MQ Publish/Subscribe

An Introduction to Topic Objects, Nodes and Strings (among other things)

Matthew Whitehead
mwhitehead@uk.ibm.com
Agenda

- Publish/Subscribe in IBM MQ
- Administration of publish/subscribe
- Management of publish/subscribe
- Subscriptions and publications
- Quick look at topologies
What is publish/subscribe?
How does it compare to point-to-point?

point-to-point
How does it compare to point-to-point?

**point-to-point**

**publish/subscribe**

Message producer

Message consumer

Message consumer

Message consumer

Subscription

Message consumer

Subscription

Message consumer

Subscription

Message consumer
But which subscriptions receive the messages?

- Publishing and subscribing is based on ‘topics’
  - **Green** messages go to **green** subscribers
  - **Orange** messages go to **orange** subscribers
  - But nobody wants a **blue** message!
Publish/Subscribe in IBM MQ
WebSphere MQ’s publish/subscribe over the years

Publish/Subscribe brokers

WebSphere MQ Publish/Subscribe SupportPac MA0C
IBM Integration Bus® (formerly WebSphere Message Broker®)
WebSphere MQ

WebSphere MQ Publish/Subscribe APIs

MQ V 5.1 MQ V 6 MQ V 7 MQ V 8

WebSphere MQ Publish/Subscribe APIs

Command message based publish/subscribe API
JMS publish/subscribe API
XMS publish/subscribe API
Native MQI publish/subscribe API
Topics
It's all about the **topic tree**

- **Price**
  - **Fruit**
    - **Apples**
      - Topic string: `/Price/Fruit/Apples`
    - **Oranges**
      - Topic string: `/Price/Fruit/Oranges`
  - **Vegetables**
    - **Potatoes**
      - Topic string: `/Price/Vegetables/Potatoes`
Matching publications to subscriptions

- Subscriptions are attached to matching nodes in the topic tree
- Publications identify the relevant topic node
- A copy of the publication is delivered to the queue identified by each matching subscription
• **Wildcarding** subscriptions at the topic node level can receive messages from multiple topic strings
Designing your topic tree structure

- Make it extendable.
- Understand a rapidly changing set of topic strings.
- Avoid excessively wide or deep dynamic topic trees.
  - Use structure where appropriate.
  - Limit it to subscribable content.
Configuration
- Topic objects are a point of administration associated with a node in the topic tree.
- You start with a base object defined for the ‘ ’ node … the rest are optional.
- They provide hook points in the topic tree to configure specific pub/sub behaviour for a branch.
- A dynamically created topic node inherits its attributes from administered topic objects associated with topic nodes above it in the topic tree.
Many attributes can be set on topic objects to effect a publisher or subscriber’s behaviour.

Dynamic nodes inherit their behaviour from nodes above.

Create a topic object for topic string ‘/Price/Fruit’

- DEFINE TOPIC(FRUITOBJ) TOPICSTR(‘/Price/Fruit’) DEFPSIST(YES)

Attributes default to *inherit settings from above* (e.g. ‘ASPARENT’).

- (So by default, a new object does nothing)

Publish a message to topic string ‘/Price/Fruit/Oranges’

- What message persistence to use?
- Are publications enabled?
Access control is set as for queues, but for a defined **topic object**, *not a topic string!*

- Authority checks performed on the topic tree
  - Walk up the tree, just like attributes.
  - Keep checking until an authorisation is found or we run out of topic tree.
Managing topics
Managing topics

- Displaying topic object definitions
  - This shows how administered topic objects are configured
Managing topics

- Displaying the topic tree
  - This shows how the **topic nodes** in the topic tree behave

```sql
DISPLAY TPSTATUS('/Price/Fruit/Apples')

23 : DISPLAY TPSTATUS('/Price/Fruit/Apples')

AMQ8754: Display topic status details.

- TOPICSTR('/Price/Fruit/Apples') ADMIN()
- CLUSTER()
- COMMINFO(SYSTEM.DEFAULT.COMMINFO.MULTICAST)
- MDURMDL(SYSTEM.DURABLE.MODEL.QUEUE)
- MNDURMDL(SYSTEM.NDURABLE.MODEL.QUEUE)
- CLROUTE(NONE)
- DEFPRTY(0)
- DURSUB(YES)
- SUB(ENABLED)
- NPMMSGDLV(ALLAVAIL)
- MCAST(DISABLED)
- SUBCOUNT(1)
- SUBSCOPE(ALL)
- DEFPSIST(YES)
- DEFPRESP(SYNC)
- PUB(ENABLED)
- PMSGDLV(ALLDUR)
- RETAINED(NO)
- MCAST(DISABLED)
- PUBCOUNT(0)
- PUBSCOPE(ALL)
- USEDLQ(YES)
```
Applications
When creating subscriptions or opening topics to publish on, do I use a topic string or a topic object?
- A **topic string**. No, a **topic object**. No, **both**. Actually, er, **any of them!**

