WebSphere MQ Publish/Subscribe
[z/OS & Distributed]

Morag Hughson (hughson@uk.ibm.com)

Session # 9419

WebSphere MQ Publish/Subscribe

N O T E S

- In WebSphere MQ V7 Publish/Subscribe becomes an in-built part of the MQ API (Application Programming Interface) and the administration model of WebSphere MQ.
- WebSphere MQ V7 extends the MQ API (Application Programming Interface) to allow application programmers to use the publish/subscribe application model with ease. New verbs and changes to existing verbs are introduced in this presentation.
- WebSphere MQ V7 also extend the administrative interfaces (MQSC and PCF) to allow administrators to manage Publish/Subscribe applications.
- We will cover the following main areas:-
  - Topic tree administration control
  - No code change Publish/Subscribe
  - Application Monitoring
- There are a number of fully working samples provided for your reference throughout this presentation. These are very similar to the samples shipped with WebSphere MQ V7 but have been cropped a little to squeeze them onto 2 pages in each case.

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.

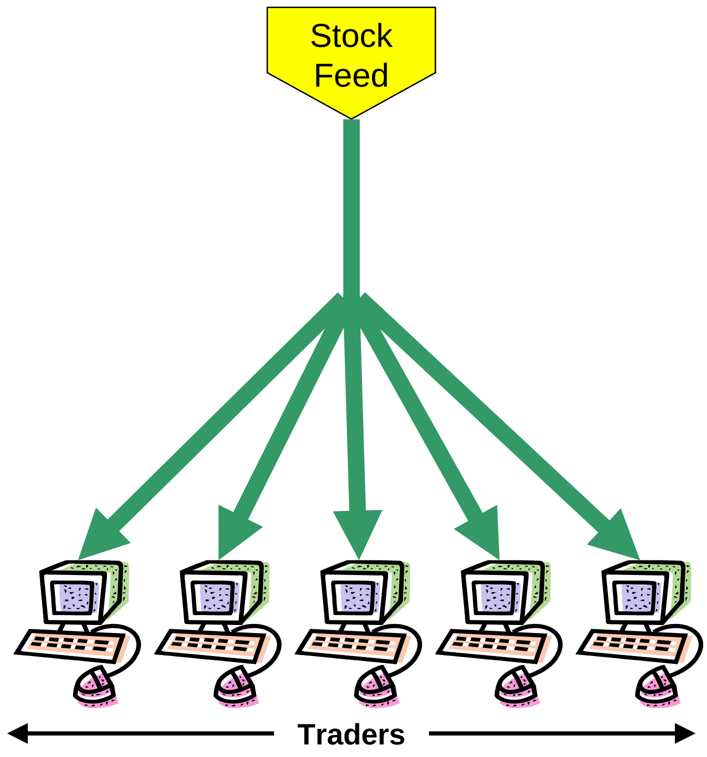
- providers of information need have no knowledge of consumers
- consumers of information need have no knowledge of providers
- Providers of information are called **publishers**
- Consumers of information are called **subscribers**
- new providers/consumers can be added without disruption

What is Publish/Subscribe? - Notes

N O T E S

- Publish/subscribe systems have become very popular in recent years as a way of distributing data messages from publishing computers to subscribing computers. Such systems are especially useful where data supplied by a publisher is constantly changing and a large 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, publisher applications of data messages do not need to know the identity or location of the subscriber applications which will receive the messages. 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.

The classic example



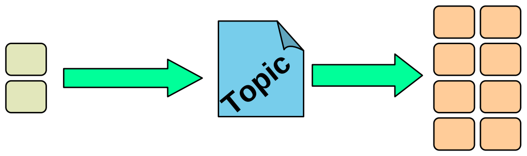
- A "feed" provides a continuous flow of information which is pushed to interested parties
- Traders consume this information and use it as a basis for the buying and selling stock

The classic example - Notes

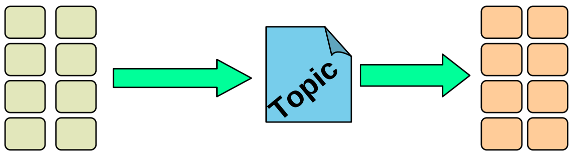
N O T E S

- 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.

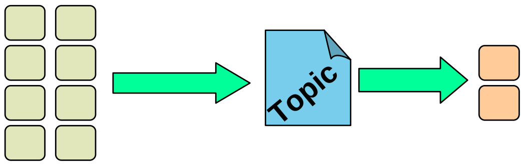
Loose-coupling with Publish/Subscribe



Few-to-many: Research, news tickers



Many-to-many: Prices and Quotes



Many-to-few: Orders

Loose-coupling with Publish/Subscribe - Notes

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.

Publications and subscriptions

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

Publications and subscriptions - Notes

N O T E S

- Just to recap, applications which provide information are called publishers. Applications which consume information are called subscribers.
- A subscriber specifies the topic it is interested in receiving information about by specifying it on the MQSUB verb. A subscriber may make multiple subscriptions to the queue manager.
- A publisher publishes its information by putting a message to a topic.
- 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.

Types of publications

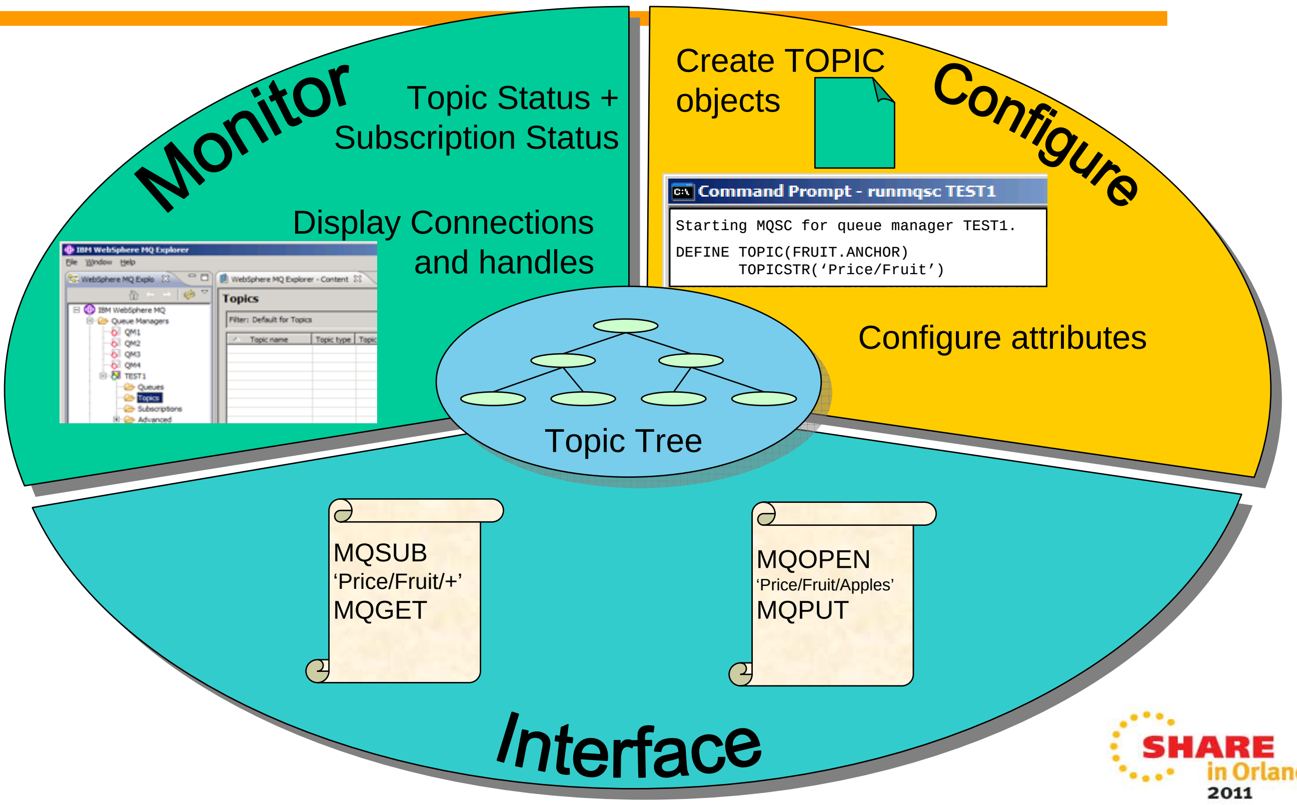
- Events
  - Continuing succession of logically independent messages, for example:
    - trades
    - customer buying an airline ticket
  - Subscribers receive as available
- State
  - Information that is being regularly updated or replaced, for example:
    - stock prices
    - furnace temperatures
  - Queue managers can retain copy of last publication
  - Subscribers may receive immediately or check at their own initiative

Types of publications - Notes

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.
- 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.
- Examples of this type of information are:
  - a stock trade
  - a customer buying an airline ticket
- 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



```
Starting MQSC for queue manager TEST1.

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

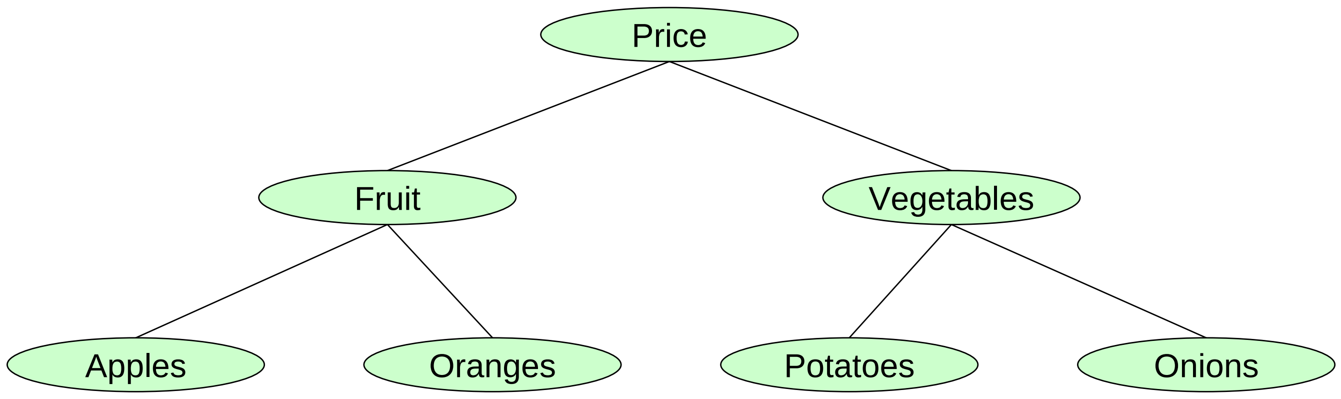
SHARE in Orlando 2011

Publish/Subscribe in WebSphere MQ - Notes

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 (see next page). 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 behaviour of the topic tree by defining topic objects and changing attributes on them. Of course you only need to do this if you want to change the default behaviour. You may not need any topic objects – we will at this in more detail later.
- You can programmatically interface with the topic tree as a subscriber using MQSUB and as a publisher using MQOPEN and MQPUT.
- 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.

Topic strings and topic tree



Price/Fruit/Apples

Price/Fruit/Oranges

Price/Vegetables/Potatoes

Price/Vegetables/Onions

Topic strings and topic tree - Notes

N O T E S

- Topic strings can be any characters you choose. You can, and should, add structure to you 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.

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;
```

Publishing Application - Notes

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.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>                               /* includes for MQI */
void errorMessage( char* msgStr, MQLONG CC, MQLONG RC )
{
  printf( "%s\nCompletion Code: %d\nReason Code    : %d\n",
          msgStr, CC, RC );
}
void usageError( )
{
  printf( "==================================================\n" );
  printf( "Parameters for Publisher Program\n" );
  printf( "     Topic String\n" );
  printf( "     Queue Manager Name (optional)\n" );
  printf( "==================================================\n" );
}
int main(int argc, char **argv)
{
  FILE   * fp  = stdin;            /* Declare file for sample input */
  MQOD     od  = {MQOD_DEFAULT};   /* Object Descriptor             */
  MQMD     md  = {MQMD_DEFAULT};   /* Message Descriptor            */
  MQPMO    pmo = {MQPMO_DEFAULT};  /* put message options           */
  MQHCONN  hConn = MQHC_UNUSABLE_HCONN; /* connection handle        */
  MQHOBJ   hObj  = MQHO_UNUSABLE_HOBJ;  /* object handle            */
  MQLONG   CompCode, Reason;       /* completion and reason code    */
  MQLONG   OpenOpts;               /* MQOPEN options                */
  MQLONG   messlen;                /* message length                */
  char     buffer[100];            /* message buffer                */
  char     QMName[50] = {0};       /* queue manager name            */

  if (argc < 2)
  {
    usageError( );
    return 99;
  }
  if (argc > 2) strcpy(QMName, argv[2]);
  /******************************************************************/
  /* Connect to queue manager                                       */
  /******************************************************************/
  MQCONN(QMName,                  /* queue manager                  */
         &hConn,                  /* connection handle              */
         &CompCode,               /* completion code                */
         &Reason);                /* reason code                    */
  /******************************************************************/
  /* If connect failed, display error message and exit              */
  /******************************************************************/
  if (CompCode == MQCC_FAILED)
  {
    errorMessage("MQCONN", CompCode, Reason);
    goto MOD_EXIT;
  }

