WebSphere MQ Publish/Subscribe Part I

Agenda

- **Part I**
  - **Introduction**
  - **What is Pub/Sub?**
  - **Pub/Sub in WebSphere MQ**
  - **Pub/Sub and the WMQ API**
  - **Topic Tree Design**
- **Part II**
  - WMQ Pub/Sub Administration
  - WMQ Pub/Sub Operations
  - Tricks of the Trade
  - Topologies
  - Using WMQ V7 Pub/Sub and WebSphere Application Server
  - What's New

What is Publish/Subscribe?

This Slide Intentionally Left Blank

What is Publish/Subscribe?

Publish/Subscribe is a term used to define an application model in which the provider of some information is decoupled from the consumers of that information.

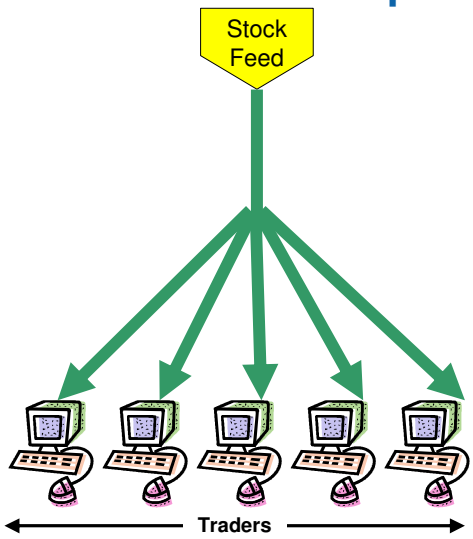
- Providers of information need have no knowledge of consumers
- Consumers of information need have no knowledge of providers
- Providers of information are called **publishers**
- Consumers of information are called **subscribers**
- New providers/consumers can be added without disruption

N O T E S

- Publish/subscribe systems have become very popular today as a way of distributing data between applications. Such systems are especially useful where data supplied by a publisher is constantly changing and a large number (or a variable number) of subscribers needs to be quickly updated with the latest data. Perhaps the best example of where this is useful is in the distribution of stock market data.
- In such systems, applications that are publishing the data do not need to know the identity or location of the subscriber applications which will receive the data. Similarly, the subscribing applications do not need to know the identity or location of the publishing application providing their information. In this sense the providers and consumers are said to be loosely-coupled.

Capitalware's MQ Technical Conference v2.0.1.3

The classic example



- A "feed" provides a continuous flow of information which is pushed to interested parties
- Traders consume this information and use it as a basis for the buying and selling stock

N

O

T

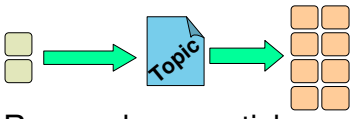
E

S

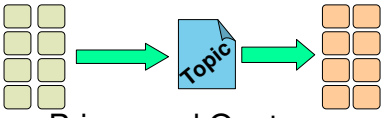
- Perhaps the most-commonly quoted example of a Publish/Subscribe system is one which provides stock-market information. Here a "feed" provides (publishes) a continuous flow of information containing the latest stock prices. The latest stock prices are required by traders who need this information in order to conduct trades. Traders register their interest in (subscribe to) particular stock prices and receive updates as prices change. Traders can be added/removed without disruption to the providers of the information who have no knowledge of who is receiving their information.
- The terms "push" and "pull" are also becoming increasingly popular when describing the flow of information between applications. If we concentrate on this example, traders receive new information from the stock-feed as soon as a stock price changes. In this sense the information can be thought of as being pushed directly to them. This pushing of information from provider to consumer is one of the major differentiators between publish/subscribe and more conventional systems. Our stock market example could have equally been designed in such a way that updated stock prices only flowed to the traders when they specifically requested, or pulled them from a central repository (server) of all stock prices. In such a system, the emphasis would instead be on the traders, to request a refresh of their stock prices on a continual basis.
- In fact WebSphere MQ supports both modes of operation.

Capitalware's MQ Technical Conference v2.0.1.3

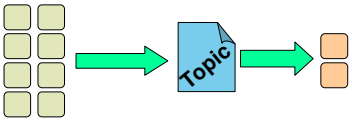
Loose-coupling with Publish/Subscribe



Few-to-many: Research, news tickers



Many-to-many: Prices and Quotes



Many-to-few: Orders

N O T E S

- In the WebSphere MQ Publish/Subscribe model the only thing which connects publishing and subscribing applications is the topic or subject which the publisher associates with his information. Publishers and subscribers need only agree on the topic to become connected to one another. Each different piece of information has its own topic associated with it. Subscribers nominate which types of information they want to receive by subscribing to specific topics.
- Publishers of information are unaware of subscribers to the extent that they may publish information even if there are no subscribing applications requiring it. Publishing and subscribing are completely dynamic processes. New subscribers and new publishers can be added to the system without disruption.
- With respect to a given topic, or piece of information, all possible combinations of publishers/subscribers are possible, that is:
  - information about each topic may be provided by a single or multiple publishing applications
  - the information may be received and processed by one or more subscribing applications
- The number of publishers and subscribers connected by a single topic depends upon the type of information which is flowing between them. As we will see later, WebSphere MQ supports both state and event based information, or topics.

Capitalware's MQ Technical Conference v2.0.1.3

Publications and subscriptions

- **Publishers provide information about specific topics by sending** publications **to the queue manager**
  - MQI publishers use the MQPUT verb
  - JMS publishers use the equivalent methods described by the JMS specification
- **Subscribers register** subscriptions **with the queue manager to indicate their interest in information relating to specific topics.**
  - MQI subscribers use the MQSUB verb
  - JMS subscribers use the equivalent methods described by the JMS specification
- **The queue manager forwards each publication it receives to all subscribers with a subscription which matches the associated topic**

N O T E S

- Just to recap, applications which provide information are called publishers. Applications which consume information are called subscribers.
- A publisher publishes its information by putting a message to a topic using the MQPUT verb (or its JMS equivalent).
- A subscriber specifies the topic it is interested in receiving information about by specifying it on the MQSUB verb (or its JMS equivalent). A subscriber may make multiple subscriptions to the queue manager.
- It is the job of the queue manager, or queue manager network if multiple queue managers have been connected together, to ensure that all subscribing applications with matching subscriptions to the topic being published on receive the publisher's message, known as a publication.
- There is a separate presentation about publish/subscribe in a multiple queue manager scenario.

