

Configure MQ connections between IBM Integration Bus v10 and IBM MQ v8

Venu Movva

February 24, 2016

In IBM® Integration Bus v10, the Integration node can now run alone without any queue managers running. In this tutorial, you learn different ways to configure MQ connections between IBM Integration Bus v10 and IBM MQ v8.

Introduction

Before IBM Integration Bus v10, the Integration node depended on the queue manager to connect to IBM MQ. If the Integration node had to process a message to the queue manager, the node had to run on the queue manager. Starting in IBM Integration Bus v10, the dependency on IBM MQ has been removed. The Integration node can now run alone without any queue managers running. In this tutorial, you learn the different ways to configure MQ connections between IBM Integration Bus v10 and IBM MQ v8.

To connect an Integration node to the MQ queue manager, you can choose from the following options at the MQInput and MQOutput nodes:

- Local queue manager
- MQ client connection properties
- Client Channel Definition Table (CCDT)

By choosing one of these options, the Integration node can connect to the queue manager and propagate the messages.

Set up the system

To configure the components and objects that are described in this tutorial, you must have the following skills and system requirements.

Required skills

Before you begin this tutorial, you must have skills in the following areas:

- Implementation knowledge of MQ client/server architecture
- Intermediate knowledge of IBM MQ administration
- Intermediate knowledge of IBM Integration Bus administration
- IBM Integration Bus development skills

System requirements

Table 1 outlines the infrastructure setup that is required for the three options. In this table, Y indicates that the product is installed on the server, and NA indicates that the product is not installed or does not apply to the server.

You are *not required* to configure all prerequisites as listed in Table 1. Therefore, configure only the prerequisites for the option that you intend to implement in your environment.

Table 1. Server environment overview for each option

| Options | IBM Integration Bus v10 on Microsoft® Windows® | IBM Integration Bus v8 on Windows | IBM Integration Bus v8 on Windows | MQ v8 on Linux® |
|----------------------|--|-----------------------------------|-----------------------------------|-----------------|
| Local queue manager | Y | Y | NA | NA |
| MQ client connection | Y | NA | Y | Y |
| CCDT | Y | NA | Y | Y |

Integration node creation: You do not need to create the Integration node again when you use the `-q` flag or `mqsichangebroker` command.

Create the MQ objects for connectivity on a Linux server

If you are running MQ on a Linux server, create the required MQ objects for IBM Integration Bus connectivity:

1. Check the version of MQ:

```
-bash-4.1$ dspmqver
```

Listing 1 shows an example of the output that is displayed for the version of MQ.

Listing 1. MQ version details

```
Name: WebSphere MQ
Version: 8.0.0.2
Level: p800-002-150519.TRIAL
BuildType: IKAP - (Production)
Platform: WebSphere MQ for Linux (x86-64 platform)
Mode: 64-bit
0/S: Linux 2.6.32-71.el6.x86_64
InstName: Installation1
InstDesc:
Primary: Yes
InstPath: /opt/mqm
DataPath: /var/mqm
MaxCmdLevel: 801
```

2. Create the `IIB10_QM1` and `IIB10_QM2` queue managers:

```
crtmqm IIB10_QM1
crtmqm IIB10_QM2
```

3. Start the `IIB10_QM1` and `IIB10_QM2` queue managers:

```
strmqm IIB10_QM1
strmqm IIB10_QM2
```

4. Check the status of the queue managers:

```
-bash-4.1$ dspmq
```

Creation of the queue managers is successful, and their status is shown as *Running*:

```
QMNAME(IIB10_QM1) STATUS(Running)
QMNAME(IIB10_QM2) STATUS(Running)
```

5. Create the required objects under the IIB10_QM1 queue manager:

```
-bash-4.1$ runmqsc IIB10_QM1
```

a. Define the local queue:

```
DEFINE QL(INPUT.QL)
```

You see the following output:

```
AMQ8006: WebSphere MQ queue created.
```

b. Define the channel:

```
DEFINE CHANNEL(IIBV10.SVRCONN) CHLTYPE(SVRCONN)
```

You see the following output:

```
AMQ8014: WebSphere MQ channel created.
```

c. Define the listeners:

```
DEFINE LISTENER(QM1.LIS) TRPTYPE(TCP) PORT(1025) CONTROL(QMGR)
```

You see the following output:

```
AMQ8626: WebSphere MQ listener created.
```

d. Start the listener:

```
START LISTENER(QM1.LIS)
```

You see the following output:

```
AMQ8021: Request to start WebSphere MQ listener accepted.
```

6. Create the client-connection channel for IIB10_QM1 on the IIB10_QM1 queue manager:

```
DEFINE CHANNEL(IIBV10.SVRCONN) CHLTYPE(CLNTCONN) CONNAME('192.168.112.131(1025)') QMNAME(IIB10_QM1)
```