  /******************************************************************/
  /* Open the target topic for output                               */
  /******************************************************************/
  od.ObjectType = MQOT_TOPIC;
  od.Version    = MQOD_VERSION_4;
  od.ObjectString.VSPtr    = argv[1];
  od.ObjectString.VSLength = MQVS_NULL_TERMINATED;
  OpenOpts = MQOO_OUTPUT | MQOO_FAIL_IF_QUIESCING;

  MQOPEN(hConn,                     /* connection handle            */
         &od,                       /* object descriptor for topic  */
         OpenOpts,                  /* open options                 */
         &hObj,                     /* object handle                */
         &CompCode,                 /* MQOPEN completion code       */
         &Reason);                  /* reason code                  */
  /******************************************************************/
  /* If open failed, display error message and exit                 */
  /******************************************************************/
  if (CompCode == MQCC_FAILED)
  {
    errorMessage("MQOPEN of a topic", CompCode, Reason);
    goto MOD_EXIT;
```

```
   }

   /******************************************************************/
   /* Read lines from the file and publish them on the topic         */
   /* Loop until null line or end of file, or there is a failure     */
   /******************************************************************/
   memcpy(md.Format, MQFMT_STRING, MQ_FORMAT_LENGTH);
   pmo.Options = MQPMO_FAIL_IF_QUIESCING | MQPMO_NO_SYNCPOINT;

   printf("Enter message text to publish to '%s'\n", argv[1]);
   while (CompCode != MQCC_FAILED)
   {
     if (fgets(buffer, sizeof(buffer), fp) != NULL)
     {
       messlen = (MQLONG)strlen(buffer); /* length without null      */
       if (buffer[messlen-1] == '\n')  /* last char is a new-line    */
       {
         buffer[messlen-1]  = '\0';    /* replace new-line with null */
         --messlen;                    /* reduce buffer length       */
       }
     }
     else messlen = 0;        /* treat EOF same as null line         */

     /****************************************************************/
     /* Publish each buffer to the topic                             */
     /****************************************************************/
     if (messlen > 0)
     {
       MQPUT(hConn,               /* connection handle               */
             hObj,                /* object handle                   */
             &md,                 /* message descriptor              */
             &pmo,                /* default options (datagram)      */
             messlen,             /* message length                  */
             buffer,              /* message buffer                  */
             &CompCode,           /* completion code                 */
             &Reason);            /* reason code                     */
       /**************************************************************/
       /* report any failures                                        */
       /**************************************************************/
       if (CompCode == MQCC_FAILED)
       {
         errorMessage("MQPUT to topic", CompCode, Reason);
         goto MOD_EXIT;
       }
     }
     else break;
   }
MOD_EXIT:
   /******************************************************************/
   /* Close the target topic (if it was opened)                      */
   /******************************************************************/
   if (hObj != MQHO_UNUSABLE_HOBJ)
   {
     MQCLOSE(hConn,                  /* connection handle            */
             &hObj,                  /* object handle                */
             MQCO_NONE,
             &CompCode,              /* completion code              */
             &Reason);               /* reason code                  */
     if (CompCode == MQCC_FAILED)
       errorMessage("MQCLOSE of topic", CompCode, Reason);
   }

   /******************************************************************/
   /* Disconnect from Queue Manager                                  */
   /******************************************************************/
   if (hConn != MQHC_UNUSABLE_HCONN)
   {
     MQDISC(&hConn,                  /* connection handle            */
            &CompCode,               /* completion code              */
            &Reason);                /* reason code                  */
     if (CompCode == MQCC_FAILED)
       errorMessage("MQDISC", CompCode, Reason);
   }
   return(0);
 }
```

Topic Objects

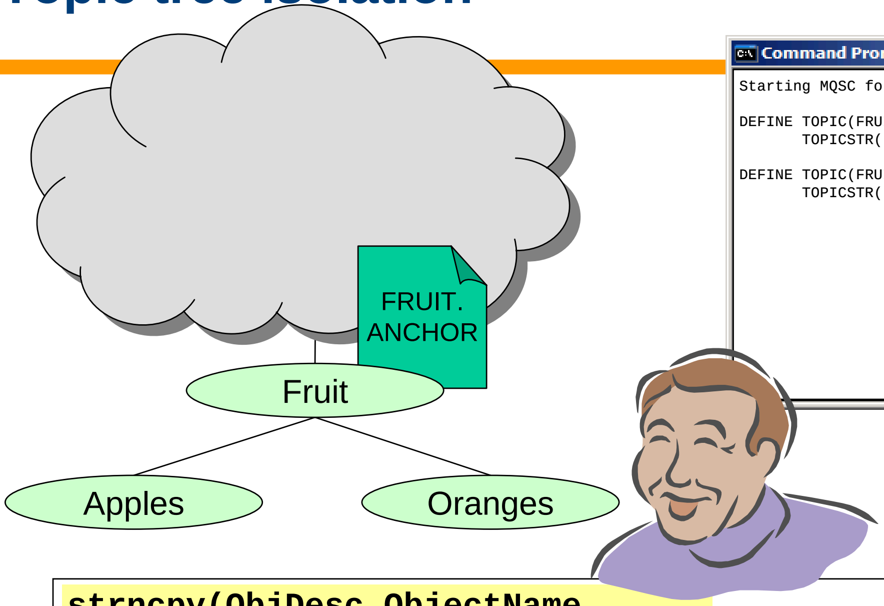
- Not <u>necessary</u> for Publish/Subscribe
- Provide an administrative control point for your topic tree
  - Configuration attributes (including cluster control)
  - Security profiles
  - Topic tree isolation

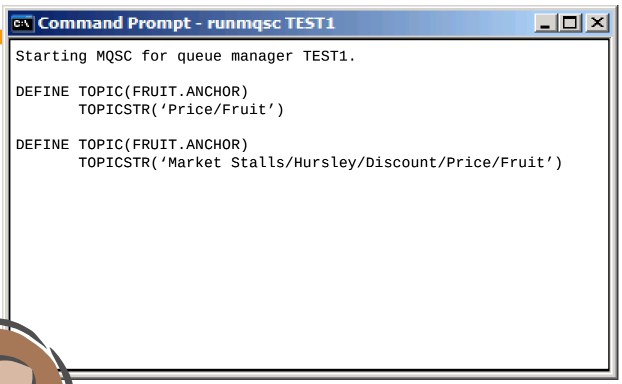
Topic Objects - Notes

N O T E S

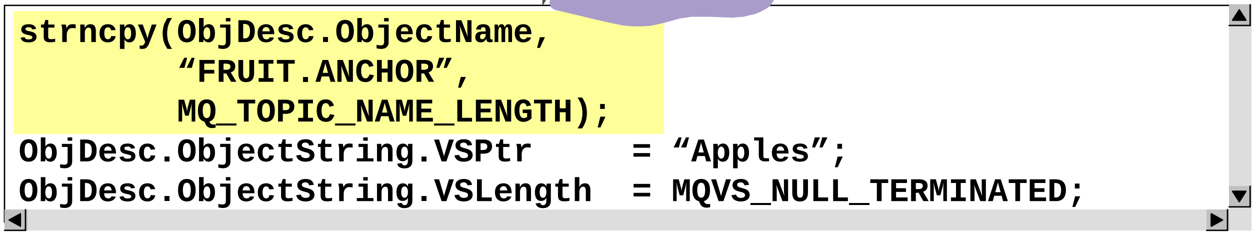
- Topic objects are a new construct in WebSphere MQ V7. They can be used to control the behaviour of your topic tree.
- You do not need to define any topic objects in order to use Publish/Subscribe with WebSphere MQ V7, however you may want to define some if you need to configure the topic tree to use non-default attributes; if you want to apply different security profiles to parts of your topic tree; or if you want to isolate you 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.

Topic tree isolation





```
Starting MQSC for queue manager TEST1.

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

DEFINE TOPIC(FRUIT.ANCHOR)
       TOPICSTR('Market Stalls/Hursley/Discount/Price/Fruit')
```



```
strncpy(ObjDesc.ObjectName,
        "FRUIT.ANCHOR",
        MQ_TOPIC_NAME_LENGTH);
ObjDesc.ObjectString.VSPtr     = "Apples";
ObjDesc.ObjectString.VSLength  = MQVS_NULL_TERMINATED;
```

Topic tree isolation - Notes

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 'Market Stalls/Hursley/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 ( hConn,
        &SubDesc,
        &hObj,
        &hSub,
        &CompCode,
        &Reason);

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

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

```
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;
```

Subscribing Application - Notes

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.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cmqc.h>                              /* includes for MQI  */
void errorMessage( char* msgStr, MQLONG CC, MQLONG RC )
{
  printf( "%s\nCompletion Code: %d\nReason Code    : %d\n",
          msgStr, CC, RC );
}
void usageError( )
{
  printf( "==================================================\n" );
  printf( "Parameters for Subscriber Program\n" );
  printf( "     Topic String\n" );
  printf( "     Subscription Name\n" );
  printf( "     Queue Manager Name (optional)\n" );
  printf( "==================================================\n" );
}
int main(int argc, char **argv)
{
  MQSD     sd  = {MQSD_DEFAULT};   /* Subscription Descriptor       */
  MQMD     md  = {MQMD_DEFAULT};   /* Message Descriptor            */
  MQGMO    gmo = {MQGMO_DEFAULT};  /* get message options           */
  MQHCONN  hConn = MQHC_UNUSABLE_HCONN; /* connection handle        */
  MQHOBJ   hObj  = MQHO_UNUSABLE_HOBJ;  /* object handle            */
  MQHOBJ   hSub  = MQHO_UNUSABLE_HOBJ;  /* subscription handle      */
  MQLONG   CompCode, Reason;       /* completion and reason code    */
  MQBYTE   buffer[101] = {0};      /* message buffer                */
  MQLONG   buflen, messlen;        /* lengths                       */
  char     QMName[50] = {0};       /* queue manager name            */

  if (argc < 3)
  {
    usageError( );
    return 99;
  }
  if (argc > 3) strcpy(QMName, argv[3]);
  /******************************************************************/
  /* Connect to queue manager                                       */
  /******************************************************************/
  MQCONN(QMName,                  /* queue manager                  */
         &hConn,                  /* connection handle              */
         &CompCode,               /* completion code                */
         &Reason);                /* reason code                    */
  /******************************************************************/
  /* If connect failed, display error message and exit              */
  /******************************************************************/
  if (CompCode == MQCC_FAILED)
  {
    errorMessage("MQCONN", CompCode, Reason);
    goto MOD_EXIT;
  }
  /******************************************************************/
  /* Subscribe using a managed destination queue                    */
  /******************************************************************/
  sd.ObjectString.VSPtr    = argv[1];
  sd.ObjectString.VSLength = MQVS_NULL_TERMINATED;
  sd.SubName.VSPtr         = argv[2];
  sd.SubName.VSLength      = MQVS_NULL_TERMINATED;
  sd.Options = MQSO_CREATE
             | MQSO_DURABLE         /* Need a SubName with this     */
             | MQSO_FAIL_IF_QUIESCING
             | MQSO_MANAGED;
  MQSUB(hConn,                      /* connection handle            */
        &sd,                        /* object descriptor for queue  */
        &hObj,                      /* object handle (output)       */
        &hSub,                      /* object handle (output)       */
        &CompCode,                  /* completion code              */
        &Reason);                   /* reason code                  */
  /******************************************************************/
  /* If subscribe failed, display error message and exit            */
  /******************************************************************/
  if (CompCode == MQCC_FAILED)
  {
    errorMessage("MQSUB to a topic", CompCode, Reason);
    goto MOD_EXIT;
  }

  /****************************************************************/
  /* Get messages from the destination queue                      */
  /* Loop until there is a failure                                */
  /****************************************************************/
  gmo.Options = MQGMO_WAIT           /* wait for new messages       */
              | MQGMO_NO_SYNCPOINT   /* no transaction              */
              | MQGMO_CONVERT;       /* convert if necessary        */
  gmo.WaitInterval = 30000;          /* 30 second limit for waiting */

  printf("Publications from topic '%s'\n", argv[1]);
  while (CompCode != MQCC_FAILED)
  {
    buflen = sizeof(buffer) - 1; /* buffer size available for GET   */
    memcpy(md.MsgId,    MQMI_NONE, sizeof(md.MsgId));
    memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));
    MQGET(hConn,               /* connection handle                 */
          hObj,                /* object handle                     */
          &md,                 /* message descriptor                */
          &gmo,                /* get message options               */
          buflen,              /* buffer length                     */
          buffer,              /* message buffer                    */
          &messlen,            /* message length                    */
          &CompCode,           /* completion code                   */
          &Reason);            /* reason code                       */
    /****************************************************************/
    /* report any failures or display the message received          */
    /****************************************************************/
    if (Reason == MQRC_NO_MSG_AVAILABLE) printf("No more messages\n");
    else if (CompCode == MQCC_FAILED)
    {
      errorMessage("MQGET", CompCode, Reason);
      goto MOD_EXIT;
    }
    else printf("message <%s>\n", buffer);
  }
MOD_EXIT:
  /****************************************************************/
  /* Close the subscription handle and managed destination handle   */
  /****************************************************************/
  if (hSub != MQHO_UNUSABLE_HOBJ)
  {
    MQCLOSE(hConn,                   /* connection handle           */
            &hSub,                   /* object handle               */
            MQCO_REMOVE_SUB,         /* close options               */
            &CompCode,               /* completion code             */
            &Reason);                /* reason code                 */
    if (CompCode == MQCC_FAILED)
      errorMessage("MQCLOSE of subscription", CompCode, Reason);
  }
  if (hObj != MQHO_UNUSABLE_HOBJ)
  {
    MQCLOSE(hConn,                   /* connection handle           */
            &hObj,                   /* object handle               */
            MQCO_NONE,               /* close options               */
            &CompCode,               /* completion code             */
            &Reason);                /* reason code                 */
    if (CompCode == MQCC_FAILED)
      errorMessage("MQCLOSE of destination", CompCode, Reason);
  }
  /****************************************************************/
  /* Disconnect from Queue manager                                */
  /****************************************************************/
  if (hConn != MQHC_UNUSABLE_HCONN)
  {
    MQDISC(&hConn,                    /* connection handle          */
           &CompCode,                 /* completion code            */
           &Reason);                  /* reason code                */
    if (CompCode == MQCC_FAILED)
      errorMessage("MQDISC", CompCode, Reason);
  }
  return(0);
 }
```

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

Subscription operation options - Notes

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.