Capitalware's MQ Technical Conference v2.0.1.3

Types of publications

- Events
  - Continuing succession of logically independent messages, for example:
    - trades
    - customer at a supermarket checkout
  - Subscribers receive as available
- State
  - Information that is being regularly updated or replaced, for example:
    - stock prices
    - furnace temperatures
  - Messaging provider retains a copy of the most recent publication
  - Subscribers may receive immediately or check at their own initiative

N

O

T

E

S

- When a publish/subscribe system is being designed it is important to decide whether the information being published on each topic is either state or event related.
- Examples of this type of information are:
  - a stock trade
  - a customer purchasing groceries at a supermarket checkout
- Event publications are usually independent from one another. They usually indicate that some further action or processing is needed. A subscriber missing an event may be disastrous and generally subscriptions to event publications all need to be in place before any events are published. There may be more than one publisher of event publications for a given topic.
- State publications usually contain information that is being updated at regular intervals. If a subscriber misses a state publication then generally it isn't a problem since an updated view of the state will about to be published again. The queue manager may also be instructed to retain the last copy of a state publication. This can be sent to new subscribers to that state topic rather than letting them wait for the information to be published again. Usually there is only a single publisher per state topic.
- Examples of this type of information are:
  - a stock price
  - the temperature of a furnace

Publish/Subscribe in WebSphere MQ

This Slide Intentionally Left Blank

History of MQ Publish/Subscribe

- **MQSeries V5.0**
  - MQSeries Publish/Subscribe introduced as SupportPac MA0C
- **WebSphere MQ V5.3 fixpack 8**
  - MQ Publish/Subscribe shipped as part of the product on platforms already supported
- **WebSphere MQ V6**
  - MQ Publish/Subscribe fully integrated into the product on all distributed platforms
- **WebSphere MQ V7**
  - Publish/Subscribe implementation completely rewritten
  - Previous Pub/Sub functionality retained
    - Renamed “Queued Publish/Subscribe Interface”
    - Needed for coexistence with MQ V6 applications
    - Enhanced to support RFH2 Publish/Subscribe messages for WMB V6 applications

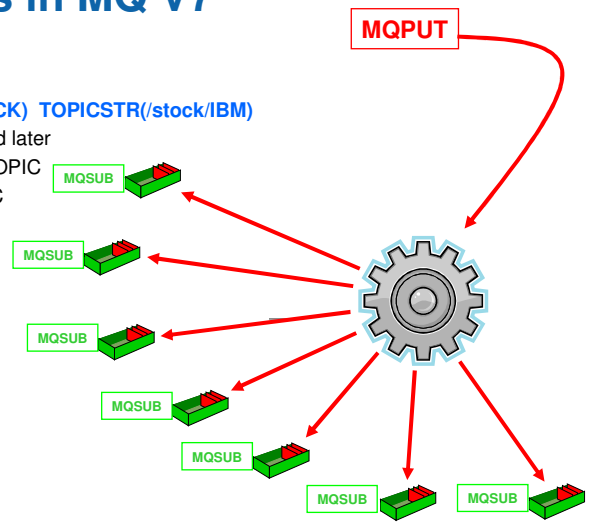
N O T E S

- Publish/Subscribe has been a feature of WebSphere MQ for a very long time. It was originally introduced with MQSeries V5.0 as a product extension (SupportPac MA0C) in 1998 – 15 years ago!
- It was supported for several years as a product extension, but as of WebSphere MQ V5.3 fixpack 8 (2004) the components were shipped along with the product, for those platforms where MA0C was already supported.
- As of WebSphere MQ V6 (2005) the Publish/Subscribe components were fully integrated into the product on all distributed platforms
- WebSphere MQ V7 (2008) changed the game completely as far as the Publish/Subscribe implementation was concerned. The Pub/Sub broker as a separate component was gone, with the functionality directly incorporated into the queue manager. This meant that pub/sub was now directly available on the zOS platform, where previously it had not been. Concepts such as stream queues were now gone, the API was changed from being message-based to a callable API, etc. But we could not abandon the many customers that were using the previous implementation. So a form of this Pub/Sub functionality was retained, in the form of what was called the “Queued Publish/Subscribe Interface”. This was needed for coexistence with MQ V6 (“RFH1”) applications. But it was also enhanced to also support WebSphere Message Broker (“RFH2”) Publish/Subscribe as well.
- This presentation is an in-depth look at all aspects of WebSphere MQ Publish/Subscribe as it exists today, focusing largely in three areas:
  - Topic tree administration control
  - No code change Publish/Subscribe
  - Application Monitoring

Capitalware's MQ Technical Conference v2.0.1.3

Pub/Sub Basics in MQ V7

- **TOPIC objects**
  - Basic definition:
    - **DEF TOPIC(STOCK) TOPICSTR(/stock/IBM)**
  - More attributes covered later
  - SYSTEM.DEFAULT.TOPIC
  - SYSTEM.BASE.TOPIC
- **MQOPEN (+ MQPUT)**
  - Key parameters:
    - TOPIC *object*
    - Topic *string*
  - (Later referred to as PUB)
- **MQSUB (+ MQGET)**
  - Key parameters:
    - TOPIC object
    - Topic string
    - Subscriber queue
  - (Later referred to as SUB)