You see the following output:

```
AMQ8014: WebSphere MQ channel created.
```

7. Create the IIB10_QM2 CLNTCONN channels as part of the CCDT configuration for IIB10_QM2 on the IIB10_QM1 queue manager:

```
DEFINE CHANNEL(IIBV10.QM2.SVRCONN) CHLTYPE(CLNTCONN) CONNAME('192.168.112.131(1026)') QMNAME(IIB10_QM2)
```

You see the following output:

```
AMQ8014: WebSphere MQ channel created.
```

8. Provide the authorizations for the IIB10_QM1 queue manager for the Integration node to connect to the queue manager:

```
setmqaut -m IIB10_QM1 -t qmgr -p system +connect
setmqaut -m IIB10_QM1 -n INPUT.QL -t q -p system +inq +get
```

9. Ensure that all authorizations reflect the queue manager that is being used:

```
refresh security
```

10. Create the required objects under the IIB10_QM2 queue manager:

```
bash-4.1$ runmqsc IIB10_QM2
```

- a. Define the local queue:

```
DEFINE QL(OUTPUT.QL)
```

You see the following output:

```
AMQ8006: WebSphere MQ queue created.
```

- b. Define the channel:

```
DEFINE CHANNEL(IIBV10.QM2.SVRCONN) CHLTYPE(SVRCONN)
```

You see the following output:

```
AMQ8014: WebSphere MQ channel created.
```

- c. Define the listeners:

```
DEFINE LISTENER(QM2.LIS) TRPTYPE(TCP) PORT(1026) CONTROL(QMGR)
```

You see the following output:

```
AMQ8626: WebSphere MQ listener created.
```

11. Provide the authorizations for the IIB10_QM2 queue manager for the Integration node to connect to the queue manager:

```
setmqaut -m IIB10_QM2 -t qmgr -p system +inq +connect +setall
setmqaut -m IIB10_QM2 -n OUTPUT.QL -t q -p system +put +setall
```

12. Ensure that all the authorizations reflect the queue manager that is being used:

```
refresh security
```

Create the MQ objects for connectivity on a Windows server

If you are running MQ on a Windows server, create the required MQ objects for IBM Integration Bus connectivity:

1. Create the LOCAL_QM1 and LOCAL_QM2 queue managers:

```
crtmqm LOCAL_QM1
crtmqm LOCAL_QM2
```

2. Start the LOCAL_QM1 and LOCAL_QM2 queue managers:

```
strmqm LOCAL_QM1
strmqm LOCAL_QM2
```

3. Check the status of the queue managers:

```
-bash-4.1$ dspmq
```

Creation of the queue managers is successful, and their status is shown as *Running*:

```
QMNAME(LOCAL_QM1) STATUS(Running)
QMNAME(LOCAL_QM2) STATUS(Running)
```

4. Create the required objects under the LOCAL_QM1 queue manager:

```
runmqsc LOCAL_QM1
```

5. Define the local queue:

```
DEFINE QL(INPUT.QL)
```

You see the following output:

```
AMQ8006: WebSphere MQ queue created.
```

6. Create the required objects under the LOCAL_QM2 queue manager:

```
runmqsc LOCAL_QM2
```

7. Define the local queue:

```
DEFINE QL(OUTPUT.QL)
```

You see the following output:

```
AMQ8006: WebSphere MQ queue created.
```

Configure IBM Integration Bus on the Windows server

To configure IBM Integration Bus on the Windows server:

1. Create the IIBV10 Integration node, where IIBV10 is the broker name and the Integration node does not have any queue manager dependency:

```
mqsicreatebroker IIBV10
```

After you create the integration node, you see the following output:

```
BIP8071I: Successful command completion.
```

2. Check the status of the Integration node:

```
mqsilist
```

The following status is displayed.

```
BIP1326I: Integration node 'IIBV10' is stopped.
```

```
BIP8071I: Successful command completion.
```

3. Start the Integration node:

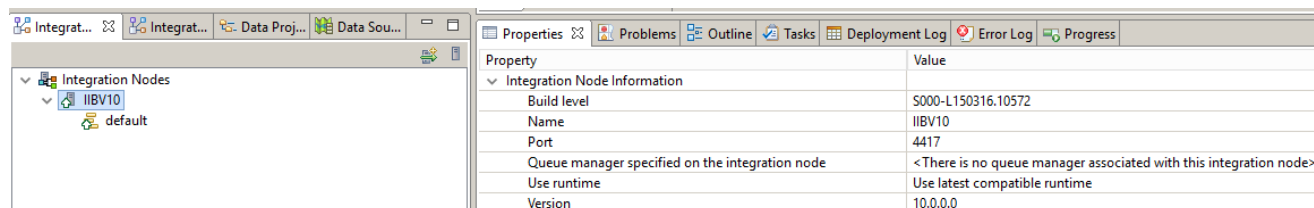
```
mqsistart IIBV10
```

You see the following output:

```
BIP8096I: Successful command initiation.
```

4. Check the Integration node properties by using the IBM Integration Toolkit (Figure 1). The queue manager that is specified on the Integration node property indicates that no queue manager is associated with this Integration node.