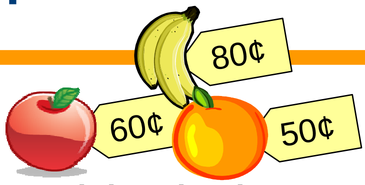
Other attributes of a subscription

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

MB V6 style - # and +

MQ 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;
```

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 - 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 - 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 - Notes

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 you topic string 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.

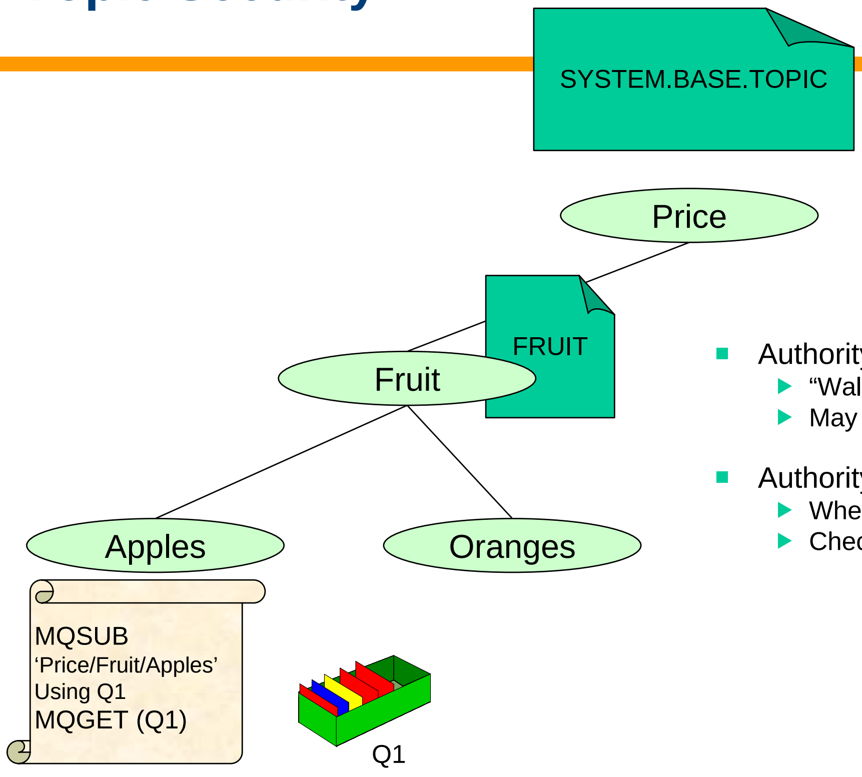
- There are two different wildcard schemes supported in WebSphere MQ V7.
- A character based wildcard scheme which copies the way the queued publish/subscribe interface that existed in WebSphere MQ V6 operated. This uses ‘*’ to replace many characters and ‘?’ to replace one character.
- A topic element 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 - 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.

Topic Security



- Authority check on topic objects
  - "Walk up the tree"
  - May be more than one check
- Authority check on destination queue
  - When not using MQSO_MANAGED
  - Check is for PUT to that queue

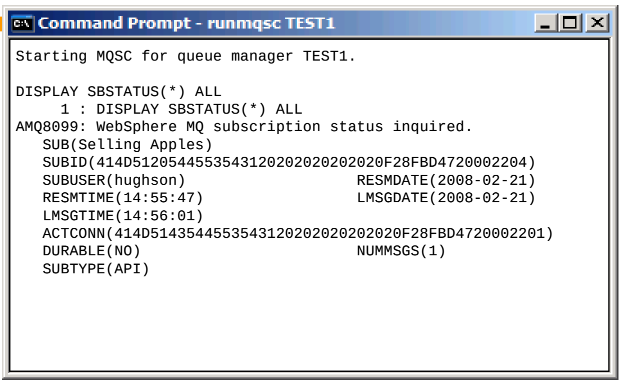
Topic Security

N O T E S

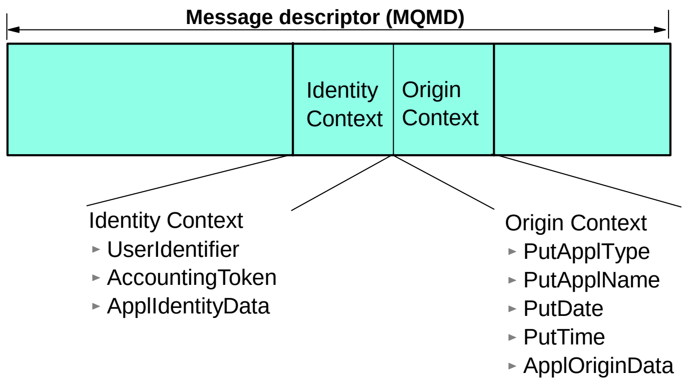
- When MQOPENing a topic (MQOT_TOPIC) for MQOO_OUTPUT – that is, in order to publish, or when making an MQSUB call to subscribe to a topic, a security check is done to see if your user ID has authority to use that topic.
- In our example we have called MQSUB at the point in the topic tree, "Price/Fruit/Apples". There is no topic object at this point in the topic tree, so to find the profile we need to check authorities against we walk up the topic tree to find a node which does have a topic object. The next point is "Price/Fruit". This does have a topic object, FRUIT, so we will check that this user ID has subscribe authority on the profile for the FRUIT topic. If that user ID does have authority, our search stops there. If it does not, we carry on searching up the topic tree and will check the SYSTEM.BASE.TOPIC to see if this user ID has subscribe authority there.
- An additional authorisation check is done for an MQSUB call when the application wishes to use a specific destination queue (i.e. is not using the MQSO_MANAGED option). In this case we also check that this user ID has authority to PUT to that destination queue.

Security on MQSUB call

- Alternate user ID
  - MQSO_ALTERNATE_USER_
    AUTHORITY
- User IDs sharing subs
  - MQSO_FIXED_USERID
  - MQSO_ANY_USERID
- Set Identity context
  - Owning user ID
  - MQSD.PubAccountingToken
  - MQSD.PubApplIdentityData



```
Starting MQSC for queue manager TEST1.

DISPLAY SBSTATUS(*) ALL
     1 : DISPLAY SBSTATUS(*) ALL
AMQ8099: WebSphere MQ subscription status inquired.
   SUB(Selling Apples)
   SUBID(414D5120544553543120202020202020F28FBD4720002204)
   SUBUSER(hughson)                        RESMDATE(2008-02-21)
   RESMTIME(14:55:47)                      LMSGDATE(2008-02-21)
   LMSGTIME(14:56:01)
   ACTCONN(414D5143544553543120202020202020F28FBD4720002201)
   DURABLE(NO)                             NUMMSGS(1)
   SUBTYPE(API)
```

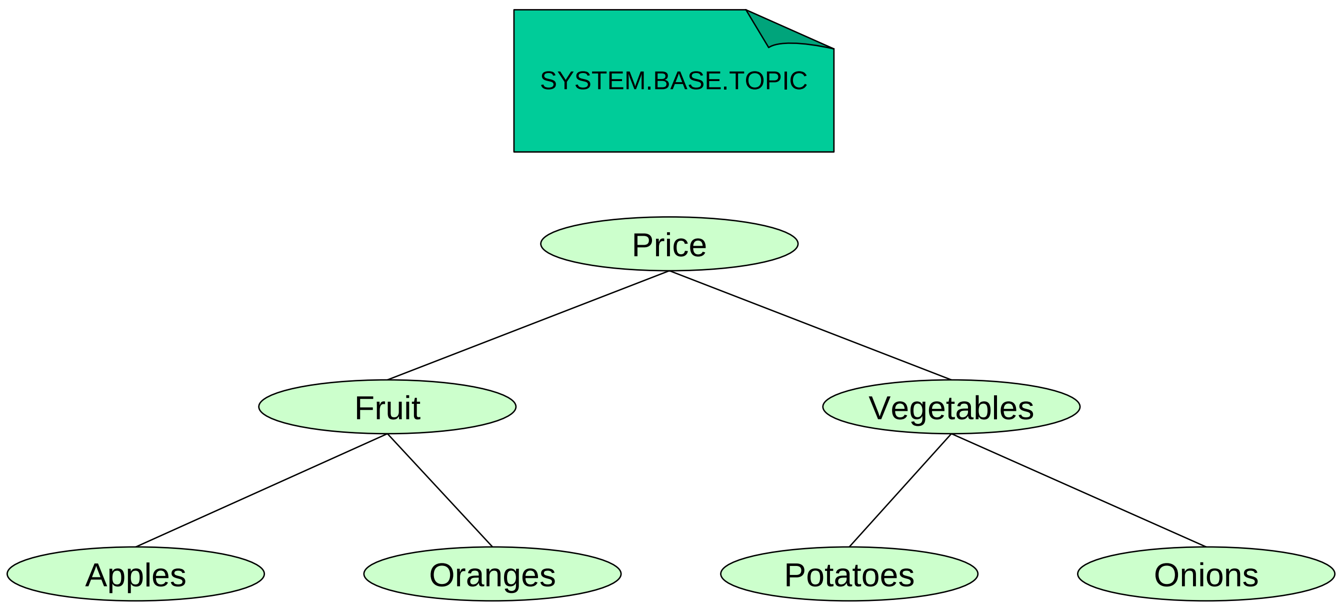


Security on MQSUB call - Notes

N O T E S

- An MQSUB call will only be successful if you have authority to the part of the topic tree you are trying to use, and if you are providing the hObj to the queue manager, have access to PUT messages to the queue. This is covered in more detail in another presentation on MQ Security. However, there are several options and fields in the MQSUB API call which are also relevant to security that we shall coverhere.
- In the same way that you can with MQOPEN, you can use an Alternate user ID on MQSUB. You provide the user ID in the MQSD.AlternateUserId field (and on Windows you may also use the MQSD.AlternateSecurityId field). As with MQOPEN you must have authority to user alternate user IDs to be allowed to do this. The option MQSO_ALTERNATE_USER_AUTHORITY indicates you want to do this.
- When you make a subscription you can choose whether you are willing to share it with another user. Only one user can have a handle to a subscription at a time, but if you use the option MQSO_FIXED_USERID, no other user ID can resume the subscription when you are not using it. The opposite is to say MQSO_ANY_USERID. The default is MQSO_FIXED_USERID. You can see who the owner of a subscription is using MQSC display commands, or equivalent.
- When making a subscription you can choose to set your identity context fields for publications that are sent to you. The option MQSO_SET_IDENTITY_CONTEXT indicates you want to do this and then the MQSD. PubAccountingToken and MQSD.PubApplIdentityData fields become input fields. The other identity context field is your user ID (which may be an alternate one from above). As with MQOPEN you must have appropriate authority to be able to set your identity context.

Base topic Object

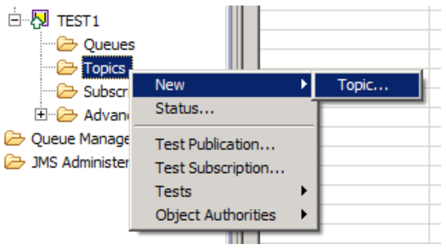


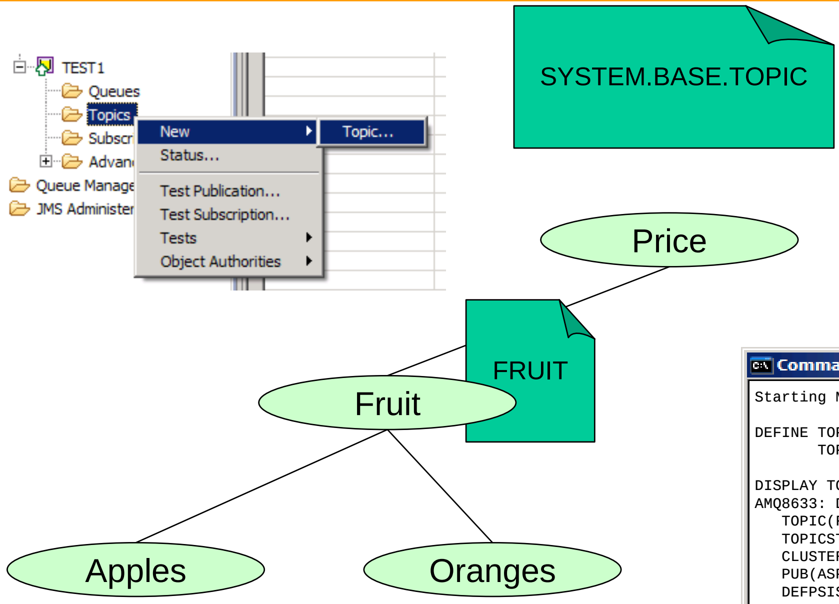
Base topic object

N O T E S

- By default there exists a base topic object, the SYSTEM.BASE.TOPIC that contains all the settings for topic object behaviour. If you define no other topic objects in your system, the behaviour will be taken from this topic object. 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 behaviour you require.
- If you delete this object, 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 need to change that behaviour, you will first need to re-define this object again.

Defining a topic object



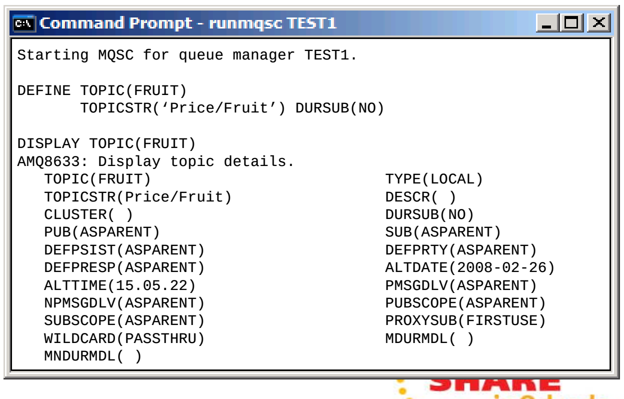


DEFINE TOPIC

ALTER TOPIC

DELETE TOPIC

DISPLAY TOPIC



