Chapter 28.

April 2009

Welcome to the April 2009 edition of IBM InfoSphere Information Server Developer's Notebook. This month we answer the question;

How and why do I read or write data from a message queue?

Excellent question! This month's author just completed a customer proof where we made heavy use of message queues, using the IBM software product MQ/Series Server to deliver and manage same. Using the common connector architecture of Information Server, it was almost too easy to accomplish our goals. To Information Server, MQ looks like just another data source.

The primary Information Server component used in this edition of IISDN is DataStage, with a brief overview/mention of the Information Server Federated Server and Information Server Classic Federated (Server) components. This edition of IISDN also offers a good sized overview of MQ/Series; installation, configuration, and use.

Software versions

All of these solutions were *developed and tested* on (IBM) InfoSphere Information Server (IIS) version 8.1, using the Microsoft Windows XP/SP2 platform to support IIS client programs, and a RedHat Linux Advanced Server 4 (RHEL 4) FixPak U6 32 bit SMP server (Linux kernel version 2.6.9-67.EL-smp) to support the IIS server side components.

IBM InfoSphere Information Server allows for a single, consistent, and accurate view of data across the full width of the corporate enterprise, be it relational or non-relational, staged or live data. As a reminder, the IBM InfoSphere Information Server product contains the following major components;

WebSphere Business Glossary Anywhere™, WebSphere Information Analyzer™, WebSphere Information Services Director™, WebSphere DataStage™, WebSphere QualityStage™, WebSphere Metadata Server and Metabridges™, WebSphere Metadata Workbench™, WebSphere Federation Server™, Classic Federation™, Event Publisher™, Replication Server™, InfoSphere Data Architect™, DataMirror Transformation Server™, and others.

Obviously, IBM InfoSphere Information Server is a large and capable product, addressing many strategic needs across the enterprise, and supporting different roles and responsibilities.

28.1 Terms and core concepts

According to Wikipedia.com (http://en.wikipedia.org/wiki/MQ_Series), IBM WebSphere MQ (formerly MQ/Series), is middle-ware (software) that "allows independent and potentially non-concurrent applications on a distributed system to communicate with each other". On some level then, MQ/Series allows for connectivity between disparate systems. With its guaranteed message delivery, MQ/Series could also be viewed as a heterogeneous transaction coordinator. In this edition of (IBM) InfoSphere Information Server Developer's Notebook (IISDN), we will detail setup and use of just the MQ/Series Server, and then also sending and receiving messages within the Information Server component of DataStage, to and from MQ.

As mentioned above, this month's author just completed a customer proof that made heavy use of MQ/Series messaging, while at the same time using Information Server change data capture, and federated server components.

Database gateways, federated database servers

In Figure 28-1 below, there are two groupings of servers; those on the left side of the dotted line (inside the walls of our supposed corporation), and those on the right side of the dotted line, (third party providers, our parts suppliers, transport agents, other). The red arrows indicate connectivity, basically point to point communication.

In the real world, you probably wouldn't allow the red arrows, point to point connectivity with the outside world. In the real world, you would direct connectivity to the outside world via a centralized gateway; a firewall (network router), and/or network appliance to control access, check for viruses, malicious attempts to communicate, and more. The network appliance can also cache static content like image files and related. This type of optimized access and control is represented by the blue arrows.