Figure 1. Integration node validation with the IBM Integration toolkit



Message flow

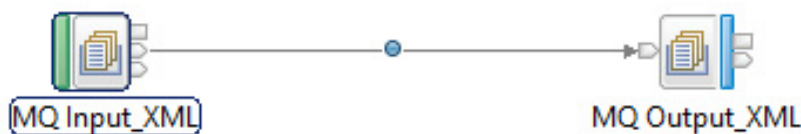
A sample message flow was developed by using the IBM Integration toolkit. You can use it for all three methods that are described in this tutorial. This message flow illustrates the connection of the following queue managers:

- IIB10_QM1 and IIB10_QM2 that are running on Linux
- LOCAL_QM1 and LOCAL_QM2 that are running on Windows

Figure 2 illustrates a flow that accepts messages in the XML format and transfers them as follows:

- From IIB10_QM1 to IIB10_QM2 (for the MQ client connection properties and CCDT methods) running on Linux
- From LOCAL_QM1 to LOCAL_QM2 (for the Local queue manager method) running on Windows

Figure 2. Message flow in the XML format



Properties configuration

For all three scenarios, you configure the properties by using MQ Enterprise Transport.

Because you are using the message flow to transform the XML messages from one queue manager to another queue manager, a distribution setup is not required at the MQ level. When you develop a message flow, the message processing by the flow depends on the properties of the nodes. For example, by setting the properties that define the INPUT and OUTPUT queue names for MQ, you determine where the message flow receives the message from and where it delivers the message.

Figure 3 shows the routing of a message from the LOCAL_QM1 queue manager to the LOCAL_QM2 queue manager. The client environment that is shown has two queue managers (LOCAL_QM1 and LOCAL_QM2) and IBM Integration Bus (IIBV10). They are installed on the same Windows server. The source application puts a message on the INPUT.QL queue under the LOCAL_QM1 queue manager. Then, the message flow picks up the message and sends it to the OUTPUT.QL queue under the LOCAL_QM2 queue manager for the target application.

Figure 3. Routing a message from LOCAL_QM1 to LOCAL_QM2

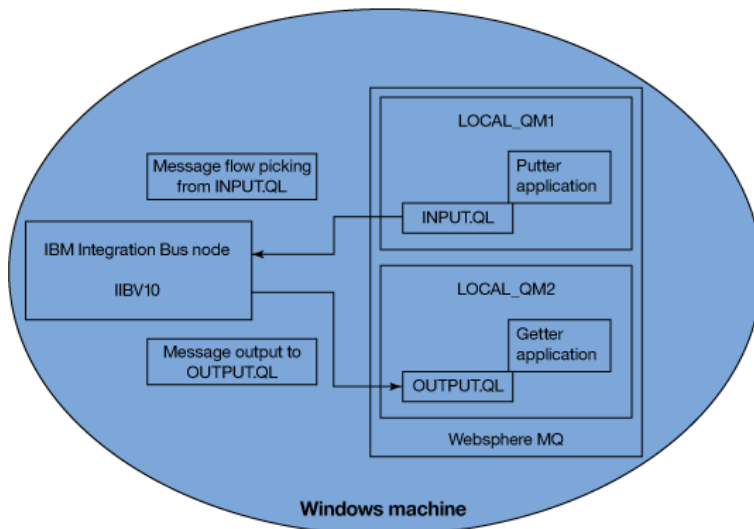
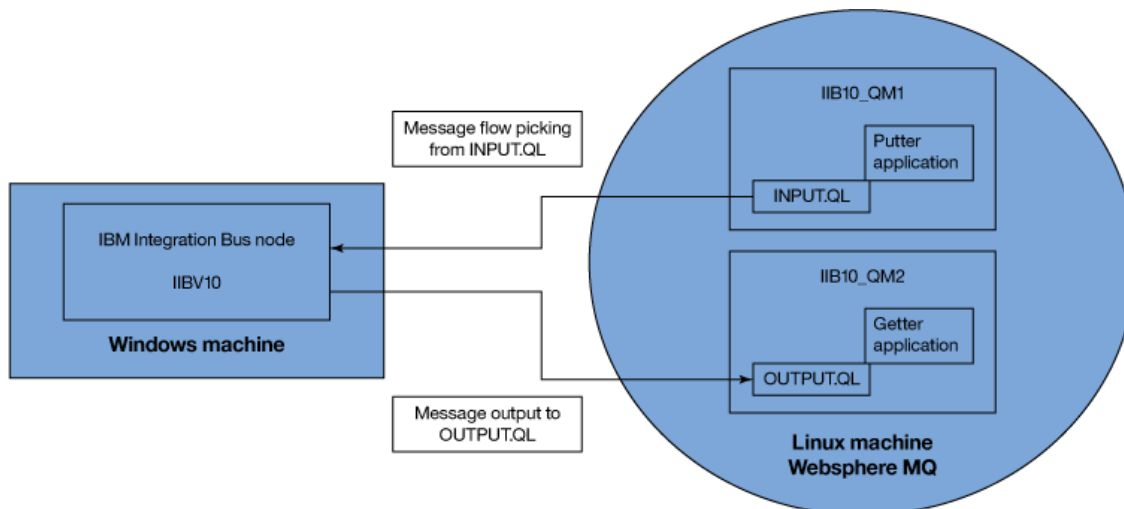