```
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(2008-02-26)
   ALTTIME(15.05.22)                      PMSGDLV(ASPARENT)
   NPMSGDLV(ASPARENT)                     PUBSCOPE(ASPARENT)
   SUBSCOPE(ASPARENT)                     PROXYSUB(FIRSTUSE)
   WILDCARD(PASSTHRU)                     MDURMDL( )
   MNDURMDL( )
```

Defining a topic object - Notes

N O T E S

- Let's say you need to disallow to 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 behaviour to start, and that behaviour will be inherited by the nodes in the topic tree below that point without the need for any further TOPIC object definitions.
- 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.
- 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.

DEFINE TOPIC

```
                                  +-TYPE(LOCAL)-+
>>--DEFINE TOPIC(topic-name)------+-------------+------- TOPICSTR(string)----->


    +-CMDSCOPE(‘ ’)-------+ (2)    +-QSGDISP(QMGR)--+ (2)
>---+---------------------+-------+----------------+------------------------->
    |                 (1) |       |            (1) |
    +-CMDSCOPE(qmgr-name)-+       +-QSGDISP(COPY)--+
    |          (1)        |       |            (1) |
    +-CMDSCOPE(*)---------+       +-QSGDISP(GROUP)-+

>--+-------------------+--+------------------+--------------------------------><
   +-| define attrs |-+  +-| topic attrs |-+
```

Define attrs:

```
                         +-NOREPLACE-+
|--+-----------------+--+-----------+-----------------------|
   +-LIKE(topic-name)-+  +-REPLACE---+
```

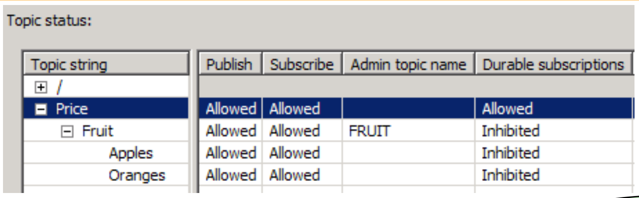
Topic attrs:

```
   +-CLUSTER(‘ ’)----------+  +-DEFPRTY(ASPARENT)-+   +DEFPSIST(ASPARENT)--+
|--+-----------------------+--+-------------------+---+--------------------+-->
   +-CLUSTER(cluster-name)-+  +-DEFPRTY(integer)--+   +DEFPSIST(-+-NO--+-)-+
                                                                 +-YES-+

   +-DEFPRESP(ASPARENT)----+   +-DESCR(‘ ’)----+  +-DURSUB(ASPARENT)--+
>--+-----------------------+---+---------------+--+-------------------+------->
   +-DEFPRESP(-+-SYNC--+-)-+   +-DESCR(string)-+  +-DURSUB(-+-YES-+-)-+
               +-ASYNC-+                                    +-NO--+
               +-NONE--+

   +-MDURMDL(‘ ’)----+  +-MNDURMDL(‘ ’)----+
>--+-----------------+--+------------------+--------------------------------->
   +-MDURMDL(q-name)-+  +-MNDURMDL(q-name)-+

   +-NPMSGDLV(ASPARENT)-------+   +-PMSGDLV(ASPARENT)-------+
>--+--------------------------+---+-------------------------+---------------->
   +-NPMSGDLV(-+-ALL------+-)-+   +-PMSGDLV(-+-ALL------+-)-+
               +-ALLDUR---+                  +-ALLDUR---+
               +-ALLAVAIL-+                  +-ALLAVAIL-+

   +-PROXYSUB(FIRSTUSE)-+  +-PUB(ASPARENT)-------+  +-PUBSCOPE(ASPARENT)---+
>--+--------------------+--+---------------------+--+----------------------+-->
   +-PROXYSUB(FORCE)----+  +-PUB(-+-ENABLED--+-)-+  +-PUBSCOPE(-+-QMGR-+-)-+
                                  +-DISABLED-+                 +-ALL--+

   +-SUB(ASPARENT)-------+  +-SUBSCOPE(ASPARENT)---+  +-WILDCARD(PASSTHRU)-+
>--+---------------------+--+----------------------+--+--------------------+---|
   +-SUB(-+-ENABLED--+-)-+  +-SUBSCOPE(-+-QMGR-+-)-+  +-WILDCARD(BLOCK)----+
          +-DISABLED-+                 +-ALL--+
```

Notes:

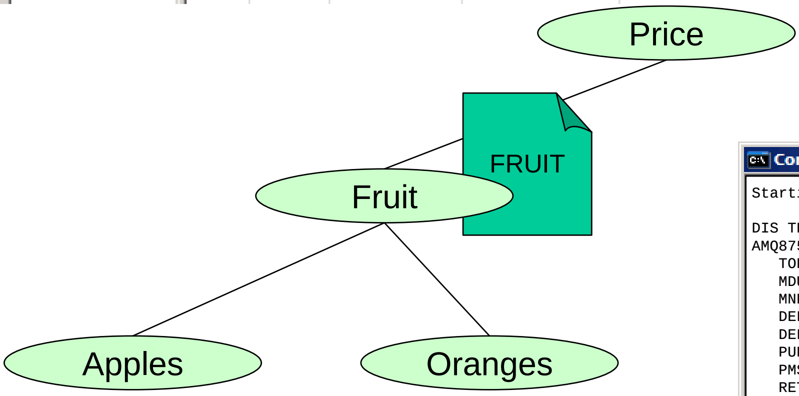
1. Valid only on z/OS when the queue manager is a member of a queue-sharing group.
2. Valid only on z/OS.

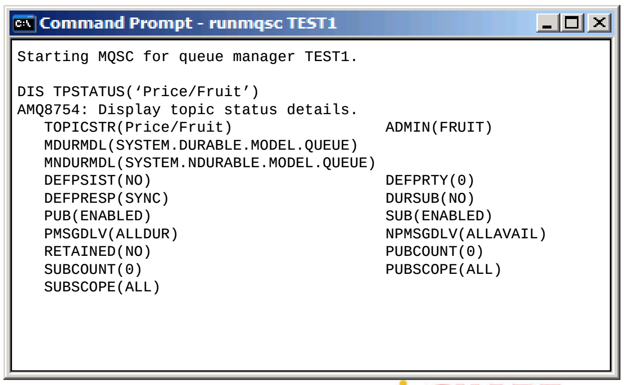
Resolving ASPARENT


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 - Notes

N O T E S

- 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.
- 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.

DISPLAY TPSTATUS

```
>>--DISPLAY TPSTATUS(topic-string)---+--------------------------+--+-----+---->
                                     +-WHERE(-FilterCondition-)-+  +-ALL-+

    +-CMDSCOPE(‘ ‘)-------+ (2)   +-TYPE(TOPIC)-+
>---+---------------------+-------+-------------+------------------------------>
    |                 (1) |       +-TYPE(SUB)---+
    +-CMDSCOPE(qmgr-name)-+       +-TYPE(PUB)---+
    |          (1)        |
    +-CMDSCOPE(*)---------+

>--+------------------+--+-----------------+--+-----------------+-------------><
   +-| topic status |-+  +-| sub status |-+  +-| pub status |-+
```

Topic status:

```
|--+------------------+-------------------------------------------------------|
   | +-,------------+ |
   | V              | |
   +---+-ADMIN----+-+-+
       +-DEFPRESP-+
       +-DEFPRTY--+
       +-DEFPSIST-+
       +-DURSUB---+
       +-MDURMDL--+
       +-MNDURMDL-+
       +-NPMSGDLV-+
       +-PMSGDLV--+
       +-PUB------+
       +-PUBCOUNT-+
       +-PUBSCOPE-+
       +-RETAINED-+
       +-SUB------+
       +-SUBCOUNT-+
       +-SUBSCOPE-+
```

Sub status:

```
|--+------------------+-------------------------------------------------------|
   | +-,------------+ |
   | V              | |
   +---+-ACTCONN--+-+-+
       +-DURABLE--+
       +-LMSGDATE-+
       +-LMSGTIME-+
       +-NUMMSGS--+
       +-RESMDATE-+
       +-RESMTIME-+
       +-SUBID----+
       +-SUBTYPE--+
       +-SUBUSER--+
```

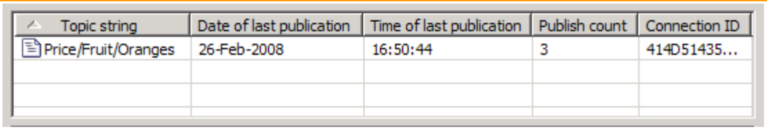
Pub status:

```
|--+------------------+-------------------------------------------------------|
   | +-,------------+ |
   | V              | |
   +---+-ACTCONN--+-+-+
       +-LPUBDATE-+
       +-LPUBTIME-+
       +-NUMPUBS--+
```

Notes:

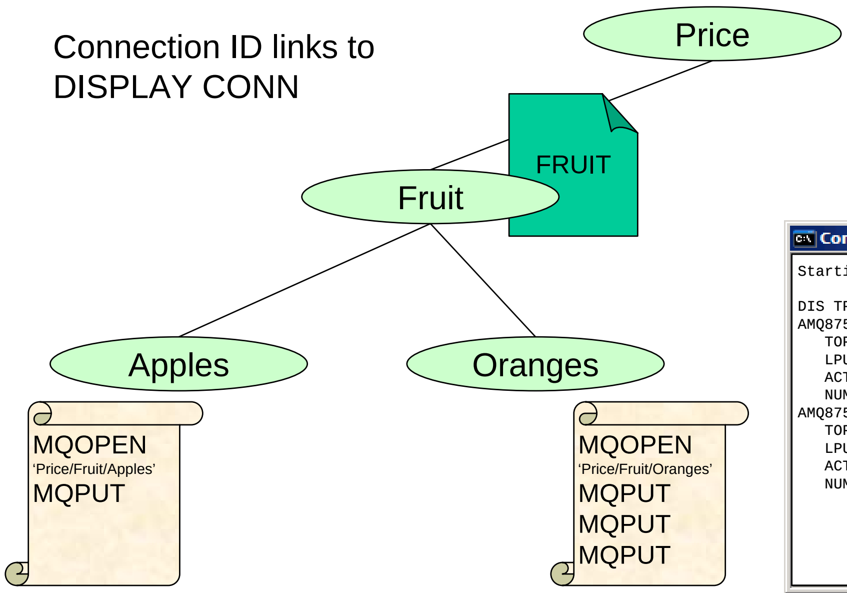
1. Valid only on z/OS when the queue manager is a member of a queue-sharing group.
2. Valid only on z/OS.

Administration for Publishers



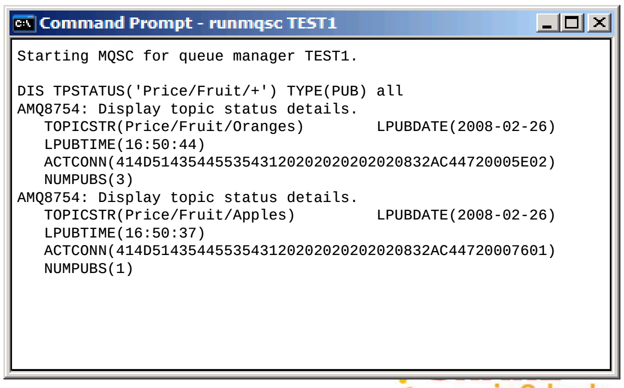
Topic string	Date of last publication	Time of last publication	Publish count	Connection ID
Price/Fruit/Oranges	26-Feb-2008	16:50:44	3	414D51435...

Connection ID links to DISPLAY CONN



TOPIC attributes

DEFPRTY
DEFPSIST
DEFPRESP
PUB
PUBSCOPE
PMSGDLV
NPMSGDLV



```
Starting MQSC for queue manager TEST1.