N

O

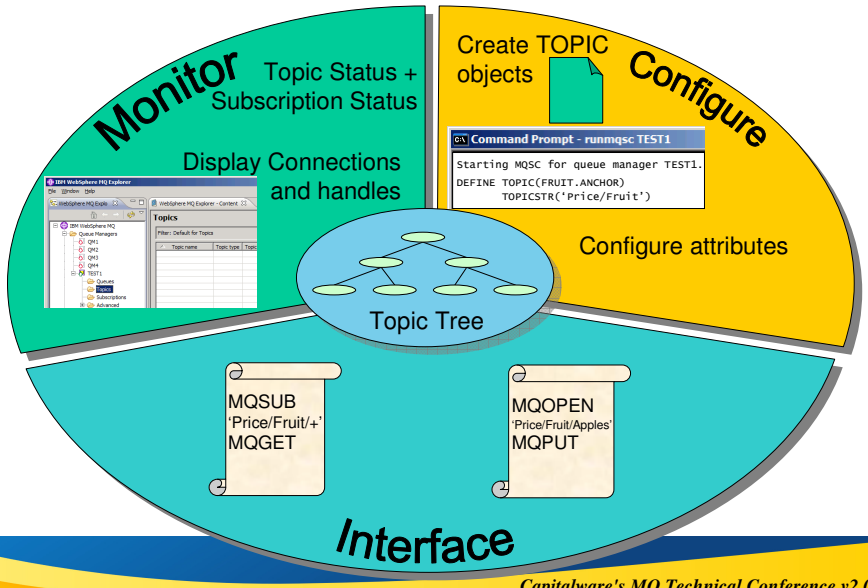
T

E

S

- WebSphere MQ v7 now includes pub/sub built into the API. This was done to replace the pub/sub function offered in WebSphere MQ v6 which was built on an MQRFH1 message interface. It replaced as well the pub/sub function offered in WebSphere Message Broker V6.x, which was built on an MQRFH2 message interface. Both WebSphere MQ and WebSphere Message Broker as of version 7 share a common Publish/Subscribe implementation.
- You'll notice significant differences between the V6 and V7 pub/sub implementations. For example, on the publishing side there is no queue; WebSphere MQ introduces a new object called a TOPIC object, and new API function to allow publishing directly to that object, as opposed to using a stream queue.
- In WebSphere MQ v6 for z/OS there was no pub/sub built into the product. WebSphere MQ v7 introduces pub/sub to the z/OS platform, which previously has no support for pub/sub.
- The new pub/sub available in v7 is easier to use and performs better than v6 pub/sub.
- In this presentation we will cover the following main areas:-
  - Topic tree administration control
  - No code change Publish/Subscribe
  - Application Monitoring

Publish/Subscribe in WebSphere MQ



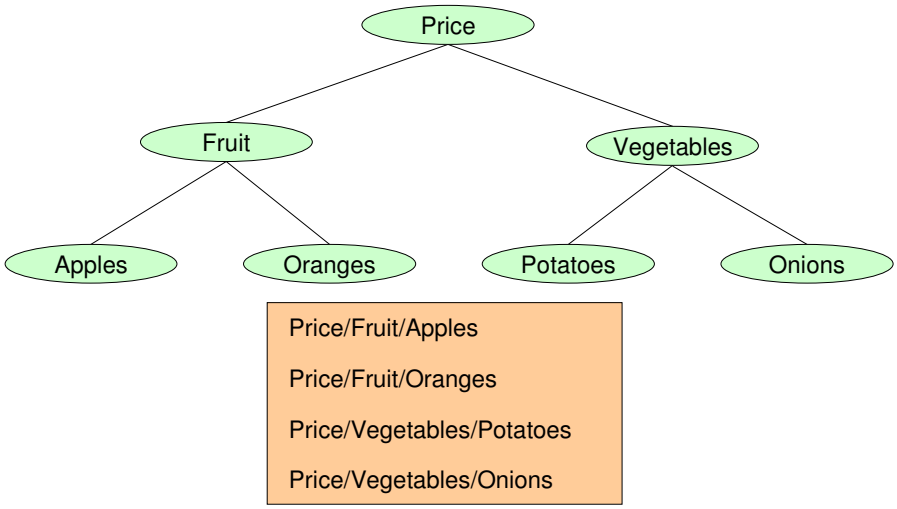
N O T E S

- The queue manager holds a view of all the topic strings you are using in a hierarchical construct known as the topic tree (the structure of which is described in more detail later). This topic tree is the central control point for all publish/subscribe. As a user you will interact with the topic tree in several different ways.
  - You can **configure** the behavior of the topic tree by defining topic objects (the purpose of which will be explained in detail shortly), changing attributes on them to suit your needs. Of course you only need to do this if you want to change the default behavior. You may not need any topic objects – we will look at this in more detail later.
  - You can **interface** with the topic tree programmatically as a subscriber using MQSUB and as a publisher using MQOPEN and MQPUT (or their equivalents in other APIs such as JMS or XMS).
  - You can **monitor** the use of your topic tree by such applications using the Topic status command, the Subscription status command and the commands to display connections and their handles.

Publish/Subscribe in the WMQ API

This Slide Intentionally Left Blank

Topic strings and topic tree



N O T E S

- Topic strings can be any characters you choose. You can, and should, add structure to your topic strings using the '/' character. This produces a topic tree with a hierarchical structure, as the example on this foil shows. Although this hierarchical topic tree was created by the use of the topic strings shown, we generally picture it as a tree such as this.
- There are some special characters, apart from the '/' character that you should avoid in your topic strings. These are '**#**', '+', '*' and '?'. We will look at these in more detail later when we discuss wildcards.
- An application that needs to publish a message about a specific topic can do so by opening that topic and putting a message to it. There is very little difference between an application that opens a queue then puts a message to it, and one that opens a topic to put a message to it, so the application code should look very familiar to you.
- This similarity in the programming model also allows us to turn a point-to-point application into a publish/subscribe application without making any code changes at all. We will look at this in the administration presentation.
- Since the object opened for output (MQOO_OUTPUT) is a topic and not a queue, the queue manager knows to publish the message to all interested parties rather than placing it on a specific queue.
- Let's look at the MQOD (Object Descriptor) in more detail. You will be familiar with fields such as the ObjectType and ObjectName that you use today when opening a queue. There are some differences to the way you use these fields if you are opening a topic rather than a queue.
- As you might guess, the ObjectType where you would place the value MQOT_Q when opening a queue, will instead have the value MQOT_TOPIC when you are opening a topic in order to publish a message.
- We said earlier that if you wish to use publish/subscribe with WebSphere MQ V7 you do not require any topic objects to be defined. You can in fact go right ahead and use a topic string directly in your application. You do this by placing your topic string in the ObjectString field.
- We have not mentioned any specific put-message options here. We will look at some specific publishing ones later, but almost all those that you know for point-to-point application programming are still applicable.

Publishing Application

- **MQOPEN a topic**
- **MQOD describes a topic to publish to**
  - ObjectType
    - MQOT_Q for point-to-point
    - MQOT_TOPIC for publish
  - ObjectString/ObjectName
- **MQPUT a message**

```
OpnOpts = MQOO_OUTPUT
        | MQOO_FAIL_IF_QUIESCING;
MQOPEN( hConn,
        &ObjDesc,
        OpnOpts,
        &hObj,
        &CompCode,
        &Reason);
MQPUT ( hConn,
        hObj,
        &MsgDesc,
        &pmo,
        strlen(pBuffer),
        pBuffer,
        &CompCode,
        &Reason);
```

```
MQOD   ObjDesc = {MQOD_DEFAULT};

ObjDesc.ObjectType              = MQOT_TOPIC;
ObjDesc.Version                 = MQOD_VERSION_4;
ObjDesc.ObjectString.VSPtr      = "Price/Fruit/Apples";
ObjDesc.ObjectString.VSLength   = MQVS_NULL_TERMINATED;
```