Figure 4 shows the routing of a message from `IIBV10_QM1` to `IIBV10_QM2`. The client environment has two queue managers (`IIB10_QM1` and `IIB10_QM2`) on a Linux server and IBM Integration Bus (`IIBV10`) on a Windows server. The putter application puts a message on the `INPUT.QL` queue under `IIB10_QM1`. Then, the message flow picks the message and sends it to the `OUTPUT.QL` queue under `IIB10_QM2` for the getter application.

Figure 4. Routing a message from IIBV10_QM1 to IIBV10_QM2



Scenario 1: Local queue manager

In this scenario, IBM Integration Bus v10 and IBM MQ v8 are both installed on the same Windows server, and both the components are connecting in Binding mode. In Binding mode, IBM

Integration Bus and MQ connect to each other on the same server. Also, a local connection is made to a local queue manager that uses server bindings. A client connection is made to remote queue managers. You can use local connections, client connections, or a combination of local and client connections to your queue managers.

If you select the local queue manager option, you must specify a queue manager name. The message flow makes a server connection to the queue manager (by using server bindings). The Integration node and queue manager must be running on the same server.

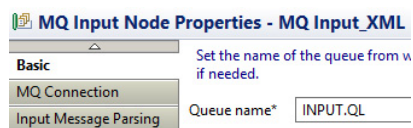
Configure the MQInput node properties

The MQInput node receives messages from applications and connects to the Integration node by using the IBM MQ Enterprise Transport.

To configure the MQInput node properties:

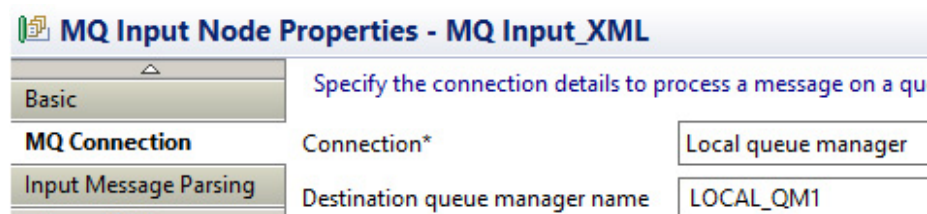
1. Update the `INPUT.QL` queue name:
 - a. Navigate to the MQ Input Node Properties window (Figure 5).
 - b. On the **Basic** tab, for **Queue name**, enter `INPUT.QL`.

Figure 5. Queue name property



2. Update the connection details for the `LOCAL_QM1` queue manager:
 - a. Go to the MQ Input Node Properties window (Figure 6).
 - b. On the **MQ Connection** tab, for **Connection**, enter `Local queue manager`. For **Destination queue manager name**, enter `LOCAL_QM1`.

Figure 6. Connection details for the LOCAL_QM1 queue manager



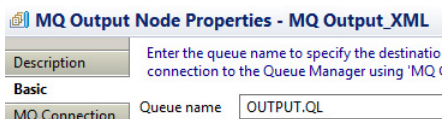
Configure the MQOutput node properties

The MQOutput node delivers an output message from a message flow to an MQ queue. The node uses `MQPUT` to put the message to the destination queue or queues that you specify.

To configure the MQOutput node properties:

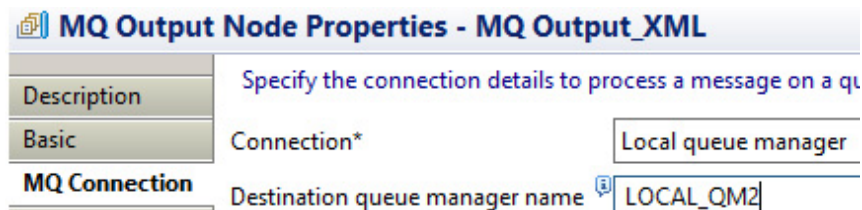
1. Update the OUTPUT.QL queue name:
 - a. Go to the MQ Output Node Properties window (Figure 7).
 - b. On the **Basic** tab, for **Queue name**, enter OUTPUT.QL.

Figure 7. Queue name property



2. Update the connection details for the LOCAL_QM2 queue manager:
 - a. Go to the MQ Output Node Properties window (Figure 8).
 - b. On the **MQ Connection** tab, for **Connection**, enter Local queue manager. For **Destination queue manager name**, enter LOCAL_QM2.