DIS TPSTATUS('Price/Fruit/+') TYPE(PUB) all
AMQ8754: Display topic status details.
   TOPICSTR(Price/Fruit/Oranges)        LPUBDATE(2008-02-26)
   LPUBTIME(16:50:44)
   ACTCONN(414D514354455354312020202020202020832AC44720005E02)
   NUMPUBS(3)
AMQ8754: Display topic status details.
   TOPICSTR(Price/Fruit/Apples)         LPUBDATE(2008-02-26)
   LPUBTIME(16:50:37)
   ACTCONN(414D514354455354312020202020202020832AC44720007601)
   NUMPUBS(1)
```

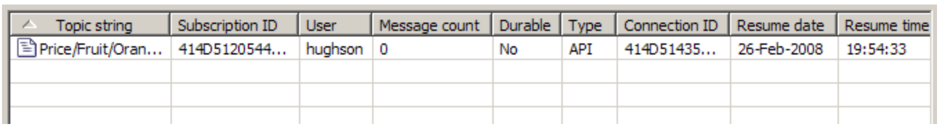
in Orlando
2011

Administration for Publishers - Notes

N O T E S

- 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.
- There are various options on MQPUT that can be left to resolve from the object that was opened. Priority, Persistence and Asynchronous Put Response (new in V7 – discussed in another presentation). 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.
- 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. PUBSCOPE will be discussed later when we cover “Distributed Publish/Subscribe”. PMSGDLV and NPMSGDLV will be covered later when we talk about configuring behaviour for publication failures.
- 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.
- 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.

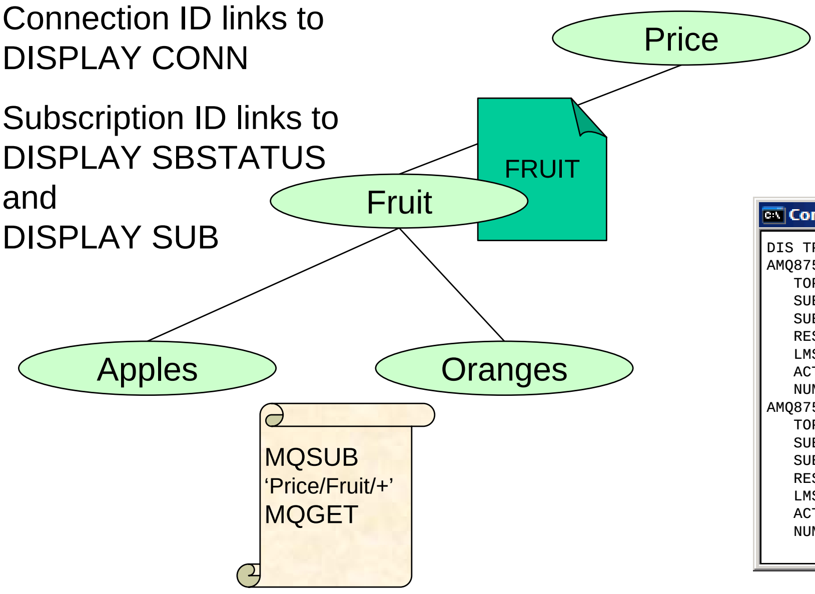
Monitoring your Application subscriptions



Topic string	Subscription ID	User	Message count	Durable	Type	Connection ID	Resume date	Resume time
Price/Fruit/Oran...	414D5120544...	hughson	0	No	API	414D51435...	26-Feb-2008	19:54:33

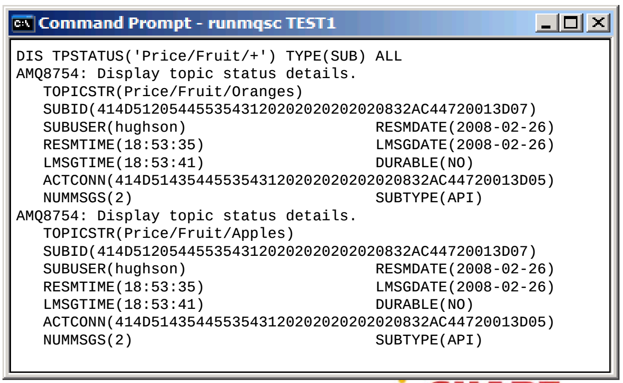
Connection ID links to DISPLAY CONN

Subscription ID links to DISPLAY SBSTATUS and DISPLAY SUB



TOPIC attributes

DURSUB
SUB
SUBSCOPE
PROXYSUB
WILDCARD



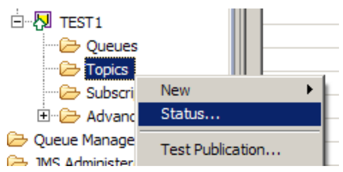
```
DIS TPSTATUS('Price/Fruit/+') TYPE(SUB) ALL
AMQ8754: Display topic status details.
   TOPICSTR(Price/Fruit/Oranges)
   SUBID(414D5120544553543120202020202020832AC44720013D07)
   SUBUSER(hughson)                       RESMDATE(2008-02-26)
   RESMTIME(18:53:35)                     LMSGDATE(2008-02-26)
   LMSGTIME(18:53:41)                     DURABLE(NO)
   ACTCONN(414D5143544553543120202020202020832AC44720013D05)
   NUMMSGS(2)                             SUBTYPE(API)
AMQ8754: Display topic status details.
   TOPICSTR(Price/Fruit/Apples)
   SUBID(414D5120544553543120202020202020832AC44720013D07)
   SUBUSER(hughson)                       RESMDATE(2008-02-26)
   RESMTIME(18:53:35)                     LMSGDATE(2008-02-26)
   LMSGTIME(18:53:41)                     DURABLE(NO)
   ACTCONN(414D5143544553543120202020202020832AC44720013D05)
   NUMMSGS(2)                             SUBTYPE(API)
```

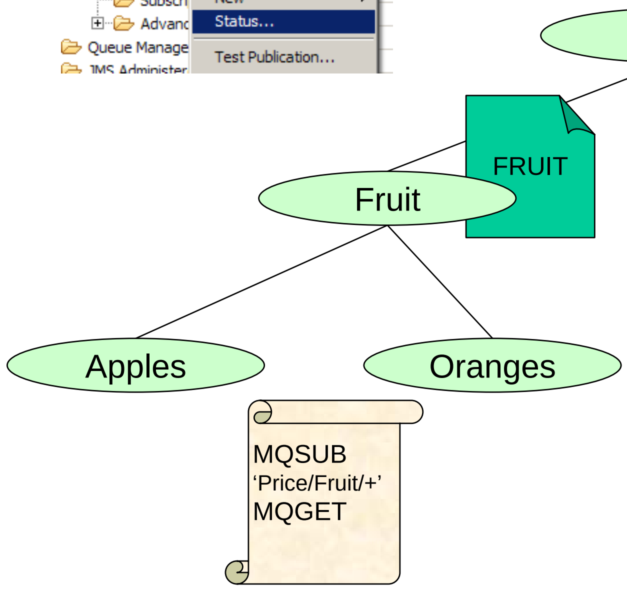
Monitoring your Application subscriptions - Notes

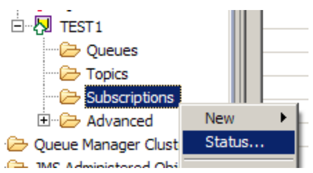
N O T E S

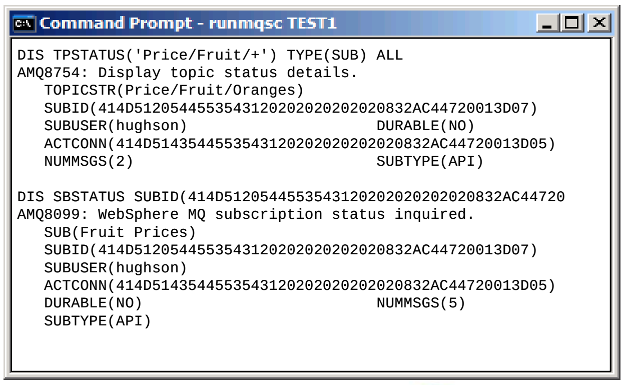
- There are a few attributes on the topic object that are relevant to subscribers. We will look at those here along with the topic status display that shows information about subscribers.
- The TOPIC attribute DURSUB determines whether the creation of durable subscriptions is allowed at this point in 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. SUBSCOPE and PROXYSUB will be discussed later when we cover "Distributed Publish/Subscribe". 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









```
DIS TPSTATUS('Price/Fruit/+') TYPE(SUB) ALL
AMQ8754: Display topic status details.
   TOPICSTR(Price/Fruit/Oranges)
   SUBID(414D5120544553543120202020202020832AC44720013D07)
   SUBUSER(hughson)                         DURABLE(NO)
   ACTCONN(414D5143544553543120202020202020832AC44720013D05)
   NUMMSGS(2)                               SUBTYPE(API)

DIS SBSTATUS SUBID(414D5120544553543120202020202020832AC44720
AMQ8099: WebSphere MQ subscription status inquired.
   SUB(Fruit Prices)
   SUBID(414D5120544553543120202020202020832AC44720013D07)
   SUBUSER(hughson)
   ACTCONN(414D5143544553543120202020202020832AC44720013D05)
   DURABLE(NO)                              NUMMSGS(5)
   SUBTYPE(API)
```

Subscriptions – two perspectives - Notes

N O T E S

- Using DISPLAY TPSTATUS TYPE(SUB) 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. 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) links to DISPLAY SBSTATUS where we will see a single subscription with that ID since the perspective of that display is the subscription.

DISPLAY SBSTATUS

```
>>--DISPLAY SBSTATUS-+-(generic-name)-+--+-------------------------+--+-----+->
                     +-SUBID(string)--+  +-WHERE(-FilterCondition-)-+  +-ALL-+

    +-DURABLE(---ALL---)-+    +-SUBTYPE(---USER----)-+
>---+--------------------+----+----------------------+------------------------>
    +-DURABLE(-+-NO--+-)-+    +-SUBTYPE(-+-PROXY-+-)-+
               +-YES-+                   +-ADMIN-+
                                         +-API---+
                                         +-ALL---+

    +-CMDSCOPE(' ')-------+ (2)
>---+---------------------+-------+-------------------+----------------------><
    |                 (1) |       +-| status attrs |-+
    +-CMDSCOPE(qmgr-name)-+
    |          (1)        |
    +-CMDSCOPE(*)---------+
```

Status attributes:

```
   +-,------------+
   V              |
|----+------------+-----------------------------------------------------------|
     +-ACTCONN--+
     +-DURABLE--+
     +-LMSGDATE-+
     +-LMSGTIME-+
     +-NUMMSGS--+
     +-SUBTYPE--+
     +-RESMDATE-+
     +-RESMTIME-+
```

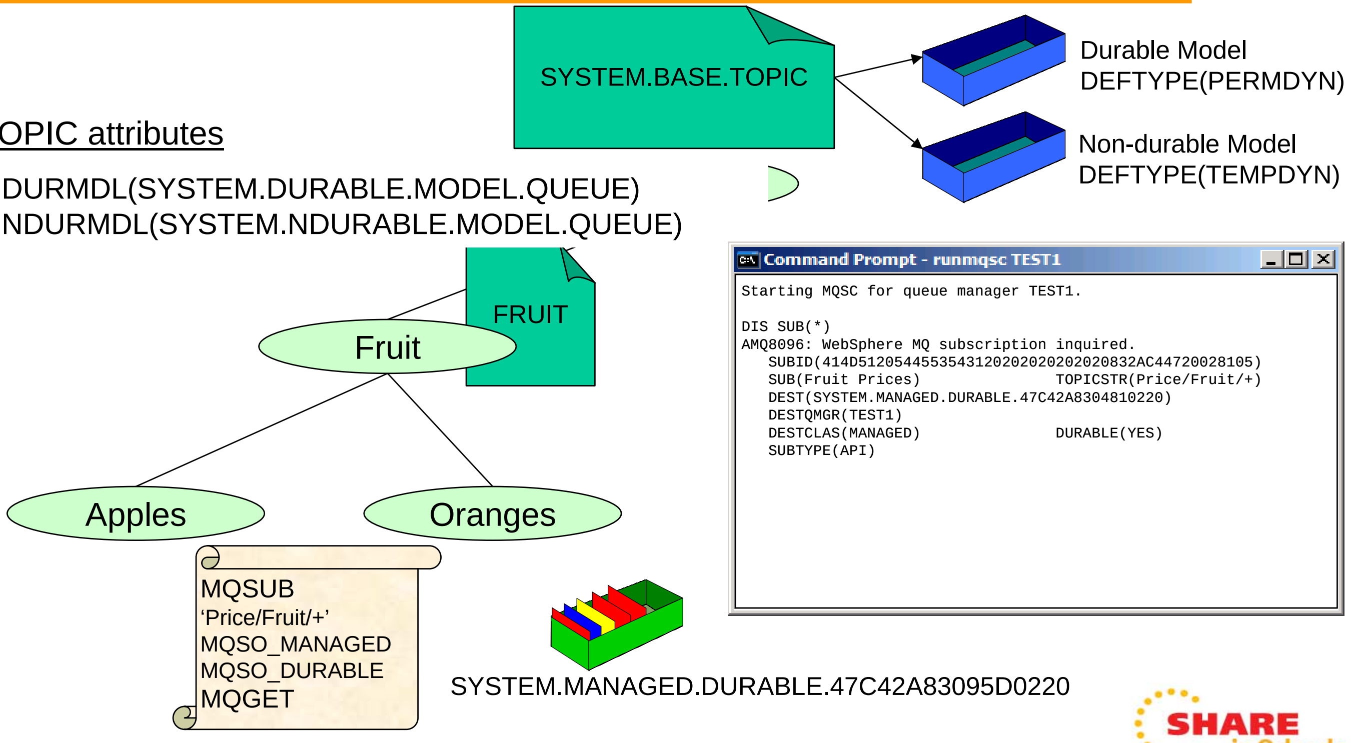
Notes:

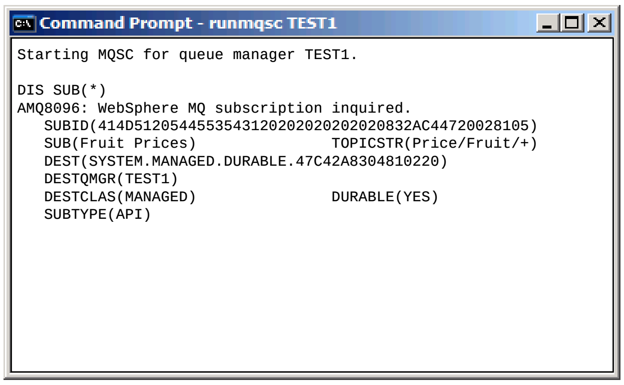
1. Valid only on z/OS when the queue manager is a member of a queue-sharing group.
2. Valid only on z/OS.

Configuring managed destinations

TOPIC attributes

MDURMDL(SYSTEM.DURABLE.MODEL.QUEUE)
MNDURMDL(SYSTEM.NDURABLE.MODEL.QUEUE)





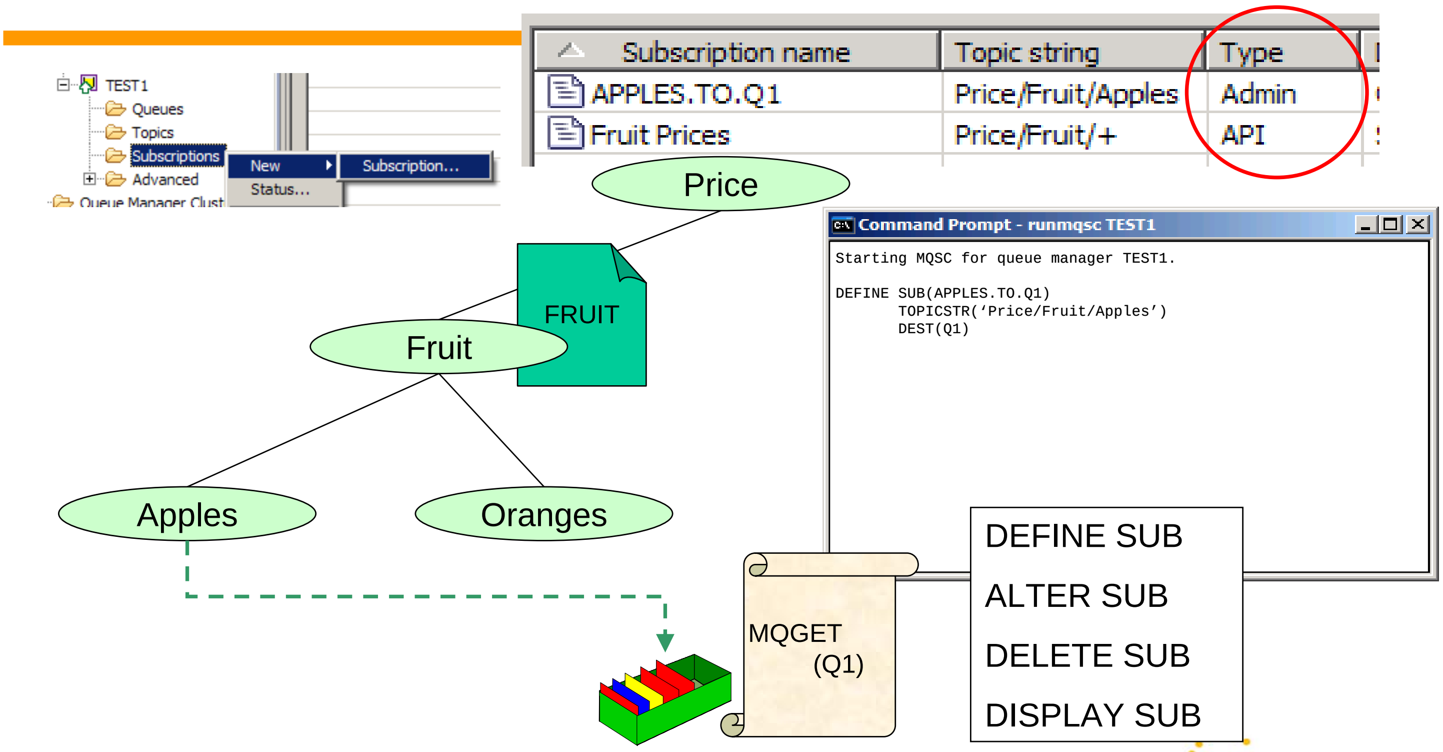
```
Starting MQSC for queue manager TEST1.

DIS SUB(*)
AMQ8096: WebSphere MQ subscription inquired.
   SUBID(414D5120544553543120202020202020832AC44720028105)
   SUB(Fruit Prices)                  TOPICSTR(Price/Fruit/+)
   DEST(SYSTEM.MANAGED.DURABLE.47C42A8304810220)
   DESTQMGR(TEST1)
   DESTCLAS(MANAGED)                  DURABLE(YES)
   SUBTYPE(API)
```

Configuring managed destinations - Notes

N O T E S

- If your applications are using managed destinations for delivery of their publications, the queue that publications reside upon is not something the application has to worry about, but an administrator may wish to configure things about this queue. Managed destinations are dynamic queues and are created based on the model queue defined at the specific point in the topic tree. The SYSTEM.BASE.TOPIC defines the two model queues (one for durable subscriptions and one for non-durable subscriptions) as SYSTEM.DURABLE.MODEL.QUEUE and SYSTEM.NDURABLE.MODEL.QUEUE. These names are provided in the MDURMDL and MNDURMDL keywords on the TOPIC object definition.
- If you don't define any TOPIC objects with model queues in these attributes then all TOPIC objects will inherit these attributes from the SYSTEM.BASE.TOPIC. If you want to over-ride these models at different points in the tree, there is one thing to remember. The model for the durable subscriber (MDURMDL) must be Permanent Dynamic. The model for the non-durable subscriber should be Temporary Dynamic.
- The dynamic queues created for subscribers using MQSO_MANAGED or DESTCLAS(MANAGED) will have a stem of SYSTEM.MANAGED.DURABLE or SYSTEM.MANAGED.NDURABLE depending on the durability of the subscription using it. You can see the queue name being used in DISPLAY SUB and DISPLAY CONN. We will look at the changes in DISPLAY CONN in detail a little later.

Creating administrative subscriptions



Subscription name	Topic string	Type
APPLES.TO.Q1	Price/Fruit/Apples	Admin
Fruit Prices	Price/Fruit/+	API

```
Starting MQSC for queue manager TEST1.

DEFINE SUB(APPLES.TO.Q1)
       TOPICSTR('Price/Fruit/Apples')
       DEST(Q1)
```

Creating administrative subscriptions - Notes

N

O

T

E

S

- You don't have to create subscriptions by coding applications to use MQSUB, you can create them administratively. This means that you can take an application that is coded simply to MQGET from a specific queue and have it consume publications by creating an administrative subscription that sends publications to the queue it is getting from.
- There are DEFINE, ALTER, DELETE and DISPLAY commands for SUB. DELETE SUB may be useful in tidying up durable subscriptions that applications have made and forgotten about – i.e. not called MQCLOSE for them when they were finished with them.
- One thing to note about subscriptions is the SUBTYPE field. If the subscription was created from an application issued MQSUB it will be SUBTYPE(API) but if it were created through an administrative command it will be SUBTYPE(ADMIN).
- We will look at DISPLAY SUB a little later.

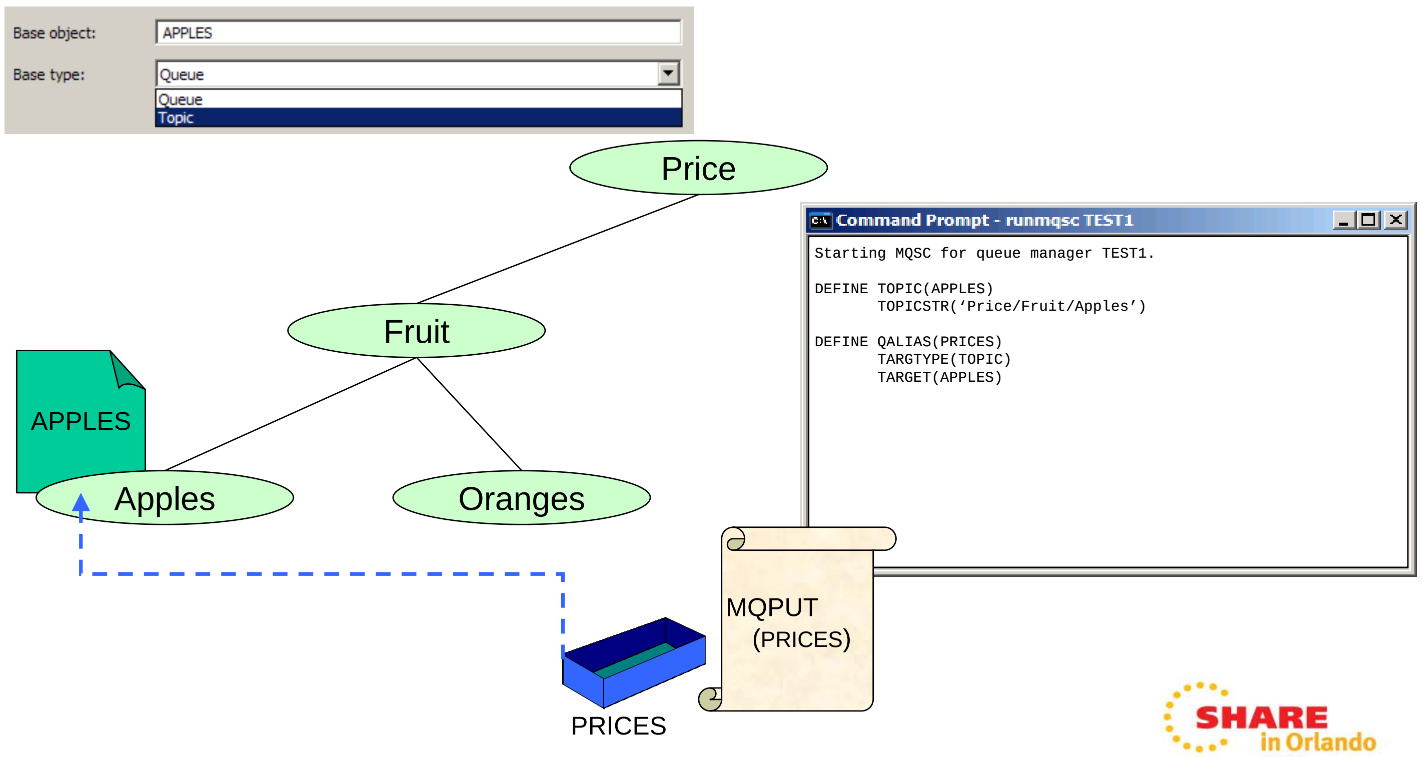
DEFINE SUB

```
                                        +-DESTCLAS(PROVIDED)-+
>>--DEFINE SUB(sub-name)--DEST(q-name)--+--------------------+---------------->
                                        +-DESTCLAS(MANAGED)--+