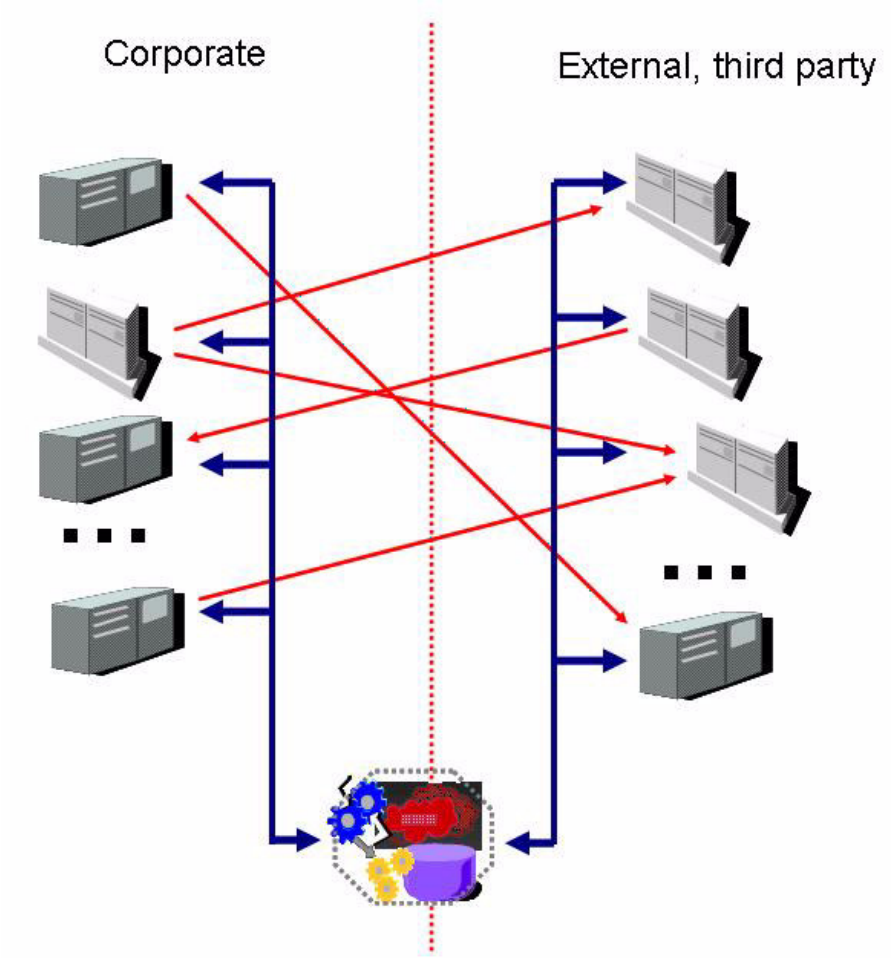

Figure 28-1 Gateways, federated database servers.

Similar to a network appliance or firewall for all outside network traffic, you might choose to make use of a *database gateway* for the same general purpose. A database gateway, or *federated database* (also: *federated server*) allows for numerous benefits that are specialized to database use, including;

- Increased security-

  Centralized control, hardened security, permissions and reporting, audit-trail, recoverability.

- Increased data quality and administrator/user productivity-

Centralized mapping, in the presence of heterogeneous data sources, the need to convert EBCDIC to ASCII, relational to hierarchical and more, basic column data type casting, INTEGERS to CHARACTER and data formatting and enrichment as needed.

– Increased connectivity-

A federated server can join hierarchical (IMS, IDMS, etcetera) data, to relational data (Informix, Oracle, etcetera), to real time news feeds (RSS), and more.

A federated server can also provide connectivity to MQ/Series Server, allowing access to MQ as though it were a standard relational table.

Note: In this edition of IISDN, we overview install and configuration of MQ/Series, its basic object hierarchy, and more. Then we detail direct connectivity between MQ/Series and the Information Server DataStage component.

– Increased source and target system performance-

Rather the hit the source database repeatedly, data can be aggregated or just plain cached at the federated server level. Through built in data replication, this data can be accurate in real time.

– Distributed query access and increased performance-

Given a query the reads two or more source systems, including disparate system types (heterogeneous system types), the federated server can perform query re-write and query optimization, to achieve higher query performance than without use of a federated server. Huh? Given the distributed query,

```
SELECT *
FROM IMS.table, Oracle.table
WHERE IMS.table.column = Oracle.table.column
AND IMS.table.column = X;
```

A federated server will use numerous optimization techniques, in this case, transitive dependency, to re-write the above query as,

```
...
WHERE IMS.table.column = Oracle.table.column
AND IMS.table.column = X
AND Oracle.table.column = X;
```


Note: A really, really good book (our favorite) detailing query optimization can be found at; http://www.redbooks.ibm.com/abstracts/sg247138.html?Open

By author, Daniel M. Farrell, et al., be certain to read chapters 10 and 11 on the topic of query optimizers.

The federated server knows it can apply the "WHERE col = X" predicate to Oracle, since the Oracle column value equals the IMS column value. The federated server will also know the general performance and resource availability of Oracle, IMS, or the federated server itself, and perform the join where it makes most sense.

– Transaction integrity and control-

  The federated server can perform the function of heterogeneous (distributed) transaction coordinator. Huh? The federated server can ensure that an update to Oracle happens entirely or not at all, in direct coordination with a change to IMS.