Figure 8. Connection details for the LOCAL_QM2 queue manager



Confirm the data flow engine connection

To confirm the data flow engine connection with the queue, display the queue status:

```
runmqsc LOCAL_QM1
dis qs(INPUT.QL) type(handle)
```

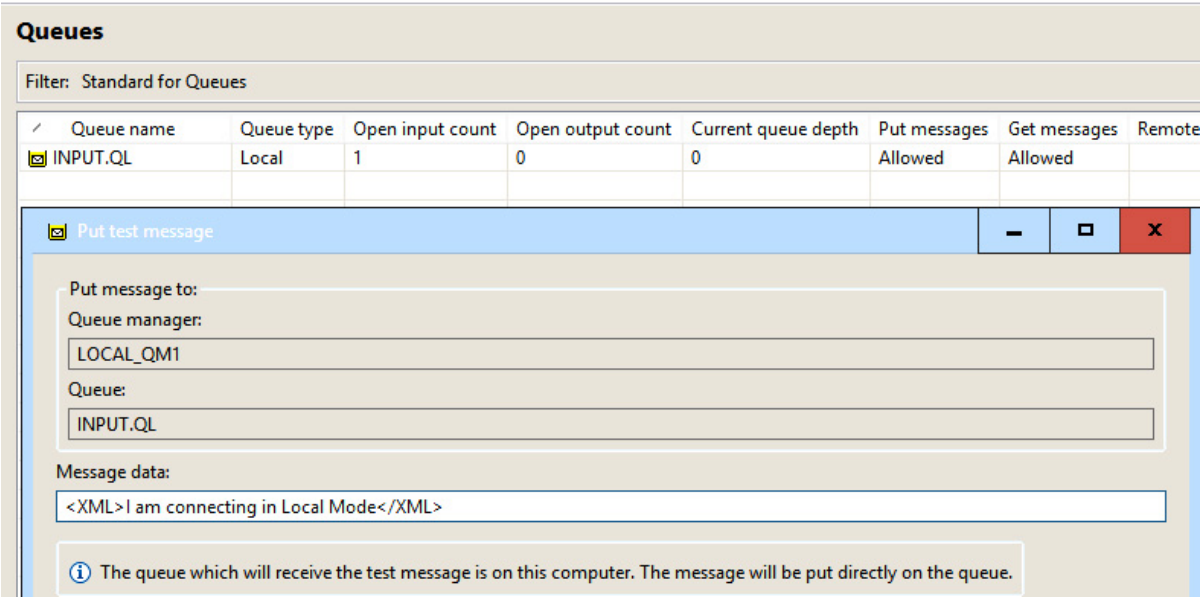
The output in Listing 2 shows APPLTAG(erver\bin\DataFlowEngine.exe), which indicates that the queue is listening to the data flow engine.

Listing 2. Queue status for the data flow engine connection

```
AMQ8450: Display queue status details.
QUEUE(INPUT.QL)  TYPE(HANDLE)
APPLDESC( )
APPLTAG(erver\bin\DataFlowEngine.exe)
APPLTYPE(USER)  BROWSE(NO)
CHANNEL( )      CONNAME( )
ASTATE(ACTIVE)  HSTATE(ACTIVE)
INPUT(SHARED)   INQUIRE(YES)
OUTPUT(NO)      PID(5260)
QMURID(0.12291) SET(NO)
TID(*)
URID(XA_FORMATID[] XA_GTRIDH[] XA_BQUAL[])
URTYPE(QMGR)    USERID(SYSTEM@NT AUTHORITY)
```


As shown in Figure 9, the Current queue depth for the `INPUT.QL` queue name is 0. It also shows the message data that is displayed. In MQ Explorer, use the `PUT` command to place the message in the `INPUT.QL` queue on the `LOCAL_QM1` queue manager.

Figure 9. Current queue depth and message data for INPUT.QL



As shown in Figure 10, the Current queue depth for the `OUTPUT.QL` queue is now 1. This value confirms that the message flow picks the message from the `INPUT.QL` queue on the `LOCAL_QM1` queue manager and sends it to the `OUTPUT.QL` queue on the `LOCAL_QM2` queue manager.

Figure 10. Current queue depth for OUTPUT.QL

| Queues | | | | | | | | |
|---|------------|------------------|-------------------|---------------------|--------------|--------------|--------------|--------|
| Filter: Standard for Queues | | | | | | | | |
| Queue name | Queue type | Open input count | Open output count | Current queue depth | Put messages | Get messages | Remote queue | Remote |
|  OUTPUT.QL | Local | 0 | 1 | 1 | Allowed | Allowed | | |

By using the IBM Integration Toolkit, the Current queue depth of the `OUTPUT.QL` queue on the `LOCAL_QM2` queue manager changed from 0 to 1 (Figure 11). This change confirms that the message was sent. Also, the message data is for the same message that was sent from the `INPUT.QL` queue on the `LOCAL_QM1` queue manager.