    +-CMDSCOPE(‘ ‘)-------+ (2)
>---+---------------------+----+---------------------+--+------------------+-->
    |                 (1) |    +-DESTCORL(correl-id)-+  +-DESTQMGR(string)-+
    +-CMDSCOPE(qmgr-name)-+
    |          (1)        |
    +-CMDSCOPE(*)---------+

   +-EXPIRY(UNLIMITED)-+  +-PSPROP(---NONE------)-+
|--+-------------------+--+-----------------------+---+-----------------+----->
   +-EXPIRY(integer)---+  +-PSPROP(-+-COMPAT--+-)-+   +-PUBACCT(string)-+
                                    +-MSGPROP-+
                                    +-RFH2----+

   +-PUBAPPID(‘ ‘)----+   +-PUBPRTY(---ASPUB-----)-+  +-REQONLY(NO)--+
>--+------------------+---+------------------------+--+--------------+-------->
   +-PUBAPPID(string)-+   +-PUBPRTY(-+-ASQDEF--+-)-+  +-REQONLY(YES)-+
                                     +-integer-+

   +-SELECTOR(‘ ‘)----+  +-SUBLEVEL(1)-------+  +-SUBSCOPE(ALL)--+
>--+------------------+--+-------------------+--+----------------+------------>
   +-SELECTOR(string)-+  +-SUBLEVEL(integer)-+  +-SUBSCOPE(QMGR)-+

                     (3)                    (3)
>--+------------------+--+------------------+--------------------------------->
   +-TOPICOBJ(string)-+  +-TOPICSTR(string)-+

                         +-VARUSER(ANY)---+  +-WSCHEMA(TOPIC)-+
>--+------------------+--+----------------+--+----------------+---------------|
   +-USERDATA(string)-+  +-VARUSER(FIXED)-+  +-WSCHEMA(CHAR)--+
```

Notes:

1. Valid only on z/OS when the queue manager is a member of a queue-sharing group.
2. Valid only on z/OS.
3. At least one of TOPICSTR and TOPICOBJ must be present on DEFINE SUB.

Alias Queues



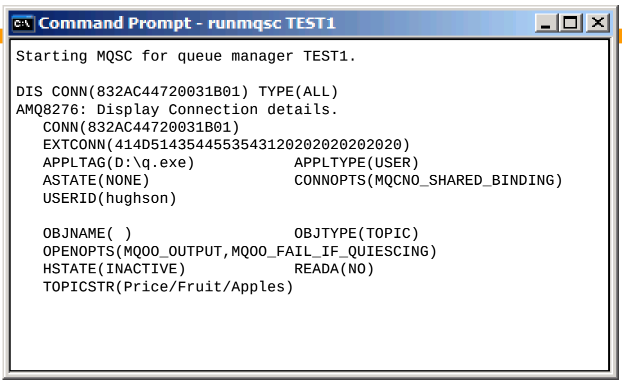
```
Starting MQSC for queue manager TEST1.

DEFINE TOPIC(APPLES)
       TOPICSTR('Price/Fruit/Apples')

DEFINE QALIAS(PRICES)
       TARGTYPE(TOPIC)
       TARGET(APPLES)
```

Alias Queues - Notes

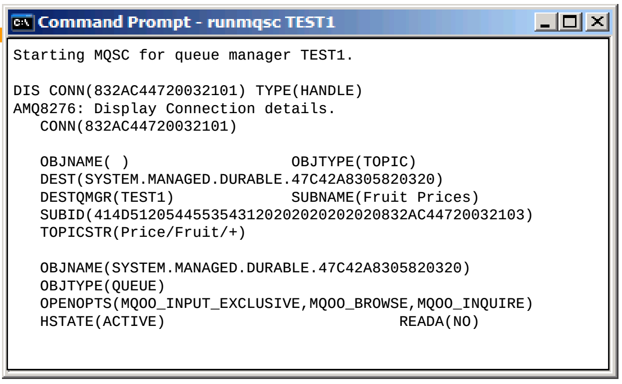
N O T E S

- In the same way that you can make a point-to-point consumer into a consumer of publications simply by means of an administrative command, you can also make a point-to-point producer of messages into a publisher of messages, again by means of an administrative command.
- Changing the queue that the putting application uses into an alias queue which points to a topic, turns that application into 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.
- 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.
- 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.

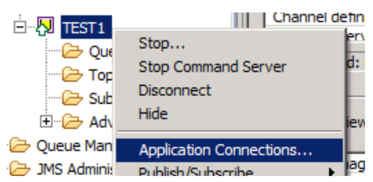
DISPLAY CONN additions

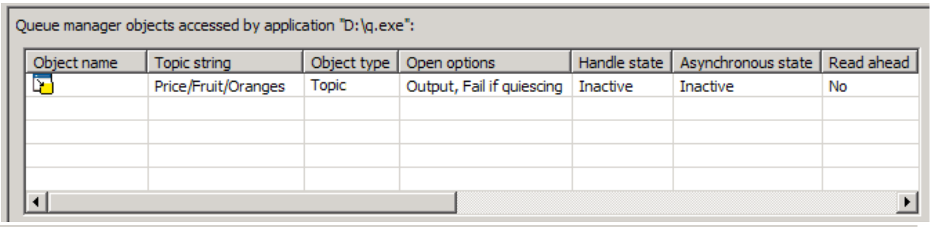


```
Starting MQSC for queue manager TEST1.

DIS CONN(832AC44720031B01) TYPE(ALL)
AMQ8276: Display Connection details.
   CONN(832AC44720031B01)
   EXTCONN(414D5143544553543120202020202020)
   APPLTAG(D:\q.exe)            APPLTYPE(USER)
   ASTATE(NONE)                 CONNOPTS(MQCNO_SHARED_BINDING)
   USERID(hughson)

   OBJNAME( )                   OBJTYPE(TOPIC)
   OPENOPTS(MQOO_OUTPUT,MQOO_FAIL_IF_QUIESCING)
   HSTATE(INACTIVE)             READA(NO)
   TOPICSTR(Price/Fruit/Apples)
```



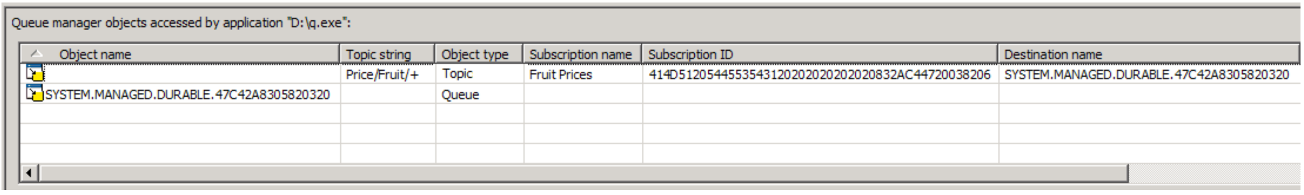
```
Starting MQSC for queue manager TEST1.

DIS CONN(832AC44720032101) TYPE(HANDLE)
AMQ8276: Display Connection details.
   CONN(832AC44720032101)

   OBJNAME( )                   OBJTYPE(TOPIC)
   DEST(SYSTEM.MANAGED.DURABLE.47C42A8305820320)
   DESTQMGR(TEST1)              SUBNAME(Fruit Prices)
   SUBID(414D5120544553543120202020202020832AC44720032103)
   TOPICSTR(Price/Fruit/+)

   OBJNAME(SYSTEM.MANAGED.DURABLE.47C42A8305820320)
   OBJTYPE(QUEUE)
   OPENOPTS(MQOO_INPUT_EXCLUSIVE,MQOO_BROWSE,MQOO_INQUIRE)
   HSTATE(ACTIVE)                        READA(NO)
```





Queue manager objects accessed by application "D:\q.exe":

Object name	Topic string	Object type	Open options	Handle state	Asynchronous state	Read ahead
	Price/Fruit/Oranges	Topic	Output, Fail if quiescing	Inactive	Inactive	No



Queue manager objects accessed by application "D:\q.exe":

Object name	Topic string	Object type	Subscription name	Subscription ID	Destination name
	Price/Fruit/+	Topic	Fruit Prices	414D5120544553543120202020202020832AC44720038206	SYSTEM.MANAGED.DURABLE.47C42A8305820320
SYSTEM.MANAGED.DURABLE.47C42A8305820320		Queue			

DISPLAY CONN additions - Notes

N

O

T

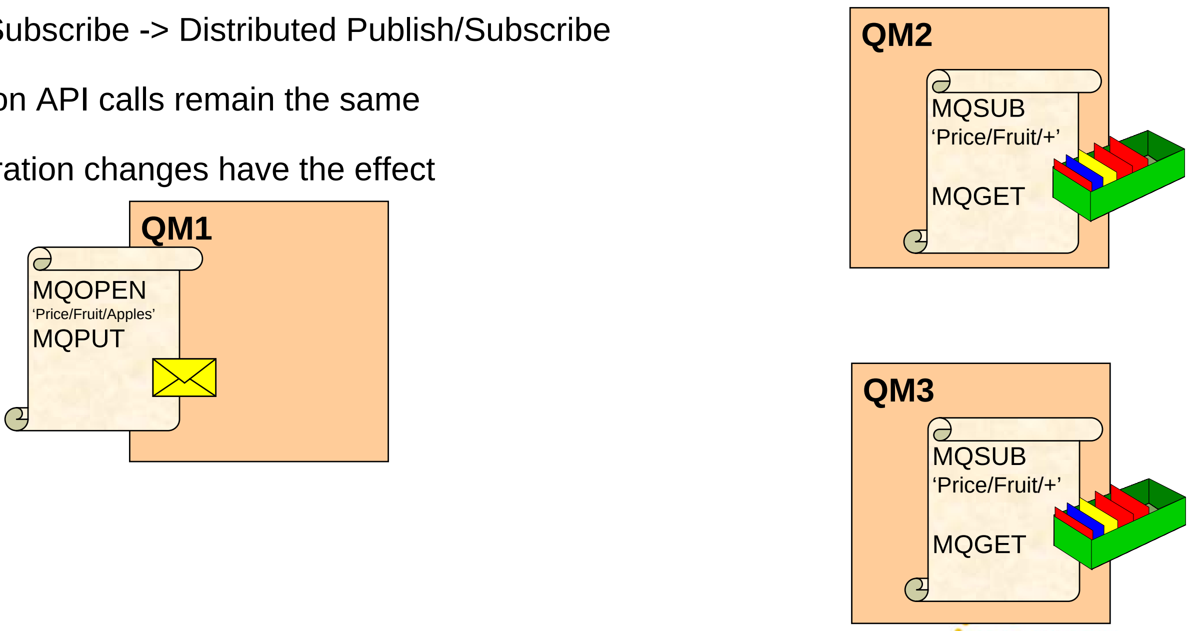
E

S

- DISPLAY CONN provides information about the applications connected to the queue manager and the handles that they have open.
- When an application opens a topic to publish messages to it, you will see this open object handle in DISPLAY CONN.
- When an application subscribes to a topic to receive publications, it is returned a handle to the subscription which you will see in DISPLAY CONN. If the subscription was made using the option MQSO_MANAGED, the handle to the subscription destination queue that has been created by the queue manager for this subscribing application can also be seen in DISPLAY CONN.
- This foils shows some examples of these displays.

Publish/Subscribe Topologies

- Local Queuing -> Distributed Queuing
- Publish/Subscribe -> Distributed Publish/Subscribe
- Application API calls remain the same
- Administration changes have the effect

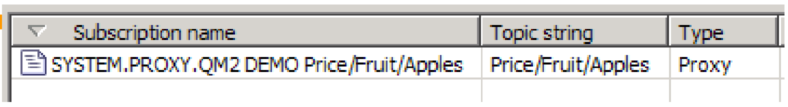


Publish/Subscribe Topologies - Notes

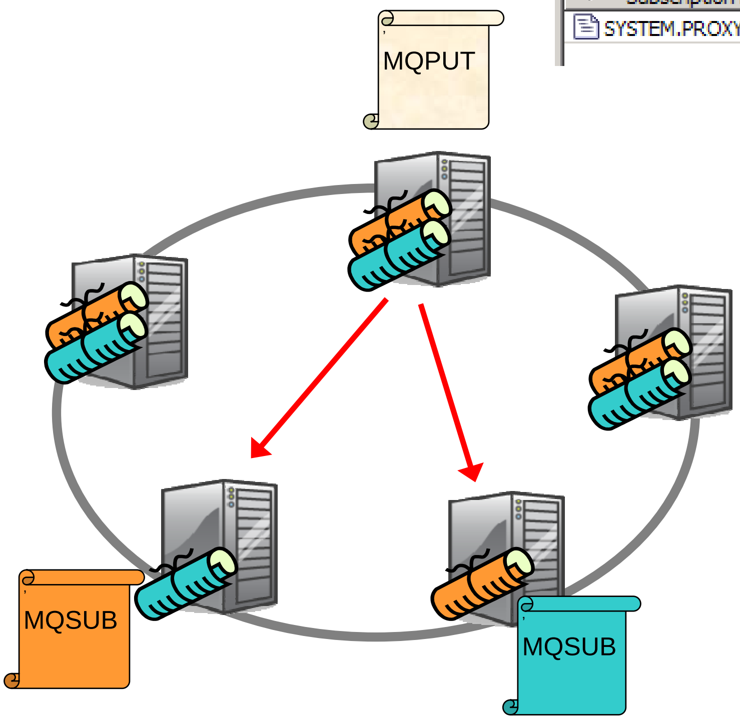
N O T E S

- In this presentation we have concentrated on the publish/subscribe API and Administration within a single queue manager. Of course, just as queuing can be on a single queue manager or can be moving messages between multiple queue managers – known as distributed queuing – so can publish/subscribe. We call this distributed publish/subscribe.
- This is the concept (and the features to implement it) that an application may be publishing to a topic on QM1 and other applications may be subscribing on that topic on others queue managers, here QM2 and QM3, and the publication message needs to flow to those other queue managers to satisfy those subscribers.
- The application code stays the same, you sill call MQSUB or MQOPEN and MQPUT, the difference, as with distributed queuing is in the administrative set-up of your queue managers.
- We are going to look at the different ways you can set up your queue managers to publish messages to another queue manager.

Pub/Sub Clusters

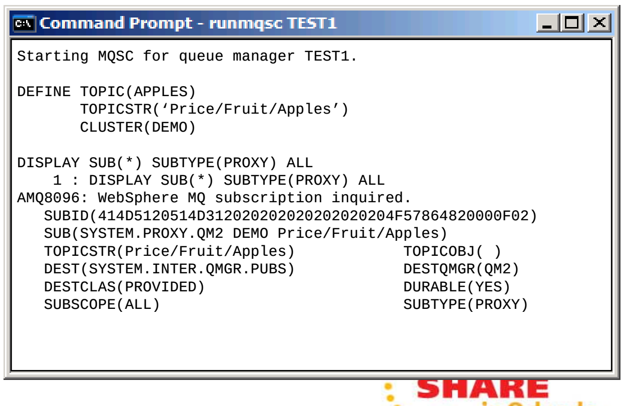


Subscription name	Topic string	Type
SYSTEM.PROXY.QM2 DEMO Price/Fruit/Apples	Price/Fruit/Apples	Proxy



TOPIC attributes

CLUSTER
SUBSCOPE
PUBSCOPE
PROXYSUB



Command Prompt - runmqsc TEST1

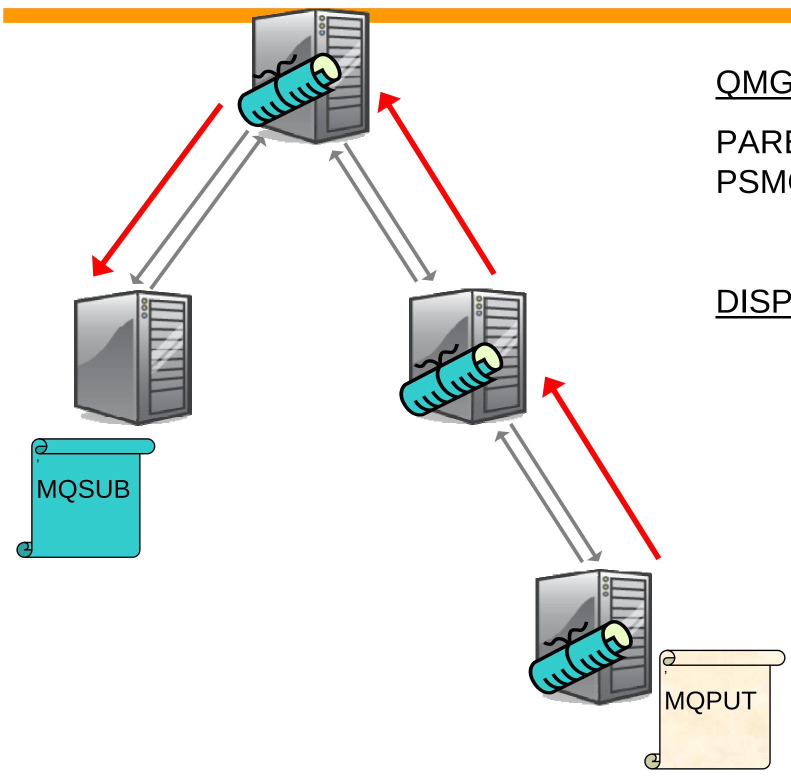
```
Starting MQSC for queue manager TEST1.