So which should I use?
- Using the topic string is probably the easiest, it’s closest to what the application is expecting
  - \( \text{Sub}( - , '/\text{Price}/\text{Fruit}/\text{Apples}' ) \rightarrow /\text{Price}/\text{Fruit}/\text{Apples} \)
- Using a topic object maps the operation to the topic string of that topic object
  - \( \text{Sub}( \text{FRUITOBJ}, '' ) \rightarrow /\text{Price}/\text{Fruit} \)
- If you use both, you get both!
  - The topic string is appended to the topic string of the object
    - \( \text{Sub}( \text{FRUITOBJ}, '\text{Apples}' ) \rightarrow /\text{Price}/\text{Fruit}/\text{Apples} \)

If in doubt, check the topic tree for which nodes are actually being used
Subscriptions
Subscription types

- There are many different types of subscriptions:
  - Administered or application created
  - Durable or non-durable
  - Managed or unmanaged subscription queues

- These different aspects of a subscription can be combined, don’t assume it’s one or the other…
Subscription types

Subscription creation and deletion

- **Application created subscriptions**
  - Applications use an API to dynamically create and delete subscriptions

- **Administratively created subscriptions**
  - An administrator defines subscriptions that can be accessed by applications
  - Applications can either use the publish/subscribe APIs to access these subscriptions or access their associated queue using point-to-point APIs.
## Subscription types

### Subscription lifetime

- **Durable subscriptions**
  - The lifetime of the subscription is independent of any application

- **Non-durable subscriptions**
  - The lifetime of the subscription is bounded by the creating application
    - Subscriptions are automatically deleted when the application closes

<table>
<thead>
<tr>
<th></th>
<th>Durable</th>
<th>Non-durable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Admin</td>
<td>✔️</td>
<td>✗</td>
</tr>
<tr>
<td>App</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>
Subscription queue management

- A subscription maps a topic to a queue. The queue relationship is either explicit or implicit...

- Managed subscription queue
  - The subscription automatically creates and deletes a queue for the use of queuing any matching publications.

- Unmanaged subscription queue
  - When the subscription is created the name and location of an existing queue must be provided by you.

<table>
<thead>
<tr>
<th></th>
<th>Managed</th>
<th>Unmanaged</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Durable</td>
<td>Non-durable</td>
</tr>
<tr>
<td>Admin</td>
<td>✔️</td>
<td>✗</td>
</tr>
<tr>
<td>Application</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>

(Not JMS)
Accessing a subscription’s messages

Via the **subscription**

- An application opens the subscription
  - *A true pub/sub application*
- Works with managed and unmanaged subscription queues
- Limited to one attached consuming application at a time
  - Unless you’re using JMS cloned/shared subscriptions
- Generally better pub/sub status feedback

Via the **queue**

- An application opens the queue associated with the subscription
  - *This is really a point-to-point application*
- Only works with unmanaged subscription queues
- Allows more freedom in what can be done
  - For example, multiple concurrent consuming applications possible from any API
Managing subscriptions