Figure 11. Current queue depth and message data for OUTPUT.QL

[illegible]

Scenario 2: MQ client connection properties

If you choose to define the MQ client connection properties, you must also specify the following properties:

- Queue manager host name
- Listener port number
- Channel name
- Destination queue manager name at the MQInput nodes
- Destination queue manager name at the MQOutput nodes

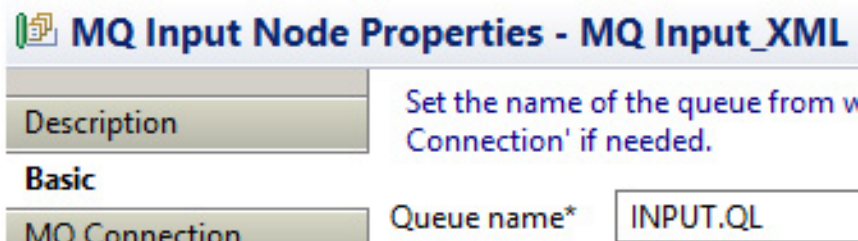
In this example, the Integration node is running on a Windows server, and the queue managers are running on a Linux server.

Configure the MQInput node properties

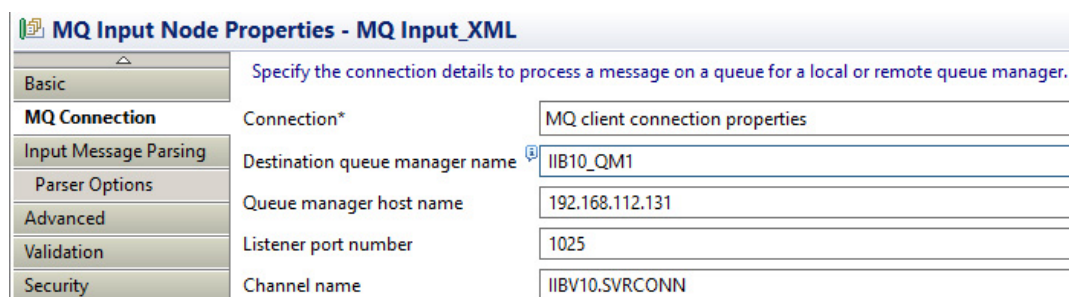
To configure the MQInput node properties:

1. Update the `INPUT.QL` queue name:
 - a. Go to the MQInput Node Properties window (Figure 12).
 - b. On the **Basic** tab, for Queue name, enter `INPUT.QL`.

Figure 12. Queue name property for the MQInput node



2. Update the connection details for the `IIB10_QM1` queue manager:
 - a. Go to the MQ Input Node Properties window.
 - b. On the **MQ Connection** tab (Figure 13):
 - i. For **Connection**, enter `MQ client Connection properties`.
 - ii. For **Destination queue manager name**, enter `IIB10_QM1`.
 - iii. For **Queue manager host name**, enter the IP address.
 - iv. For **Listener port number**, enter `1025`.
 - v. For **Channel name**, enter `IIBV10.SVRCONN`.

Figure 13. Connection details for IIB10_QM1


MQ Input Node Properties - MQ Input_XML

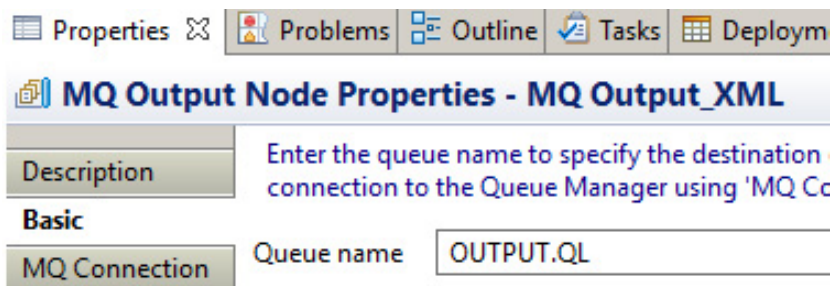
Specify the connection details to process a message on a queue for a local or remote queue manager.

| | | |
|-----------------------|--------------------------------|---------------------------------|
| Basic | | |
| MQ Connection | Connection* | MQ client connection properties |
| Input Message Parsing | Destination queue manager name | IIB10_QM1 |
| Parser Options | Queue manager host name | 192.168.112.131 |
| Advanced | Listener port number | 1025 |
| Validation | Channel name | IIBV10.SVRCONN |
| Security | | |

Configure the MQOutput node properties

To configure the MQOutput node properties:

1. Update the `OUTPUT.QL` queue name:
 - a. Go to the MQ Output Node Properties window (Figure 14).
 - b. On the **Basic** tab, for **Queue name**, enter `OUTPUT.QL`.