N O T E S

- An application that needs to publish a message about a specific topic can do so by opening that topic and putting a message to it. There is very little difference between an application that opens a queue then puts a message to it, and one that opens a topic to put a message to it, so the application code should look very familiar to you.
- This similarity in the programming model also allows us to turn a point-to-point application into a publish/subscribe application without making any code changes at all. We will look at this in the administration presentation.
- Since the object opened for output (MQOO_OUTPUT) is a topic and not a queue, the queue manager knows to publish the message to all interested parties rather than placing it on a specific queue.
- Let's look at the MQOD (Object Descriptor) in more detail. You will be familiar with fields such as the ObjectType and ObjectName that you use today when opening a queue. There are some differences to the way you use these fields if you are opening a topic rather than a queue.
- As you might guess, the ObjectType where you would place the value MQOT_Q when opening a queue, will instead have the value MQOT_TOPIC when you are opening a topic in order to publish a message.
- We said earlier that if you wish to use publish/subscribe with WebSphere MQ V7 you do not require any topic objects to be defined. You can in fact go right ahead and use a topic string directly in your application. You do this by placing your topic string in the ObjectString field.
- We have not mentioned any specific put-message options here. We will look at some specific publishing ones later, but almost all those that you know for point-to-point application programming are still applicable.

Capitalware's MQ Technical Conference v2.0.1.3

Topic Objects

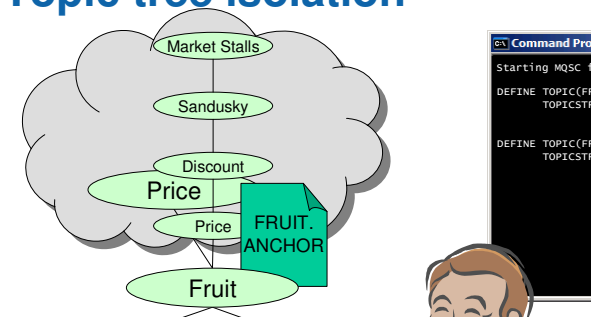
- **Not necessary for Publish/Subscribe**
- **Provide an administrative control point for your topic tree**
  - Configuration attributes
  - Security profiles
  - Topic tree isolation

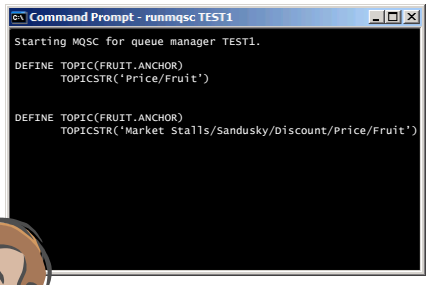
N O T E S

- Topic objects are a new construct in WebSphere MQ V7. They can be used to control the behavior of your topic tree.
- You do not need to define any topic objects in order to use Publish/Subscribe with WebSphere MQ V7, however you may want to define some if you need to configure the topic tree to use non-default attributes; if you want to apply different security profiles to parts of your topic tree; or if you want to isolate your applications from administrative changes to the topic tree – rather like you do when you use remote queue and alias queue definitions.
- We will look at how applications might use topic objects instead of (or as well as) topic strings and also how you define these objects and what some of their attributes mean.

Capitalware's MQ Technical Conference v2.0.1.3

Topic tree isolation





```
ObjDesc.ObjectString.VSPtr
ObjDesc.ObjectString.VSLength  ?                    .D;
```


N O T E S

- 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 'Locations/Sandusky/Discount/Price/Fruit...' so you wish to avoid hard-coding into your application a topic string that may change.
- 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.
- This isolates you application development from changes in the design of the full topic tree structure.
- 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.

Subscribing Application

- **MQSUB verb**
- **Subscription Descriptor (MQSD) describes the topic**
  - MQSD.ObjectString
  - MQSD.ObjectName
- **Consume publications from the returned hObj**
  - when MQSO_MANAGED used

```
MQSUB ( hConn,
        &SubDesc,
        &hObj,
        &hSub,
        &CompCode,
        &Reason);

MQGET ( hConn,
        hObj,
        &MsgDesc,
        &gmo,
        strlen(pBuffer),
        pBuffer,
        &DataLength,
        &CompCode,
        &Reason);
```

```
MQSD   SubDesc = {MQSD_DEFAULT};
SubDesc.ObjectString.VSPtr    = "Price/Fruit/Apples";
SubDesc.ObjectString.VSLength = MQVS_NULL_TERMINATED;
SubDesc.Options               = MQSO_CREATE
                              | MQSO_MANAGED
                              | MQSO_FAIL_IF_QUIESCING;
```


N O T E S

- An application that wants to register an interest in information about a certain topic needs to 'subscribe' to that topic. This can be done using the MQ API verb MQSUB. MQSUB can be thought of rather like MQOPEN. It details the resource you wish to use, and it is the point where security checks are done.
- The main structure that you need to be familiar with when using MQSUB is the MQSD (subscription descriptor). This structure is where you define the topic you are interested in; the options to used when making a subscription; and any other interesting changes to the way the subscription is made.
- When specifying the topic string you wish to subscribe for, you have the same mechanisms available to you that we already discussed on the foil about a publishing application. You can provide the whole topic string, or an anchoring topic object which defines a certain point in the topic tree and then the remaining part of the topic string to be appended to that which the topic object represents.
- Once you have successfully done an MQSUB, you can start to consume publication messages that are now being sent to you.

Subscription operation options

- **Operation – choose at least one of**
  - MQSO_CREATE
  - MQSO_RESUME
  - MQSO_ALTER
- **Combining them**
  - MQSO_CREATE + MQSO_RESUME
    - Avoids MQRC_SUB_ALREADY_EXISTS
  - MQSO_CREATE + MQSO_ALTER

N O T E S

- We've already seen examples of MQSO_CREATE. There are two other options, MQSO_RESUME and MQSO_ALTER. You must choose at least one of these.
- MQSO_RESUME can be used to get hold of a previously made subscription – a durable one – to obtain both the handle to the subscription for later closure, and the handle to the managed destination if that mode of operation was chosen.
- MQSO_ALTER also gives you back the handle to a previously made subscription, but at the same time alters various properties on the MQSD to new values that you provide.
- These uses are probably fairly obvious, but additionally, you can combine these options. If you use MQSO_CREATE + MQSO_RESUME, this will create a subscription if it doesn't exist and resume it if it does, thus avoiding the need to code your application to check for MQRC_SUB_ALREADY_EXISTS if it's not the first time your application has run. You can similarly combine MQSO_CREATE + MQSO_ALTER.