- Displaying subscriptions
  - This shows the subscriptions on a queue manager

```
DISPLAY SUB(SUB1)
2 : DISPLAY SUB(SUB1)
AMQ8096: WebSphere MQ subscription inquired.
SUBID(414D5120514D4752312312020202020202020202306)
SUB(SUB1)                      TOPICSTR(/Price/Fruit/Apples)
TOPICOBJ( )                   DESTQMGR(QMGR1)
DEST( )                      PUBAPPID( )
SELECTOR( )                  USERDATA( )
PUBACCT( )                  DESTCORL(414D5120514D4752312312020202020202020202306)
DESTCORL(QMGR1)             DESTMGR(QMGR1)
DESTQMGR(QMGR1)              PUBAPPID( )
SELECTOR( )                  USERDATA( )
SUBSCOPE(ALL)                SUBLEVEL(1)
SUBTYPE(ADMIN)               VARUSER(ANY)
WSCHEMA(TOPIC)              SUBUSER(xxxx)
CRDATE(2014-04-03)       CRTIME(09:19:15)
ALTDATE(2014-04-03)       ALTTIME(09:19:15)
DISPLAY SBSTATUS(SUB1)
3 : DISPLAY SBSTATUS(SUB1)
AMQ8099: WebSphere MQ subscription status inquired.
SUB(SUB1)
SUBID(414D5120514D4752312312020202020202020202306)
SUBUSER(xxxx)                RESMDATE(2014-04-03)
RESMTIME(09:19:15)            LMSGDATE( )
LMSGTIME( )                  ACTCONN(000000000000000000000000000000000000000000000000)
DURABLE(YES)                  MCASTREL( , )
NUMMSGS(0)                   SUBTYPE(ADMIN)
TOPICSTR(/Price/Fruit/Apples)
DISPLAY QLOCAL(SYSTEM.MANAGED.DURABLE.533D180705230020)
```
Publishing
Publication, success or failure?

- **Point-to-point** is nice and simple:
  - Did the message get onto the queue?
  - Was it persistent and transacted?
Publication, success or failure?

- Point-to-point is nice and simple:
  - Did the message get onto the queue?
  - Was it persistent and transacted?

- Publish/subscribe is not so clear cut…
  - Persistence and transactions still ensures integrity of *successful* publications.
  - But if one or more subscriptions can’t receive the publication, *should the publish fail?*
Publication, success or failure?

- Should those subscriptions impact the others, should the publisher know?
- What if the subscriptions are **durable** and the publication is **persistent**?
- Controlled at the topic level
  - Persistent Message Delivery (**PMSGDLV**) and Non-persistent Message Delivery (**NPMSGDLV**): ALL, ALLDUR, ALLAVAIL
- Don’t forget that being able to DLQ a publication is still counted as a **success**!
  - **USEDLQ** on the topic to fine tune this behaviour.
- And finally, **remember** – when there are no subscriptions, no-one gets it. That’s still a successful publish!
Retained publications

- When a message is published to a topic string, it is delivered to each matching subscription registered at that time.
- Subscriptions created after that point will not receive the message only newly published ones.
- Unless publications are *retained*
Retained publications

- When a message is published to a topic string, it is delivered to each matching subscription registered at that time.
- Subscriptions created after that point will not receive the message only newly published ones.
- Unless publications are retained
- Every time a message is published, the most recent publication for each topic string is retained by the queue manager.
- When a new subscription is created, any matching retained message is delivered to it.

- Take care, using retained can be subtle
- Don’t confuse it with persistent publications
Quick look at topologies
Everything revolves around the topic tree, dynamically built up in a queue manager
Distributed publish/subscribe

- Everything revolves around the topic tree, dynamically built up in a queue manager
- Queue managers can work together to share their topic tree knowledge between them
- Everything revolves around the topic tree, dynamically built up in a queue manager.
- Queue managers can work together to share their topic tree knowledge between them.
- Enabling publications to be propagated to subscriptions on different queue managers.
- The applications stay the same, the changes are at the configuration level.
Distributed publish/subscribe topologies

- Publish/subscribe topologies can either be created as a defined hierarchy or more dynamically as a cluster.
Summary

- Publish/Subscribe in WebSphere MQ
- Administration of publish/subscribe
- Management of publish/subscribe
- Subscriptions and publications
- Quick look at topologies
Legal Disclaimer

- © IBM Corporation 2015. All Rights Reserved.
- The information contained in this publication is provided for informational purposes only. While efforts were made to verify the completeness and accuracy of the information contained in this publication, it is provided AS IS without warranty of any kind, express or implied. In addition, this information is based on IBM’s current product plans and strategy, which are subject to change by IBM without notice. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, this publication or any other materials. Nothing contained in this publication is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software.
- References in this presentation to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. Product release dates and/or capabilities referenced in this presentation may change at any time at IBM’s sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way. Nothing contained in these materials is intended to, nor shall have the effect of, stating or implying that any activities undertaken by you will result in any specific sales, revenue growth or other results.
- All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics may vary by customer.