessary to use IIB to select against a message property, such as the example shown. Rather than specifying this:

```
Selector('Root.MQRFH2.usr.kind="Mcintosh"')
```

Specify the selector this way instead:

```
Selector('kind="Mcintosh"')
```

T

This will result in the queue managers selection engine being used, rather than IIB.

Alternatively, there is a tuning parameter in qm.ini that can be used to control which selection engine to use (see slide). The downside of using this is that it is a queue-manager-wide property, and your goal is probably not to prevent IIB being used as a selection engine – but simply to prevent it being used when it doesn't need to be. In that case, the best approach to use is, when creating administratively-defined subscriptions, to ensure that SELTYPE(STANDARD) is what is used whenever possible.

E

S

Message Selection – Performance

- **If using message selectors, consider the following for minimizing the cost:**
 - ▶ Use MessageID or CoreIID as an alternative, if possible
 - ▶ Keep selectors as simple as possible
 - ▶ If selecting against multiple properties, construct selector so tests most likely to fail are first
 - ▶ If using IIB for content filtering, avoid using IIB for selecting against message properties

Message Selection – Performance

N

What can be done if you are experiencing long MQPut times you believe are due to the use of complex selectors? There are several possible approaches to minimize this if it is determined that using message selectors is appropriate:

1. Keep selectors as simple as possible
2. If possible, select against values carried in the MQMD CorrelID field (or other MQMD field). Selection can be performed much quicker when used with an MQMD field than when selecting against an arbitrary message property (like “Kind=Watermelon”).
3. When selecting against multiple properties, construct the selector so that the properties that are least likely to evaluate to True are evaluated first – this will cause the selector to return False more quickly and keep the selection overhead to a minimum.
4. Selection can also be done by QueueConsumers, not only TopicConsumers – so consider not using selectors on the subscriptions, and have the QueueConsumers specify the selector on MQOpen. This will shift the selection overhead from the publisher to the individual subscribers, while still keeping the selection logic out of the application. This may speed up the publisher, but will result in more messages being queued, some (perhaps many) of which will be subsequently discarded. So this approach must be carefully considered.
5. Split the message filtering, so that the properties most likely to fail are part of the selector (and so evaluated at publication time) while further additional filtering is done using a selector on the consuming side (see #4) and/or by the consuming application code. As with #4, this may speed up the publisher, while reducing the number of publications delivered and therefore reducing the number of total messages that end up being discarded.
6. If using IIB for Extended Message Selection (EMS, or “Content Filtering”), avoid doing so for elements that are not actually part of the message body (such as RFH2 properties, etc).

O

T

E

S



Publication Properties

Publication Properties



- **Messages can have metadata associated with them alongside the message payload**
 - ▶ These message properties can be useful in supplying additional data
- **Published messages have associated message properties**
 - ▶ These can be useful to subscribers/consumers
 - ▶ Subscribers can select against message properties
- **Can help the message consumer understand:**
 - ▶ "What topic was this message published to?"
 - ▶ "Who published this message?"
- **A few useful ones include:**
 - ▶ **MQTopicString** Topic string the message was originally published to
 - ▶ **PubAppIdentityData** Application Identity of publisher
 - ▶ **SubUserData** User Data associated with the subscription
 - ▶ **MQPubStrIntData** String/Integer data added by the publisher

Publication Properties

N

IBM MQ messages can have metadata associated with them in the form of message properties. These message properties can be used for any number of purposes. Applications can set user properties to whatever value they choose – this capability can be put to good use with publish/subscribe applications, in that publishers can set user properties containing values that subscribers may want to select against, and subscribing applications can then pick and choose the properties they select against to fine-tune the set of publications to be delivered to them.

O

IBM MQ itself will append properties to messages. In the case of publications, a number of standard message properties are appended to publications, and subscribers can use these to, among other things, gain some insight into the publication as well as the publishing application. For example, although publishers and subscribers are generally agnostic to one another, by using these properties a subscriber can answer such questions as “What topic was a particular message originally published to?”, or “What application published this message?”. The list of message properties appended to publications can be found here:

T

https://www.ibm.com/support/knowledgecenter/SSFKSJ_9.0.0/com.ibm.mq.dev.doc/q026640_.htm.