Capitalware's MQ Technical Conference v2.0.1.3

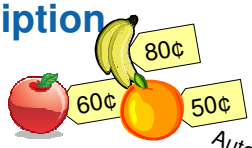
Other attributes of a subscription

Durable

Whether a subscription lives beyond the application connection

Sub Destination

Where messages are stored – provided hObj by app or returned from queue manager



Wildcard Scheme

MQ V6 style - * and ?

MB V6 style - # and +

Expiry

Automatically remove subscriptions after certain period of time

```
SubDesc.ObjectString.VSPtr     = "Price/Fruit/+";
SubDesc.ObjectString.VSLength  = MQVS_NULL_TERMINATED;
SubDesc.SubName.VSPtr          = "Selling Fruit";
SubDesc.SubName.VSLength       = MQVS_NULL_TERMINATED;
SubDesc.SubExpiry              = 3000;
SubDesc.Options                = MQSO_CREATE
                               | MQSO_DURABLE
                               | MQSO_MANAGED
                               | MQSO_WILDCARD_TOPIC
                               | MQSO_FAIL_IF_QUIESCING;
```


This Slide Intentionally Left Blank

Durability of Subscriptions

- **Non-durable by default**
  - Removed when application disconnects
- **Can be made durably**
  - Still exists while application is off-line
- **Durable vs 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

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

Durability of Subscriptions - Notes

N O T E S

- 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.
- You can also make a subscription durable. This means that the subscription continues to exist even while the application is disconnected form 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.
- Whether you use a durable subscription or a non-durable subscription depends on the requirements of your application.

API for durable subscriptions

- MQSO_DURABLE
- Provide a subscription name (MQSD.SubName)
- Remove with MQCO_REMOVE_SUB on MQCLOSE

```
MQSUB ( hConn,
        &SubDesc,
        &hObj,
        &hSub,
        &CompCode,
        &Reason);

if ( hSub != MQHO_UNUSABLE_HOBJ)
{
  MQCLOSE( hConn,
           &hSub,
           MQCO_REMOVE_SUB,
           &CompCode,
           &Reason);
}
```

```
SubDesc.SubName.VSPtr      = "Selling Apples";
SubDesc.SubName.VSLength   = MQVS_NULL_TERMINATED;
SubDesc.Options            = MQSO_CREATE
                           | MQSO_DURABLE
                           | MQSO_MANAGED
                           | MQSO_FAIL_IF_QUIESCING;
```


API for durable subscriptions - Notes

N O T E S

- To make a durable subscription you must use the MQSO_DURABLE option on MQSUB.
- In order for a subscription to be durable you must also provide it with a subscription name. This is used if you need to refer to the subscription later – for example when you use the MQSO_RESUME option on MQSUB to reacquire a handle to the subscription.
- If you want to remove a durable subscription you do so by explicitly closing the handle to the subscription using the MQCO_REMOVE_SUB option on MQCLOSE.

Subscription Destinations

- MQSO_MANAGED
- Queue manager returns hObj

- Omit MQSO_MANAGED
- Provide queue manager with hObj of queue desired
- Can have many subscriptions sharing the same queue

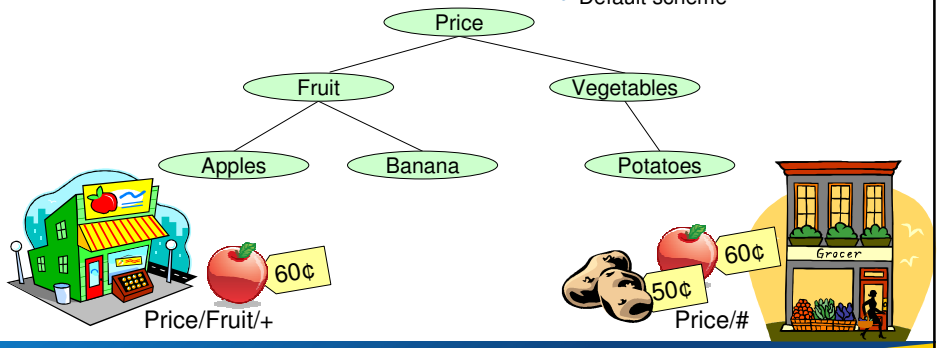
Subscription Destinations - Notes

N O T E S

- So far we have seen examples of MQSUB using the option MQSO_MANAGED. This is the option to use when the application wishes the queue manager to look after the storage of publication messages. On return from the MQSUB call your application is given two handles, an hSub and an hObj. hSub is the handle to the subscription which, as we have just seen, can be used to MQCLOSE the subscription when you are finished with it. The hObj is the handle which you can consume publications from, i.e. using MQGET.
- You can also request that the queue manager place the messages on a specific queue instead. To do this, you provide the MQSUB call with an hObj on input, so you must first MQOPEN the queue you wish to use, provide the resultant handle to the MQSUB and then you can consume from that queue.

Wildcards

- **Character based**
  - As used by WebSphere MQ V6
  - '*' – matches many characters
  - '?' – matches single character
- **MQSO_WILDCARD_CHAR**
- **Topic element based**
  - As used by WebSphere MB
  - '#' – matches many elements
  - '+' – matches single element
- **MQSO_WILDCARD_TOPIC**
  - Default scheme



N

O

T

E

S

- So far we have only seen examples of using MQSUB to subscribe to an explicit topic string.
- If you design your topic structure well, your applications can make use of wildcarded subscriptions and thus not need any redesign work as new topic strings are created that did not previously exist in the topic tree.

- There are two different wildcard schemes supported in WebSphere MQ V7.
- A character based wildcard scheme which copies the way the queued publish/subscribe interface that existed in WebSphere MQ V6 operated. This uses '*' to replace many characters and '?' to replace one character.
- A topic element based wildcard scheme which copies the way WebSphere Message Broker operated. This uses '#' to replace many elements (an element is considered to be the text between '/' characters) and '+' to replace a single element.
- You can choose to use either scheme in your subscriptions using the options MQSO_WILDCARD_CHAR or MQSO_WILDCARD_TOPIC. The default is MQSO_WILDCARD_TOPIC.