– Real time change data capture, including in flight data enrichment and aggregation-

  A federated server can capture real time, transaction log based events and replicate data locally or to remote servers. The federated server can also apply data cleansing and transformation, and more.

– Other.

Note: Based on your exact application and need, we will state that all of the above program capabilities are found within IBM InfoSphere Information Server.

To read more about federated databases as a general or research topic, see; http://en.wikipedia.org/wiki/Federated_database_system

IBM WebSphere MQ (formerly MQ/Series), installation

Like any software environment, MQ/Series has its object hierarchy, standard user interfaces, and procedures that detail how you use and interact with MQ. In this section of this edition of IISDN, we offer a brief introduction to MQ/Series Server, so that you may perform basic MQ/Series tests, and related. A good general primer to MQ can be found at,

http://www.redbooks.ibm.com/abstracts/redp0021.html?Open

As detailed on the first page of this document, we run the Information Server proper on a RedHat Linux operating system. Accordingly, we chose to run MQ/Series Server on this same Linux box. Using the RedHat Package Manager (rpm: the operating system subsystem to version and coordinate software installs), we installed the MQ/Series Server via the following command,

```
rpm -ivh MQSeriesClient-6.0.0-0.i386.rpm MQSeriesJava-6.0.0-0.i386.rpm \
  MQSeriesMan-6.0.0-0.i386.rpm MQSeriesRuntime-6.0.0-0.i386.rpm \
  MQSeriesSamples-6.0.0-0.i386.rpm MQSeriesSDK-6.0.0-0.i386.rpm \
  MQSeriesServer-6.0.0-0.i386.rpm
```

Each of the words above, denoted by white space, specifies a given MQ/Series Server install package, each with its own purpose. Due to a missing Linux operating system resource, we chose not to install the given MQ/Series Server package entitled,

```
MQSeriesIES30-6.0.0-0.i386.rpm
```

The above package would have given us graphical clients to test and administer the MQ/Series Server. Since we were modelling (testing) for activities to be performed at our customer site, and this customer site was fanatic about extra open communication ports and the like, we didn't mind not having the MQ/Server graphical client program. Besides, its always good to know the command line interface to something; comes in handy when you're least expecting it.

Note: The MQ/Series server software is installed by the Linux/Unix user id of root. Unless otherwise specified below, all MQ/Series administration is done by the mqm user id.

Note: Generally there is the MQ/Series Server distribution (group of packages), and then the MQ/Series Client. The server is what you would expect; the MQ/Series Server proper. The client is also what you would expect; a number of command line and graphical programs to affect change within the MQ/Series Server.

There are operating conventions as to where the MQ/Series Server must reside in a multi-server topology, including optimization to prepare for any server failure and continued application operation. In this edition of IISDN, we detail only co-location (same box) of the Information Server proper, and the MQ/Series Server proper. This is not a required configuration (co-location), but would likely be the most functional or most performant.

Lastly, we only detail (as usual) the Information Server component of DataStage Enterprise Edition (EE), not the lesser version of DataStage Server.

The default product installation directory for MQ/Series Server on the Linux platform is,

```
/opt/mqm/
```

The default product install will link MQ/Series Server binary executables to /usr/bin/, so no change is required to the standard Linux/Unix PATH environment variable.

MQ/Series Server log files are organized into sub-directories, one sub-directory per queue manager, (an object that is defined below). Given a queue manager name of Bob, log files are then located in,

```
/var/mqm/qmgrs/Bob/errors/
```

MQ/Series object hierarchy, a queue manager

Figure 28-2 displays the terms in the MQ/Series Server object hierarchy. Generally the terms in blue boxes are physical terms; meaning they exist as a computer process or similar object. Generally the terms in white or grey are logical terms, existing as definitions or resources within the MQ landscape.