Figure 14. Queue name of the MQOutput node


MQ Output Node Properties - MQ Output_XML

Enter the queue name to specify the destination connection to the Queue Manager using 'MQ Cc

| | | |
|---------------|------------|-----------|
| Description | | |
| Basic | | |
| MQ Connection | Queue name | OUTPUT.QL |

2. Update the connection details for the `IIB10_QM2` queue manager:
 - a. Go to the MQ Output Node Properties window (Figure 15).
 - b. On the **MQ Connection** tab:
 - i. For **Connection**, enter `MQ client connection properties`.
 - ii. For **Destination queue manager name**, enter `IIB10_QM2`.

- iii. For **Queue Manager Host Name**, enter the IP address.
- iv. For **Listener Port Number**, enter `1026`.
- v. For **Channel Name**, enter `IIBV10.QM2.SVRCONN`.

Figure 15. Connection details for IIB10_MQ2

MQ Output Node Properties - MQ Output_XML

Description Specify the connection details to process a message on a queue for a local or remote queue manager.

| | | |
|----------------------|--------------------------------|---------------------------------|
| Basic | Connection* | MQ client connection properties |
| MQ Connection | Destination queue manager name | IIB10_QM2 |
| Advanced | Queue manager host name | 192.168.112.131 |
| Request | Listener port number | 1026 |
| Validation | Channel name | IIBV10.QM2.SVRCONN |
| Policy | | |

Test the MQ client connection

Until you test the message flow, the initial current depth (`CURDEPTH`) of the `INPUT.QL` and `OUTPUT.QL` queues is 0, which means that no messages are in the queue. To test the client connection:

1. Start the MQSC commands for the `IIB10_QM1` queue manager:

```
-bash-4.1$ runmqsc IIB10_QM1
```

2. Display the current depth of the `INPUT.QL` queue:

```
dis ql(INPUT.QL) CURDEPTH
```

The `CURDEPTH` is 0 as shown in the following input queue details:

```
AMQ40409: Display Queue details.
QUEUE(INPUT.QL)  TYPE(QLLOCAL)
CURDEPTH(0)
```

3. Start the MQSC commands for the `IIB10_QM2` queue manager:

```
-bash-4.1$ runmqsc IIB10_QM2
```

4. Display the current depth of the `OUTPUT.QL` queue:

```
dis ql(OUTPUT.QL) CURDEPTH
```

The `CURDEPTH` is 0 as shown in the following output queue details:

```
AMQ8409: Display Queue details.
QUEUE(OUTPUT.QL)  TYPE(QLLOCAL)
CURDEPTH(0)
```

5. Put a sample message on the `INPUT.QL` queue:

```
-bash-4.1$ ./amqsput INPUT.QL IIB10_QM1
```

You see the following output:

```
Sample AMQSPUT0 start
target queue is INPUT.QL
<XML>I am using IIBV10 and MQV8.0</XML>

Sample AMQSPUT0 end
```

6. Start the MQSC commands for the IIB10_QM2 queue manager:

```
-bash-4.1$ runmqsc IIB10_QM2
```

You see the following output:

```
Starting MQSC for queue manager IIB10_QM2.
```

7. Display the current depth of the OUTPUT.QL queue:

```
dis ql(OUTPUT.QL) CURDEPTH
```

As shown in the following output, the `CURDEPTH` is 1, meaning that the message flow processed the message successfully to the output queue:

```
AMQ8409: Display Queue details.
QUEUE(OUTPUT.QL)  TYPE(QLLOCAL)
CURDEPTH(1)
END
```

8. Verify the flow connection to the IIB10_QM1 queue manager:

```
dis chs(IIBV10.SVRCONN) all
```

The data flow engine (execution group) is listening, and the server connection channel `IIBV10.SVRCONN` is running. The MQInput node is connected to the server connection channel, showing successful connectivity in this scenario (Listing 3).

Listing 3. Message flow connected to the IIB10_QM1 queue manager

```
AMQ8417: Display Channel Status details.
CHANNEL(IIBV10.SVRCONN)  CHLTYPE(SVRCONN)
BUFSRCVD(46)  BUFSSNT(5)
BYTSRCVD(4296)  BYTSSNT(2152)
CHSTADA(2015-10-05)  CHSTAI(12.04.28)
COMPHDR(NONE,NONE)  COMPMMSG(NONE,NONE)
COMPRATE(0,0)  COMPTIME(0,0)
CONNAME(192.168.112.1)  CURRENT
EXITTIME(0,0)  HBINT(300)
JOBNAME(00000F3600000121)
LOCLADDR(:::ffff:192.168.112.131(1025))
LSTMSGDA(2015-10-05)  LSTMSGTI(12.04.28)
MCASTAI(RUNNING)  MCAUSER(system)
MONCHL(OFF)  MSGS(3)
RAPPLTAG(server\bin\DataFlowEngine.exe)
SECPROT(NONE)  SSLCERTI()
SSLKEYDA( )  SSLKEYTI()
SSLPEER( )  SSLRKEYS(0)
STAIUS(RUNNING)  STOPREQ(NO)
SUBSTAI(RECEIVE)  CURSHCNV(1)
MAXSHCNV(10)  RVERSION(08000002)
RPRODUCT(MQCC)
```

9. Verify the flow connection to the IIB10_QM2 queue manager:

```
dis chs(IIBV10.QM2.SVRCONN) all
```

As shown in Listing 4, the data flow engine (execution group) is listening, and the server connection channel `IIBV10.QM2.SVRCONN` is running. These results demonstrate that the MQOutput node is connected to the server-connection channel, indicating successful connectivity.