Subscription expiration

- **Removal of subscriptions**
  - Connection loss with a non-durable
  - MQCLOSE with MQCO_REMOVE_SUB
  - Administrative DELETE SUB command
  - Expiry
- **Set expiry at subscription create time**

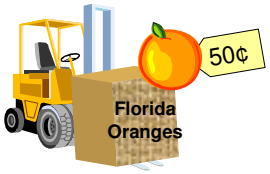
```
SubDesc.ObjectString.VSPtr    = "Price/Fruit/Apples";
SubDesc.ObjectString.VSLength = MQVS_NULL_TERMINATED;
SubDesc.SubExpiry             = 3000;
SubDesc.Options               = MQSO_CREATE
                              | MQSO_FAIL_IF_QUIESCING;
```

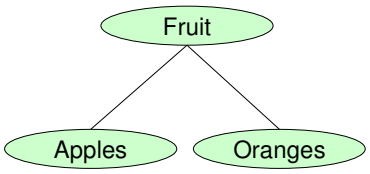

Subscription expiration - Notes

N O T E S

- We have already seen two different ways in which subscriptions can be removed from the queue manager:-
  - Connection loss for non-durable subscriptions
  - Administrative DELETE SUB command (see later)
  - MQCLOSE with MQCO_REMOVE_SUB
- There is one other way that a subscription can be removed and this is expiry.
- When a subscription is made you can set the desired expiry of the subscription and after that interval has passed the subscription will be removed from the queue manager.

Publication Selection





Price/Fruit/Oranges

Origin='Florida'

```
SubDesc.ObjectString.VSPtr        = "Price/Fruit/Orange";
SubDesc.ObjectString.VSLength     = MQVS_NULL_TERMINATED;
SubDesc.SelectionString.VSPtr     = "Origin='Florida'";
SubDesc.SelectionString.VSLength  = MQVS_NULL_TERMINATED;
SubDesc.Options                   = MQSO_CREATE
                                  | MQSO_FAIL_IF_QUIESCING;
```


N O T E S

- So far we have only looked at application's subscribing to all the messages on a particular topic string. Your application may wish to subscribe more specifically than that. Let's continue our example of fruit prices with a new seller in the market-place specialising in East Coast Produce. This seller only wants to subscribe to the prices of Florida oranges, but no other oranges.
- The application publishing the messages has added several pieces of meta data about the messages as message properties, such as the Origin of this produce and the name of the Supplier. Any of these properties can be selected upon at subscribe time.
- This means that only those publications matching both the topic string and the selection criteria are ever delivered to the subscriber.
- How to set and inquire upon message properties is outside the scope of this presentation.

Topics verses Selectors – When to use which?

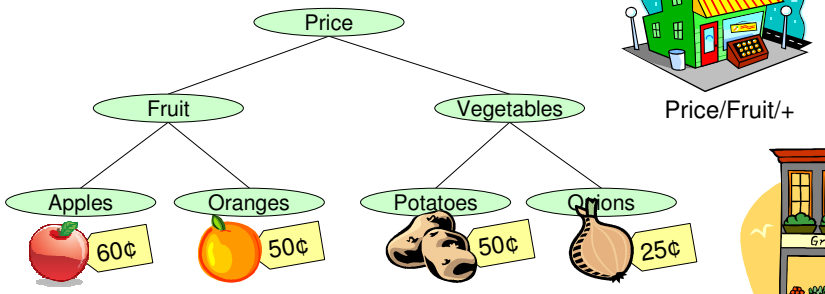
- **Topics and selectors are not mutually exclusive**
  - It's possible to combine a topic hierarchy with message properties and selection strings to best fit the usage pattern
- **In general, topic strings give better performance than message properties and selectors**
- **However, having *excessive numbers* of topic strings can impact performance**
- **Can address this by using selectors where *normally* a subscriber may not require the further selection criteria but occasionally they may**
  - e.*g. Don't encode the price into the topic structure for the rare occasion where a subscriber wants to know when it reaches $100*
    - *Instead, add it as a message property and allow a selector to be used*
- ***S*electors are not flowed around a multiple queue manager publish/subscribe topology**
  - Selector parsing is only performed on the queue managers where the subscriptions exist
- **Goal should be to strike a balance between topic tree complexity and use of selectors**

N O T E S

- Whether to use Topics or Selectors is not an either/or choice.
- It is possible to combine a topic hierarchy with message properties and selection strings to best fit the usage pattern
- Considerations to take into account:
  - Generally speaking, topic strings give better performance than message properties and selectors
  - However, having excessive numbers of topic strings can impact performance
- So the goal should be to strike a balance between topic tree complexity and use of selectors.
  - One approach is to use message properties and selectors for those cases where normally a subscriber may not require the further selection criteria, but on occasion they may. For example, don't encode the price into the topic structure for the rare occasion where a subscriber wants to know when it reaches $100. In such a case you would be better off by having the publisher include this value as a message property, and the subscribers can choose to use a selector for those limited cases where filtering on this value would benefit them.
- With pub/sub hierarchies or clusters, it is useful to note that selectors are not flowed around the entire multiple queue manager publish/subscribe topology; selector parsing is only performed on the queue managers where the subscriptions exist.

Capitalware's MQ Technical Conference v2.0.1.3

Retained Publications



Price/Fruit/+

Price/#

```
pmo.Options |= MQPMO_RETAIN
```

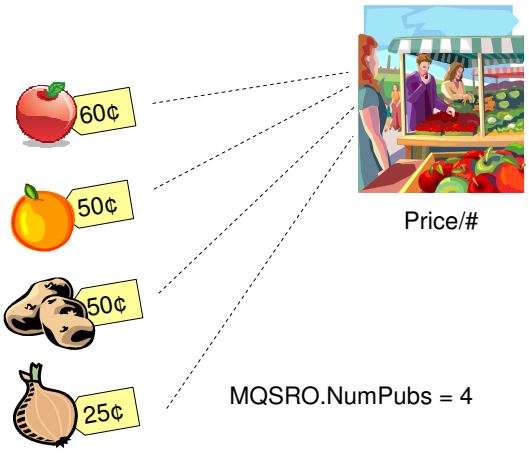
Price/Vegetables/Onions

N O T E S

- Earlier we described the concept of event information and state information. In order to provide state information, that is a publication which is held as the last known state of something, you can tell the queue manager to retain that publication. This is done using the MQPMO_RETAIN option when putting the message.
- A subscriber that comes in later would have to wait until the next time an event occurred if we didn't use this mechanism. With the use of retained publications our new subscriber is immediately given the last known information on this topic.
- For some applications it may be important to only be sent new information. Old information such as a retained publication may be useless – so you can turn off this behavior using the MQSO_NEW_PUBLICATIONS_ONLY option on your MQSUB call.