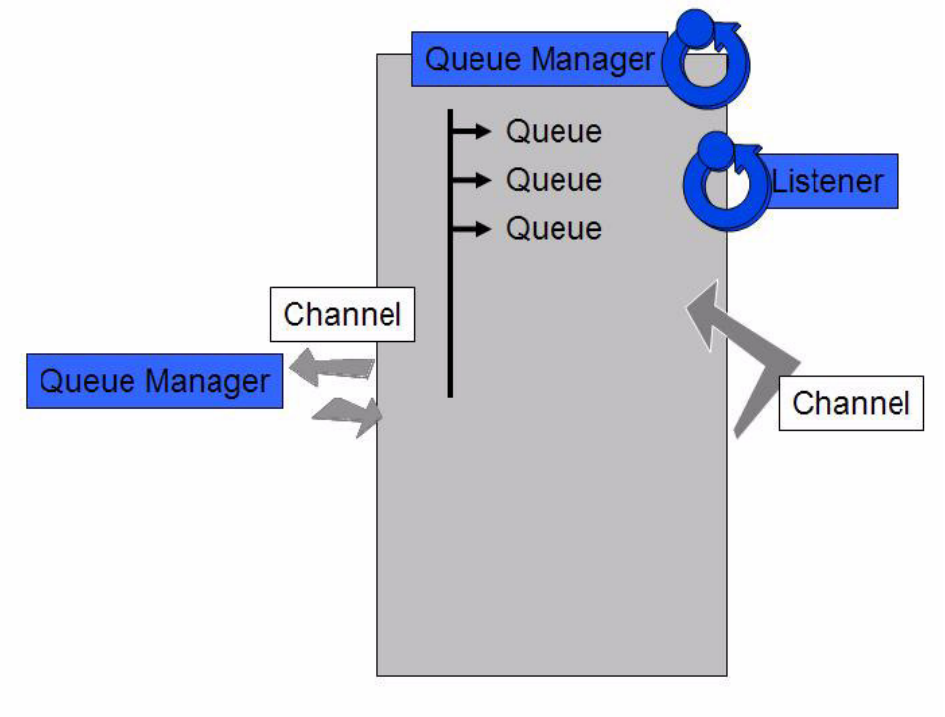

Figure 28-2 MQ/Series Server object hierarchy.

On a given operating system, the MQ/Series Server software is installed once. Similar to the concept of a database server instance, MQ/Series Server has an object entitled, queue manager. You may have several concurrently operating queue managers on one operating system. The queue manager provides common services, like logging events to text files, and acts as a container for several sub-entities described below. The queue manager exists physically as an operating system process.

The command line executable program names to operate a queue manager include;

```
# Create an MQ/Series queue manager; (examples continue using the
# queue manager name, RHHOST_QM, a variable name we chose.)
crtmqm   RHHOST_QM


# Delete an MQ/Series queue manager.
dltmqm   RHHOST_QM
```



```
# Start a queue manager.
strmqm   RHHOST_QM

# Stop a queue manager.
endmqm   RHHOST_QM

# On Linux/Unix, to determine if the queue manager is running.
ps   -ef   |   grep   amq
```

MQ/Series object hierarchy, a listener

An MQ/Series listener is the actual MQ/Series server entity that receives any inbound communication. A listener is wholly contained within a queue manager, and a given queue manger may have several active (concurrent) listeners. The listener exists as an operating system process.

Because the queue manager is associated with a given operating system host, and then an IP address, the listener is associated with that same IP address. The listener is also associated with a port number, and a communication protocol like TCP/IP.

The commands to operate a listener include;

```
# Start a listener with protocol TCP/IP, on port 1414, for the queue manager
# named RHHOST_QM
runmqlsr   -t   tcp   -p   1414   -m   RHHOST_QM   &

# To stop all listeners for a given queue manager
endmqlsr   -m   RHHOST_QM
```

MQ/Series object hierarchy, a queue, channels

If the MQ/Series Server queue manager is the office building, and the listener is the door man, then a queue is the meeting room name where packages (messages) are sent and retrieved (read, opened).

If you were sending change data capture messages from 4 tables to a given MQ/Series Server queue manager, you could send all 4 message types to one queue. Upon pulling a message from that one queue, perhaps your first task would be to determine which of 4 tables that message is related to. Or, you could choose to send each of the 4 tables change data capture events to its own

dedicated queue. There is little difference. Queues are logical terms, existing as definitions within the MQ/Series landscape.