E

When multiple subscriptions point to the same destination, there can be a need for consumers to be able to differentiate between the publications themselves, perhaps by identifying the topic the message was published to, or by identifying the publisher of a given message. Although this reduces the degree of anonymity between publisher and subscriber, there can be a business need for doing so. In such cases, the following publication properties can be of use:

S

- MQTopicString** Topic string the message was originally published to
- PubApplIdentityData** Application Identity of publisher
- SubUserData** User Data associated with the subscription
- MQPubStrIntData** String/Integer data added by the publisher



Performance

Performance

- **Pub/Sub incurs additional latency compared to point-to-point messaging**
 - ▶ Approx. 15% throughput reduction (V8)
 - ▶ V8 substantially better than V7.5
- **Felt disproportionately by the publisher**
 - ▶ Publish (MQPut) time affected by the following:
 - Number of subscribers (esp. if more than 48)
 - Persistence
 - Selectors
 - Message delivery options
- **Subscribers**
 - ▶ Delivery can be delayed if publish time is long
 - ▶ Use of interceptors affects delivery to final subscriber

Performance

N

When using Publish/Subscribe, it is important to realize there will always be less throughput than with point-to-point. That's not a bad thing in and of itself – you are gaining more flexibility by using the publish/subscribe model. But it is worth noting that the throughput reduction (approx. 15% throughput reduction with V8 and V9, which is substantially better than previous releases) is not benefitting you if there is only likely ever to be a single subscriber.

O

Remember as well that this added latency is felt disproportionately by the publishing application – and that it increases as the number of subscribers to a given topic increases. IBM MQ can handle a large number of concurrent publishers and subscribers quite well, if they are well distributed across the topic tree. And, IBM MQ can support a very large number of topic tree nodes. So for the majority of cases, performance will likely be well within expectations.

T

But as this presentation discussed in detail, there are circumstances where this latency will be more apparent, including:

- When the number of subscribers to a specific topic is non-trivial (say, 10+)
- Where the publications are persistent and being put to durable subscriptions (since multiple messages are being put by that one MQPut)
- Where wildcards are being used inappropriately.
- Where message selectors are being used (since the selectors for all the subscribers to a specific topic are evaluated as part of the MQPut)

E

Being aware of the performance cost of various options available to applications using the publish/subscribe model, and how best to minimize those costs, is probably the best strategy for getting the most out of using publish/subscribe with IBM MQ.

S



Summary

Summary

- **Give thought to the size and shape of your topic tree**
 - ▶ Topic objects enable granular administrative control of attributes, security, message delivery
- **Publishers**
 - ▶ MQPut times affected by subscriber numbers, use of selectors, etc
- **Subscribers**
 - ▶ Several options for filtering, selecting messages – can simplify application code
 - ▶ Administrative subscriptions provide added flexibility
- **JMS can limit your use of some features available thorough the MQI**
 - ▶ Topic tree isolation, Unmanaged destinations, Message delivery
 - ▶ IBM MQ provides workarounds for these
- **Where's my message?**
 - ▶ Discussed ways to identify publications which had no subscribers
- **Common mishaps and how to avoid them**

Summary

N

To summarize, avoid if possible allowing your topic tree structure to grow organically, but rather, give advance thought to the size and shape of your topic tree, making use of Topic objects to exert administrative control over topics or groups of topics. For publishers, keep in mind that MQPut times are affected by the number of subscribers to a given topic, if and how those subscribers make use of wildcards, selectors, and other available options. For subscribers, be aware of the options available for filtering and selecting the publications to be delivered to them – but be sure to understand the costs associated with those options.

O

This presentation pointed out several areas where the MQI provides flexibility that goes beyond what the JMS specification allows: Things like Topic tree isolation and Unmanaged destinations are specific to the MQ message provider and its API, but with a little thought can be made available to JMS applications as well, without the need to use vendor-specific extensions in code.

T

This presentation also covered a number of “tricks of the trade”, how things like Administratively-defined subscriptions and various Topic and Subscription options can be used to best advantage, whether for better performance, for ease of administration, as well as dealing with “where’s my message” questions as well as common mishaps and how to avoid them. Hopefully this information will help you in getting the most out of using publish/subscribe with IBM MQ.

E

S

Questions & Answers