DEFINE TOPIC(APPLES)
       TOPICSTR('Price/Fruit/Apples')
       CLUSTER(DEMO)

DISPLAY SUB(*) SUBTYPE(PROXY) ALL
    1 : DISPLAY SUB(*) SUBTYPE(PROXY) ALL
AMQ8096: WebSphere MQ subscription inquired.
   SUBID(414D5120514D31202020202020202020204F57864820000F02)
   SUB(SYSTEM.PROXY.QM2 DEMO Price/Fruit/Apples)
   TOPICSTR(Price/Fruit/Apples)            TOPICOBJ( )
   DEST(SYSTEM.INTER.QMGR.PUBS)            DESTQMGR(QM2)
   DESTCLAS(PROVIDED)                      DURABLE(YES)
   SUBSCOPE(ALL)                           SUBTYPE(PROXY)
```

Pub/Sub Clusters - Notes

N O T E S

- A pub/sub cluster is a cluster of queue managers, with the usual CLUSRCVR and CLUSSDR definitions, but that also contains a TOPIC object that has been defined in the cluster.
- With a cluster you have "any-to-any" connectivity. There are direct links between all queue managers in the cluster. This provides good availability for the delivery of messages, if one route is unavailable, there may well be another route to deliver the messages to the target subscription.
- With a TOPIC object defined in the cluster, an application connected to one queue manager in the cluster can subscribe to that topic or any node in the topic tree below that topic and receive publications on that topic from other queue managers in the cluster.
- This is achieved by the creation of proxy subscriptions on the queue managers in the cluster, so that when a publication to the topic in question happens on their queue manager, they know to forward it to the appropriate other members of the cluster.
- You can view these proxy subscriptions through the same commands we saw earlier. By default proxy subscriptions are not shown to you because the default value for SUBTYPE is USER. If you use SUBTYPE(ALL) or SUBTYPE(PROXY) you will see these subscriptions.
- There are a few attributes that are specifically related to Distributed Publish/Subscribe. PUBSCOPE and SUBSCOPE determine whether this queue manager propagates publications to queue managers in the topology (pub/sub cluster or hierarchy) or restricts the scope to just its local queue manager. You can do the equivalent job programmatically using MQPMO_SCOPE_QMGR / MQSO_SCOPE_QMGR.
- PROXYSUB is an attribute that controls when proxy subscriptions are made. By default it has value FIRSTUSE and thus proxy subscriptions are only created when a user subscription is made to the topic. Alternatively you can have the value FORCE which means proxy subscriptions are made even when no local user subscriptions exist.

Hierarchies



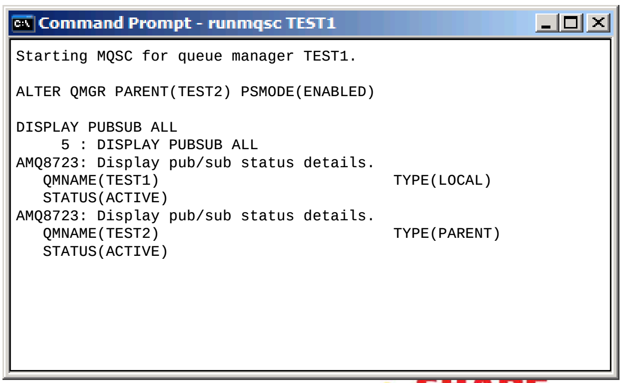
QMGR attributes

PARENT
PSMODE

TOPIC attributes

SUBSCOPE
PUBSCOPE
PROXYSUB

DISPLAY PUBSUB



```
Starting MQSC for queue manager TEST1.

ALTER QMGR PARENT(TEST2) PSMODE(ENABLED)

DISPLAY PUBSUB ALL
     5 : DISPLAY PUBSUB ALL
AMQ8723: Display pub/sub status details.
   QMNAME(TEST1)                          TYPE(LOCAL)
   STATUS(ACTIVE)
AMQ8723: Display pub/sub status details.
   QMNAME(TEST2)                          TYPE(PARENT)
   STATUS(ACTIVE)
```

SHARE in Orlando 2011

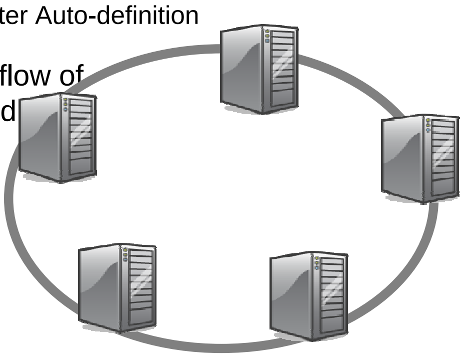
Hierarchies - Notes

N O T E S

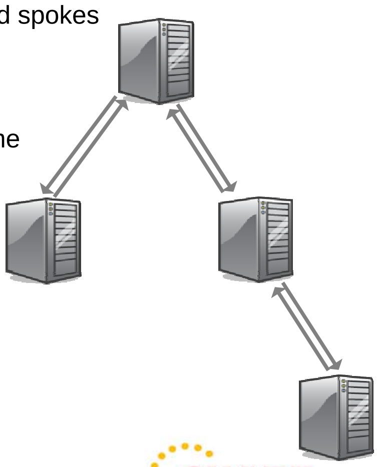
- A hierarchy is a parent/child related set of queue managers. It has less availability than a pub/sub cluster as the availability of an intermediate queue manager has an impact on the connectivity of other queue managers and thus the delivery of publications to satisfy subscribers.
- It is however a solution that scales well due to the fact that there are less connections overall without the all-to-all connectivity that the pub/sub cluster provides.
- In order to set up a hierarchy you nominate your parent queue manager using ALTER QMGR PARENT(queue-manager-name). You don’t need to nominate children, this can be worked out from the fact that they have nominated you as their parent. You can display details about your parent and children queue managers using the DISPLAY PUBSUB command.
- The same topic attributes we looked at on the pub/sub cluster page also apply to hierarchies.

Comparison
Publish/Subscribe Cluster vs Hierarchies

- Similar to WebSphere Message Broker Collectives
- Scalability limited to cluster sizes that can cope with the all-to-all cluster connectivity
- Availability – Direct links between queue managers
- Flexible, low cost administration
  - Cluster Auto-definition
- Control flow of pubs and subs by topic def



- Similar to WebSphere MQ V6 hierarchies
  - Interoperable with them
- Very scalable
- Lack of availability if intermediate queue manager or connections to it fail
  - Highly available hub and spokes
- Inflexible
  - Channels must exist
  - Transmit queues of same name as remote queue manager
- Control flow of pubs and subs by existence of control queues



SHARE in Orlando 2011

Comparison - Notes

N O T E S

- Pub/sub clusters and hierarchies provide different benefits. You should choose the appropriate topology that gives you the benefits you need.
- Pub/sub clusters are very similar to the Message Broker collectives topology in that you have any-to-any connectivity, a fully-connected set. In fact the queue managers are connected together with all-to-all connectivity which means that all the queue managers in the cluster will have channels running to all the other queue managers in the cluster. This may limit the scalability of this solution, with the number of channels being the gating factor. It is however a very available solution since to one queue manager is a bottle-neck or single point of failure for the cluster. It also provides the same benefits of clustering that you get for queued clusters, that of reduces administration definitions.
- Hierarchies are very similar to, and interoperable with, WebSphere MQ V6 Queued Pub/Sub hierarchies. They have interconnection only between parent and child queue managers and so the loss of one queue manager, or connectivity to it, can cause a break in the link for other queue managers. Less availability therefore, but more scalable as less channels are needed for each queue manager. It is less flexible and definitions (channels and transmit queues) need to be set up manually.

Summary - WebSphere MQ Publish/Subscribe

- **Application Programming**
- Publishing
  - MQPUT to topic
    - Object Descriptor (MQOD) extended for use with topics
    - New MQPMO option
- Subscribing
  - MQSUB
    - Subscription Descriptor (MQSD) contains fields and options to control behaviour
  - Consume publications
    - Use MQGET
    - Or Asynchronous Consume

- **Administration**
- Topic tree administration control
  - TOPIC objects
    - Also the security control point
- No code change Publish/Subscribe
  - QALIAS pointing at TOPIC
  - Admin SUB commands
- Application Monitoring
  - DISPLAY CONN updated
  - New Topic Status and Subscription Status commands
- Topologies
  - Pub/Sub Clusters
  - Hierarchies

Summary - Notes

N O T E S

- We have introduced you to the new API verbs, structures and options that allow you to do Publish and Subscribe from the MQ API.
  - Publishers can MQPUT to a topic
  - Subscribers can MQSUB to request publications and then MQGET them. We also have MQSUBRQ to request publications only when we need them.
- We have also covered the ways you can configure your topic tree and monitor you applications’ use of the topic tree.