Previously we ran a direct command line statements to get things up and running. Now we will enter an interpretive shell specific to MQ/Series Server. The shell (command) name is runmqsc, and is given a single parameter of the queue manager name. To exit this shell, enter the command, end. Example as shown,

```
# Enter the MQ/Series server command shell, for the queue manager
# named RHHOST_QM, then immediately exit.
runmqsc  RHHOST_QM
>       (Normally you would run MQ specific commands here.)
>end
```

The command to define a queue, view details about the queue, and see the number of messages in the queue may be done within the runmqsc command shell. Example as shown,

```
runmqsc  RHHOST_QM
> def  ql  (  RHHOST_Q  )
> dis  ql  (  *  )
> dis  ql  (  RHHOST_Q  )
> def  channel  (  CHANNEL1  ) chltype  (  SVRCONN  ) TRPTYPE  (  TCP  )
>dis  channel  (  *  )
>end
```

Note: There is a graphical user interface for all of this. We just chose not to use it.

A code review of the above follows;

- The runmqsc command calls to enter the MQ/Series Server interpretive shell for the named queue manager.
- The def ql (RHHOST_Q) command calls to define a new queue named RHHOST_Q. This queue is wholly contained within this queue manager.
- The dis ql (*) command displays a listing of all queues, with short detail. This command is non-destructive, no changes result.
- The dis ql (RHHOST_Q) command displays a detailed listing for the named queue. This command is non-destructive, no changes result.

In the output of this command, you specifically want to look at the value for CURDEPTH; this specifies the number of messages in this queue.

– The `def channel ( CHANNEL1 ) chltype ( SVRCONN ) TRPTYPE ( TCP )` command calls to define a new channel named CHANNEL1, of the given channel type and transport type. We have not yet defined the terms channel, channel type, etcetera.

  Most commonly, MQ channels exist for inter system communication, MQ/Series Server to MQ/Series Server. Those channels have types that specify whether they send or receive inter MQ Server communication.

  The channel we created above supports communication from a client program to MQ/Series Server. In our example that follows, we will use this channel name to talk from DataStage to MQ/Series Server.

– The `dis channel ( * )` command displays a listing of all channels.

– The `end` command calls to exit this MQ/Series interpretive shell.

Note: The shortest version possible is; you can send and receive messages to just queues. Information Server is going to expect the MQ/Series entity (object definition) beyond a queue, named channel.

Placing a message on the queue

Here we document two methods to write to and then read from a given queue. The second method does its work in a slightly different manner, specifically using the named MQ/Series Server channel. Since most applications will make use of a named channel, the second set of steps is most relevant to our task at hand.

The programs we are about to run come with a set of sample MQ/Series Server client application programs, and are located in the directory entitled,

```
/opt/mqm/samp/bin/
```

First set of instructions;

```
# Place a message on the queue.
amqsput RHHOST_Q RHHOST_QM < {Input File Name}

# Or,
amqsput RHHOST_Q RHHOST_QM
(Enter text until complete, a blank line terminates input.)
```



```
# Read the queue and leave message in place.
amqsbcg   RHHOST_Q   RHHOST_QM
# Read the queue and remove the message.
amqsget   RHHOST_Q   RHHOST_QM
```

Second set of instructions, these require that an environment variable be set as shown below;

```
# Where CHANNEL1 is a channel name created above, rhhost.grid is the
# operating system host name we are running MQ/Series Server on, and port
# 1414 is that port number which we started the listener on above.
# (These and all values are case sensitive.)
export MQSERVER='CHANNEL1/TCP/rhhost.grid(1414)'

# Put a message on the queue via the channel identifier.
amqsputc   RHHOST_Q   RHHOST_QM   <   {Input File Name}
amqsputc   RHHOST_Q   RHHOST_QM       (enter data until complete)

# Get a message from the queue and remove the message.
amqsgetc   RHHOST_Q   RHHOST_QM
```

MQ/Series Server permissions

All of the above was done via the mqm user id, and like any piece of server software, MQ/Series allows for complete operator authentication. Given an Information Server end user name of, iisadmin, execute the following as mqm to grant full authority to iisadmin;

```
setmqaut   -m   RHHOST_QM   -t   qmgr   -p   iisadmin   +all
setmqaut   -m   RHHOST_QM   -t   q   -n   RHHOST_Q   -p   iisadmin   +all
```

Go back and re-run the message send and receive commands above as user iisadmin. Remember to set the MQSERVER environment variable.

Note: Obviously one can set lesser permissions to user iisadmin. That discussion was beyond the scope of this document.

28.2 Complete the following examples

In this section of this document, we create Information Server DataStage component Jobs to send and receive MQ/Series Server messages. Successful operation of these Jobs is dependent on an available MQ/Series Server.