Request a Publication

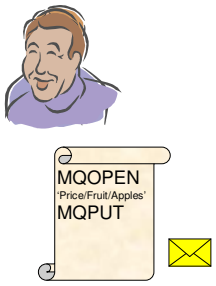
```
/* Topic: 'Price/#'                */
MQSUB ( hConn,
        &SubDesc,
        &hObj,
        &hSub,
        &CompCode,
        &Reason);

MQSUBRQ( hConn,
         hSub,
         MQSR_ACTION_PUBLICATION,
        &SubRqOpts,
        &CompCode,
        &Reason);

MQGET ( ...
```



N O T E S

- If you are only connected to the system from time to time, or if regular updates are not what you require, you can choose to only get publications when you request them.
- You make a subscription in the normal way, additionally using the MQSO_PUBLICATIONS_ONLY_REQUEST option. Then whenever you want the latest state, you make a request for it using the MQSUBRQ verb. This will just send you the latest publication on the topic you have subscribed on.
- If you have a wildcarded subscription you will get the latest publication on each topic string that matches your subscription. The field MQSRO.NumPubs will detail how many publications have been sent to you.

Question: What if there are no Subscribers?



- Standard operation for publication is to give no indication where there were no subscribers
- This is reasonable, given the pub/sub model where publishers and subscribers are disconnected and thus there is not suppose to be any awareness between publishers and subscribers
- But this behavior may not be desirable in all cases

```
pmo.Options |= MQPMO_WARN_IF_NO_SUBS_MATCHED
```

MQRC_NO_SUBS_MATCHED

N O T E S

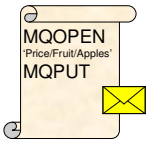
- With publish/subscribe, standard operation of Publish is to give no indication to the publishing application if there were no subscribers interested in the topic on which it is publishing. In such a case, the message will simply be discarded, without notification to the publisher that this has occurred. This is not unreasonable, given that with the pub/sub model there is not suppose to be any awareness between publishers and subscribers
- But with enterprise messaging, there may be cases where this behavior is undesirable. You may have publications where the expectation is that there will always be at least one subscriber, and the publisher will want to take some action to preserve the data in such a situation. But how will it know?
- WebSphere MQ 7.0.1 delivers a new feature to address this. A new PMO option **[MQPMO_WARN_IF_NO_SUBS_MATCHED]**, can be set to indicate that the publisher wants to be notified if there are no subscribers registered at the time the MQPut is issued.
- If this option is specified and the message is not delivered to any subscription, a CompCode of [MQCC_WARNING] is returned with the Reason Code [MQRC_NO_SUBS_MATCHED]
- Note that this option is only valid when putting to a topic, so it is not available is the Queued Pub/Sub interface is being used by the publisher. It also is not returned if the publisher is using the JMS API, as the JMS specification does not support this.
- For customers using WebSphere Message Broker (now IBM Integration Bus), the Publication node was extended in V7 to exploit this feature, enabling the message flow to optionally drive exception logic in a flow if a message is published but there are no matching subscribers.

Question: What if a failure occurs on message delivery?

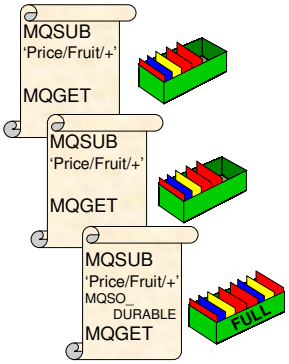
MSGDLV	behavior
ALL	**If one failure – no-one gets it**
ALLDUR	**Any failure to a durable subscriber – no-one gets it**
ALLAVAIL	**Failures don't affect the delivery to subscribers who can accept it**

TOPIC attribute defaults

PMSGDLV (ALLDUR)
NPMSGDLV (ALLAVAIL)



MQRC_PUBLICATION_FAILURE



N O T E S

- If a publication satisfies a number of subscribers and the queue manager is unable to deliver the publication to one of the subscribers, say because its destination queue is full (and additionally the message cannot be placed on the Dead Letter Queue (DLQ) either because it is not being used or because it is out of commission for some reason) – what happens? Well, there are essentially three choices and these are configured on the topic in question using the PMSGDLV and NPMSGDLV attributes – so different behavior can be configured for persistent and non-persistent messages on the topic.
- You can configure your system so that all subscribers must be delivered the publication or none of them get it. This is the value ALL. If one of the subscribers cannot accept it as in our diagram, the MQPUT fails with MQRC_PUBLICATION_FAILURE.
- You can configure your system so that all durable subscribers must be delivered the publication or none of them get it. This is the value ALLDUR. If one of the durable subscribers cannot accept it as in our diagram, the MQPUT fails, again with MQRC_PUBLICATION_FAILURE.
- You can configure your system so that no failure of one specific subscriber has any affect on the delivery of the publication to other subscribers. This is the value ALLAVAIL.

Questions?

This Slide Intentionally Left Blank

Topic Tree Design Considerations

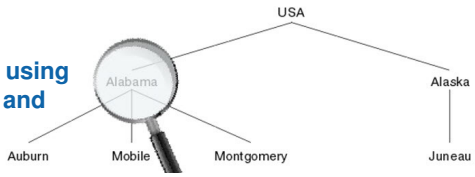
Why worry about the size and shape of the topic tree?

- **Simpler administration!**
  - Fewer topic objects and topic nodes mean less to worry about
- **Better performance!**
  - A well-thought-out structure can have a large impact on performance
- **MQ can efficiently scale to tens or hundreds of thousands of topic nodes**
  - But certain topic designs can cause MQ pain, which impacts users

Capitalware's MQ Technical Conference v2.0.1.3

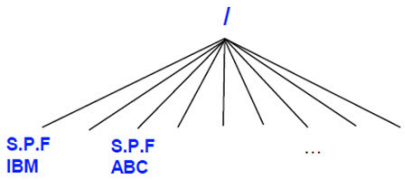
Designing Your Topic Tree Structure

- **No "standard" way of designing the shape of a topic tree**
  - Depends on the specific data
- **General recommendation is to describe the high level / broad information, then break this down into finer detail**
  - Start by looking at the "big picture"
  - Topic Tree structure likely to evolve over time
- **Can combine topics with Selectors or Content Filters**
  - Both can have an impact on performance
- **Topic objects can be used for isolation of a subset of the tree**
- **If using distributed pub/sub avoid clustering the root node – consider using high level Topics such as "/global" and "/local"**



Considerations for Designing Your Topic Tree

If possible, try to avoid



- **A shallow topic tree**
  - StockPriceForIBM
  - StockPriceForABC
  - StockPriceForXYZ
- **An unnecessarily deep topic tree**
  - This/Is/The/Stock/Price/For/IBM
  - Where one topic node in the tree only points (and will only point) to a single topic node within the topic tree
    - Don't capture structure within the tree that you will never be interested in and has no purpose
    - Remember that Topic Objects can be used to "expand" the tree structure
- **Mandating multi-level wildcards (#) in the middle of a topic string unnecessarily**
  - This can have a performance implication
- **Different types of information on the same topic string**
  - Mandating the use of Selectors or Content Filters, or client-side filtering
    - StockPriceForIBMAndXYZ
- ***Best way to appreciate this is to understand what's happening internally…***

What happens when a new Topic is added to the tree?

- **When an application opens a topic string…**
  - The topic string is looked up in a hash table
    - The more topic strings the bigger the hash table
- **If found, *job done***
- **If not found**
  - Topic string is stripped down to its '/' delimited parts
  - Every new topic node that is referenced in the topic string is dynamically created
  - Every new new node is linked to its parent
  - For every topic node created, that branch of the topic tree is walked backwards to discover
    - The node's configuration
    - The set of wildcard subscriptions that may match it
- **The more topic nodes the more memory consumed by the queue manager**
  - *Very broad topic trees put stress on certain parent topic nodes*
  - *Very deep topic trees add unnecessary topic nodes and levels to the tree that must be walked*
    - *Very many topics can strain the memory resources of the system*
    - *Subscriptions with # wildcards at the start or middle of the topic string put them higher up the tree - means they will need to be checked for almost all topic nodes being created*

N O T E S

- First, it makes your administration simpler! The less topic objects and topic nodes there are, the less there are to worry about.
- Second, MQ can efficiently scale to tens or hundreds of thousands of topic nodes but certain topic designs can cause MQ pain, which impacts the user.
  - (Future improvements take that scaling even further)
- *It's best to understand what's happening internally…*
- When an application opens a topic string…
  - The topic string is looked up in a hash table
    - The more topic strings the bigger the hash table
  - If found, *job done*
  - If the topic string is not found the topic string is stripped down to its '/' delimited parts and every new topic node that is referenced in the topic string is dynamically created. Each new node is linked to its parent.
  - For every topic node created, that branch of the topic tree is walked backwards to discover
    - The node's configuration
    - The set of wildcard subscriptions that may match it
  - The more topic nodes the more memory consumed by the queue manager
- *For this reason:*
  - *Very broad topic trees put stress on certain parent topic nodes*
  - *Very deep topic trees add unnecessary topic nodes and levels of the tree that need to be walked.*
  - *Very many topics can put a strain on memory resources of the system*
  - *Subscriptions with # wildcards at the start or middle of the topic string put them higher up the tree and can therefore mean they will need to be checked for almost all topic nodes being created*

Capitalware's MQ Technical Conference v2.0.1.3

What happens when an application publishes to a topic string?

- **When an application publishes a message on a topic string…**
  - For every subscription associated with the topic node…
    - A decision is made to deliver a copy of the message or not
    - If the subscription specified a selector, the message properties are parsed to check for a match
- **Consider this example:**
  - Consider a single topic with a thousand subscriptions
    - Each publication will result in a thousand checks
  - Consider a hundred topics, each with ten subscriptions, all with selectors
    - Each publication will result in ten checks
  - Both would result in the same number of publications
    - But the latter with a lot less work

Other processing affected by the size and shape of the tree

- **Various housekeeping tasks run in the background**
  - These are also affected by the size and shape of the topic tree
- **Periodically the topic nodes are scanned to see if they are still "in use"**
  - A topic tree node is considered to NOT be in use if
    - It is a leaf topic that is not represented by a Topic Object
    - It has no publishers or subscriptions currently directly attached
      - Prior to 7.0.1.8 wildcard subscriptions also were counted
    - It has not been used for a period of time (minimum 30 minutes by default)
- **Nodes that are considered to be not "in use" are deleted and unlinked from their parent**
- **This is necessary to keep a cap on memory use and a cluttered topic tree when viewing**

Other processing affected by the size and shape of the tree (continued)

- **What are the implications of the shape of the topic tree?**
  - The bigger the topic hierarchy, the more additional work incurred
- **How can you minimize this additional work?**
  - Try not to have topics that are never or very rarely reused
    - e.g. Don't encode a unique job ID into a topic string
    - Instead, add it as a message property and use a selector
  - Don't encode too many additional '/' delimited layers in the tree where unnecessary
    - Remember that you can use Topic Tree Isolation to add levels to the tree later, if needed
  - **If a topic is not re-published to and has no direct subscribers (only through wildcards) it may get deleted before it is re-used, meaning the creation overhead is incurred for every publish**
  - Use the TREELIFE queue manager property to control this

N O T E S

- When an application publishes a message on a topic string
  - For every subscription associated with the topic node:
    - A decision is made to deliver a copy of the message or not, if the subscription has a selector the message properties are parsed to assess its suitability
- For this reason:
  - By example, if there is a single topic with a thousand subscriptions, all with selectors, each publication will result in a thousand checks.
  - However, if there are one hundred topics, each with ten subscriptions, all with selectors, each publication will result in ten checks.
  - Both would result in the same number of publications, the latter with a lot less work
- And finally, in the background…
  - Periodically the topic nodes are scanned to discover if they are still *needed*
    - An unneeded topic node is a leaf topic with no publishers or subscriptions currently directly attached
- And the topic node has not been used for a period of time (minimum 30 minutes by default)
  - In the case of unneeded topics they are deleted and unlinked from their parent
  - This scanning is necessary to keep a cap on memory use and a cluttered topic tree when viewing.
- For this reason:
  - The bigger the topic hierarchy, the more additional work incurred.
  - Try not to have topics that are never or very rarely reused.
    - E.g. Don't encode a unique job ID into a topic string, add it as a message property.
  - Don't encode too many additional '/' delimited layers in the tree where unnecessary.
  - If a topic is not re-published too and has no direct subscribers (only through wildcards) it may get deleted before it is re-used, meaning the creation overhead is incurred for every publish.

Capitalware's MQ Technical Conference v2.0.1.3

Summary

Summary - WebSphere MQ Publish/Subscribe

- Application Programming
- **Pub/Sub**
  - Event and State models
- **Publishing**
  - MQPUT to topic
    - Object Descriptor (MQOD) extended for use with topics
    - New MQPMO option
- **Subscribing**
  - MQSUB
    - Subscriber controls delivery
      - Wildcards, selectors, etc
    - Subscription Descriptor (MQSD) used to control behavior
  - Consume publications
    - Use MQGET or Asynchronous Consume
- Administration
- **Topic tree administration control**
  - TOPIC objects
    - Topic Tree Isolation
    - Retention, Message Delivery, etc
- **Topic tree Design**
  - Not too wide, not too deep
  - Considerations