1. Create a sample input data file.

   We created an ASCII text file with 1 column, 1 record, and given formatting.

2. Log on, perform basic tasks.
   - Log on to the Information Server DataStage component Designer program.
   - Create 2 new DataStage Parallel Jobs. Save the first Job with the name, PJ01_WriteToMQ, and the second with the name, PJ02_ReadFromMQ.

3. Continue work on Job, PJ01_WriteToMQ.
   - Drag 2 Stages from the Palette view to the Parallel Canvas, example as shown in Figure 28-3.
   - Per our design, you need a Sequential File stage to read from, and a WebSphere MQ Connector to write to.
   - Position and rename the Stages and Links as displayed in Figure 28-3.
   - Be certain you can read from the input file via that Stage's View Data button, before proceeding.

Note: The WebSphere MQ Connector is the preferred MQ input/output Stage. The other Stage entitled just, MQ Connector, is a legacy (old) Stage from past releases of the product.

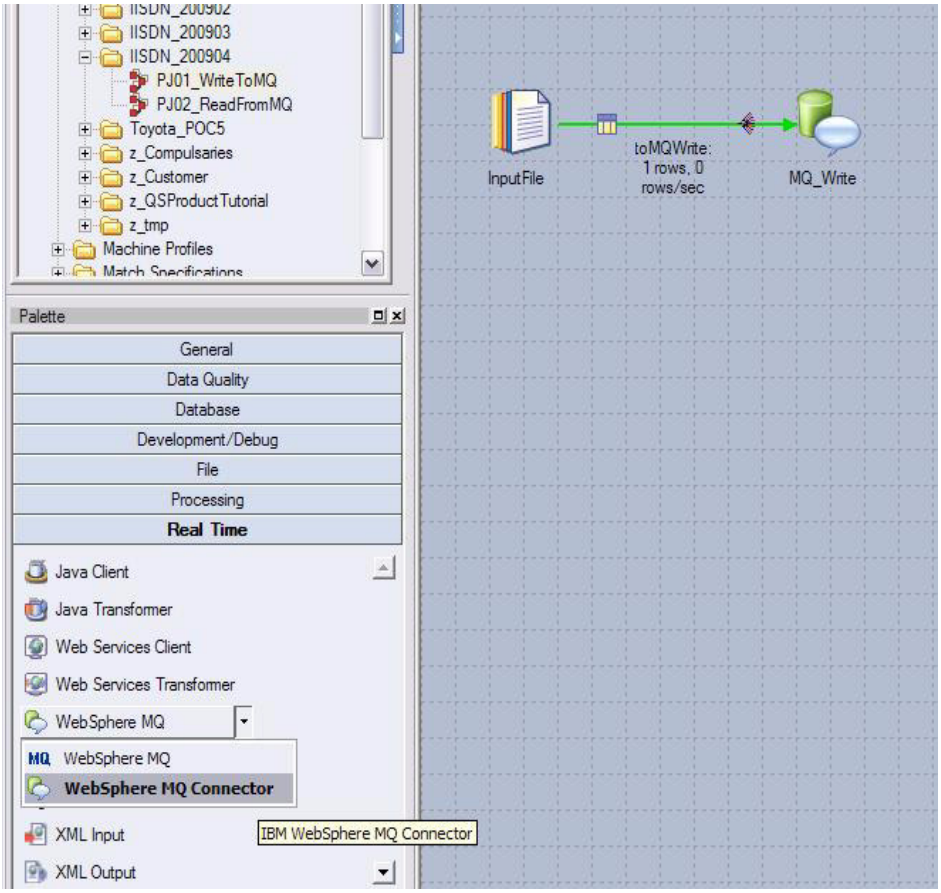

Figure 28-3 PJ01_WriteToMQ, the Job we create first.

4. Configure the WebSphere MQ Connector as displayed in Figure 28-4.

   Minimally you must set the following;

   – Queue manager (name)
   – Username
   – Password
   – Queue name

   Also, we set the Transaction -> Record count, to zero. The write to MQ will complete upon receipt of the last record from the input data source.

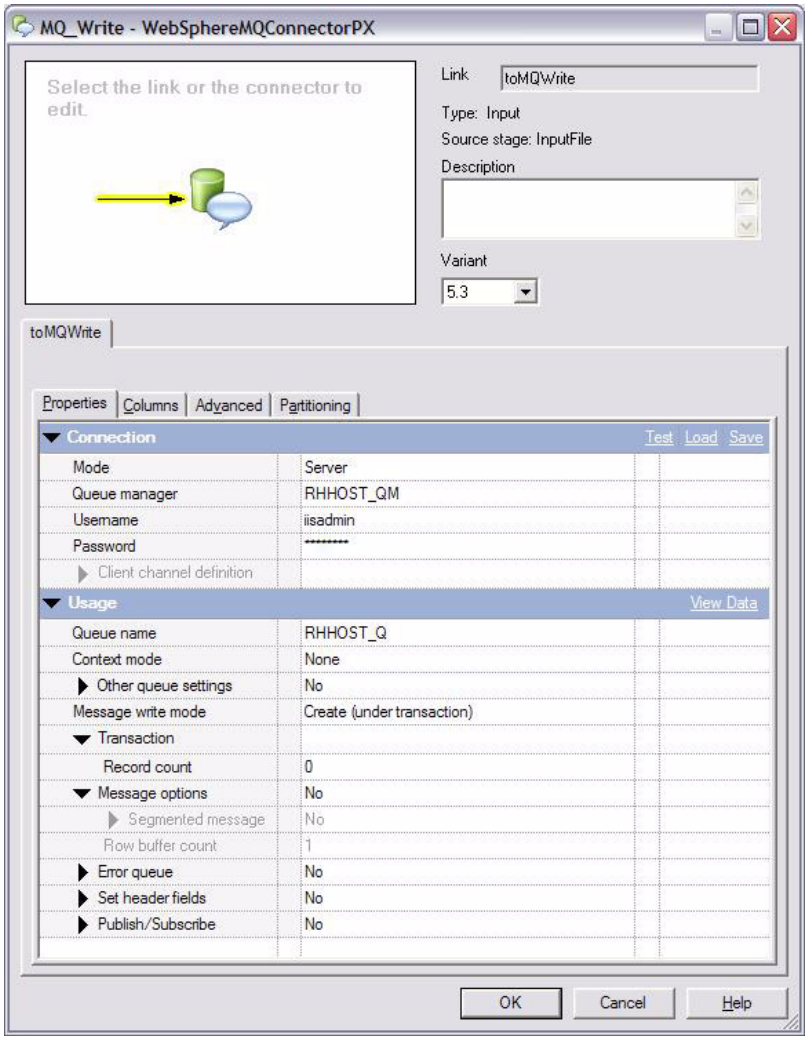

Figure 28-4 WebSphere MQ Connector Stage.

5. Save, compile and test.

   A successful test appears in Figure 28-3, writing one message to the queue.

6. Continue work on Job, PJ02_ReadFromMQ.

   Example as shown in Figure 28-5.

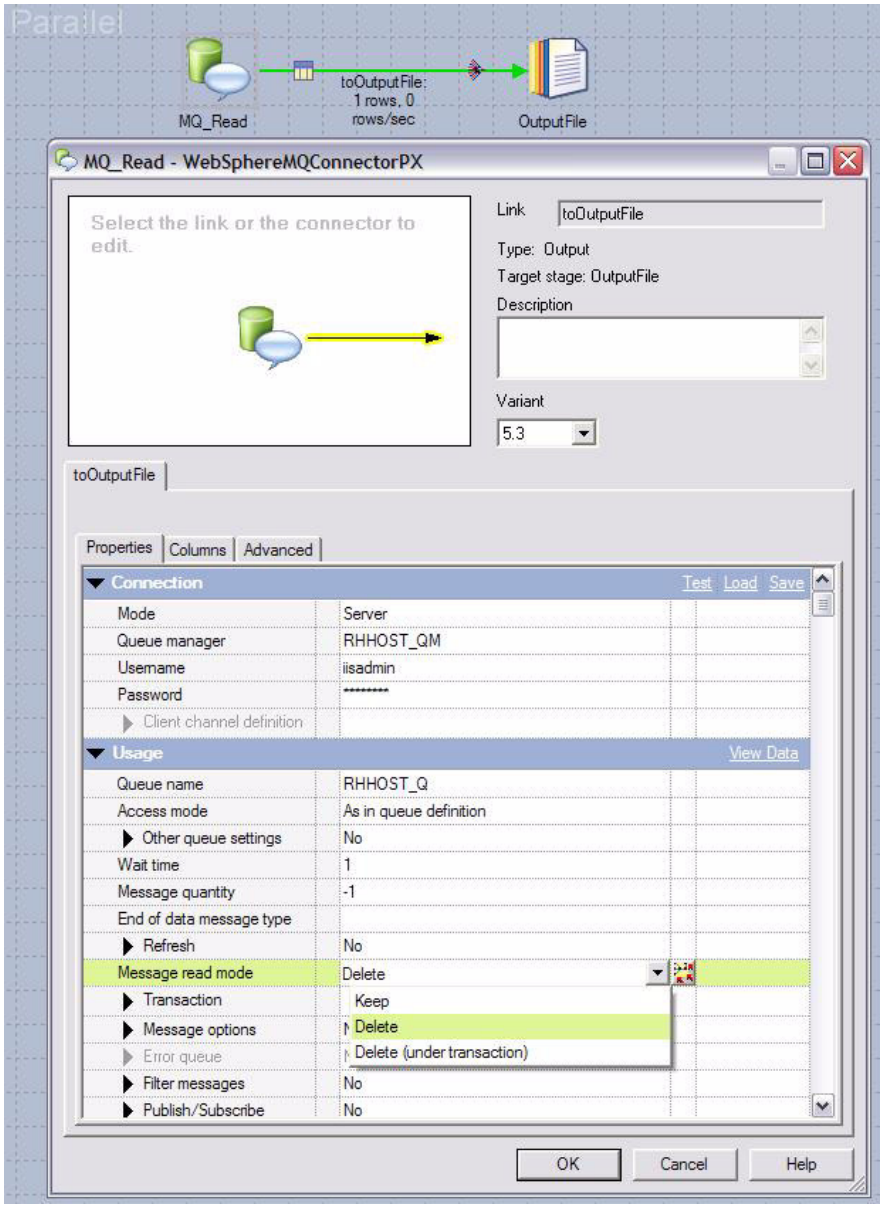

Figure 28-5 PJ02_ReadFromMQ, complete Job as displayed.

7. Configure the WebSphere MQ Connector.
   - In Figure 28-5, we set the Usage -> Message quantity, to -1 (unlimited).
   - And we set the Usage -> Wait time to 1 (second).

- Under Usage -> Message read mode, you can chose to remove the message from the queue when reading (Delete), or leave it in place (Keep).

8. Save, compile and test.

   A successful test appears in Figure 28-5, reading one message to the queue.

Further topics, not previously discussed-

Channels

While we set the MQ/Server queue manager name and queue name above, no where did we set the channel name. The channel name is set via environment variable; remember the MQSERVER environment variable above?

To set any environment variable inside the Information Server DataStage component at the Job level, complete the following;

- With the given Job open for edit, select Edit -> Job Properties -> Parameters TAB -> Add Environment Variables button, from the menu bar.
- Add a variable named MQSERVER, as formatted and set above.

Transaction control, commit points

While the Information Server DataStage component WebSphere MQ Connector supports this capability and parameter setting, that topic goes just a bit too far for the scope of this document.

Formatting of the inbound or outbound MQ/Series Server message

The WebSphere MQ Connector Stage expects to send or receive one column.

Normally the input column is formatted via XML, in which case we use the XML Input Stage; see the June/2007 edition of this IISDN document for details on using that Stage. Or, the input column is fixed length or delimited; see the May/2008 edition of this IISDN document for details on using the Column Importer Stage.

For formatted output to MQ, use the XML Output Stage or the Column Exporter Stage, detailed in the same editions of IISDN listed just above.

28.3 In this document, we reviewed or created:

We introduced the architectural concept of database gateways; federated databases, and federated servers. Federated servers can be used to increase application performance, security, data quality, and more.

And we demonstrated use of the Information Server DataStage component WebSphere MQ Connector; including information towards installing, configuring, and testing MQ/Series Server.

Persons who help this month.

Emily Drew, and Newton Drew.

Additional resources:

Have you met Newton? What more do you need- I still have that four foot long, 35 pound "stick" in my yard.

Legal statements:

Special attributions:

The listed trademarks of the following companies require marking and attribution:

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Microsoft trademark guidelines

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Intel trademark information

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

ITIL is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency, which is now part of the Office of Government Commerce.

Other company, product, or service names may be trademarks or service marks of others.