Listing 4. Message flow connected to the IIB10_QM2 queue manager

```
AMQ8417: Display Channel Status details.
CHANNEL(IIBV10.QM2.SVRCONN) CHLTYPE(SVRCONN)
BUFSRCVD(10)      BUFSSENT(9)
BYTSRCVD(2884)    BYTSSENT(2844)
CHSTADA(2015-10-05) CHSTATI(12.05.55)
COMPHDR(NONE,NONE) COMRMSG(NONE,NONE)
COMPRATE(0,0)     COMPTIME(0,0)
CONNAME(192.168.112.1) CURRENT
EXITTIME(0,0)     HBINT(300)
JOBNAME(0000IOZB00000000)
LOCLADDR(:::ffff:192.168.112.131(1026))
LSTMSGDA(2015-10-05) LSTMSGTI(12.05.55)
MCASTAT(RUNNING)   MCAUSER(system)
MONCHL(OFF)        MSGS(7)
RAPPLTAG(server\bin\DataFlowEngine.exe)
SECPROT(NONE)      SSLCERTI()
SSLKEYDA( )        SSLKEYTI()
SSLPEER( )         SSLRKEYS(0)
STATUS(RUNNING)    STOPREQ(NO)
SUBSTATE(RECEIVE)  CURSHCNV(1)
MAXSHCNV(10)       RVERSION(08000002)
RPRODUCT(MQCC)
```

10. Get the sample message from the `OUTPUT.QL` queue:

```
-bash-4.1$ ./amqsget OUTPUT.QL IIB10_QM2
```

You see the following output:

```
Sample AMQSGETO start
message <XML>I am using IIBV10 and MQV8.0</XML>
```

You have now placed a sample message on `INPUT.QL` by using the `AMQSPUT` program. The message flow picks the message from the `INPUT.QL` queue and sends it to the `OUTPUT.QL` queue on `IIB10_QM2`. You also see the `CURDEPTH` change to 1.

Scenario 3: Client Channel Definition Table

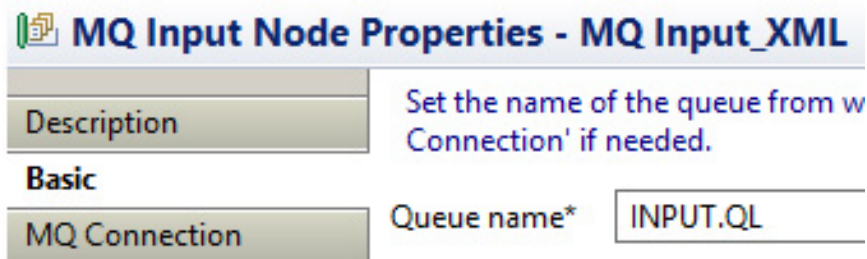
In this example, the Integration node is running on a Windows system, and the queue managers are running on a Linux system. If you choose the CCDT option, you must specify a queue manager name and configure the MQ CCDT file on a Windows system.

Configure the MQInput node properties

To configure the MQInput node properties:

1. Update the `INPUT.QL` queue name:
 - a. Go to the MQ Input Node Properties window (Figure 16).
 - b. On the **Basic** tab, for Queue name, enter `INPUT.QL`.

Figure 16. Queue name for the MQInput node



MQ Input Node Properties - MQ Input_XML

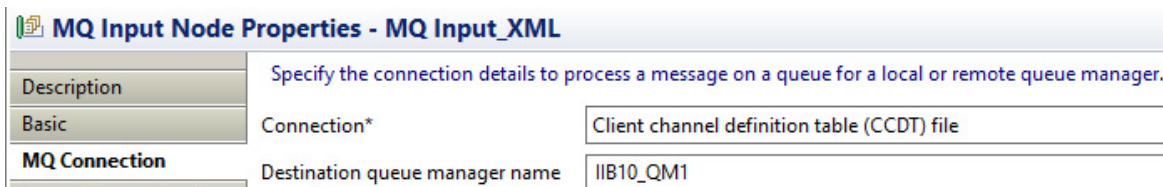
Description: Set the name of the queue from w Connection' if needed.

Basic

MQ Connection: Queue name* INPUT.QL

2. Update the MQ Client connection details for the IIB10_QM1 queue manager:
 - a. Go to the MQ Input Node Properties window (Figure 17).
 - b. On the **MQ Connection** tab, for **Connection**, enter client channel definition table (CCDT) file. For **Destination queue manager name**, enter IIB10_QM1.

Figure 17. Connection details for IIB10_QM1



MQ Input Node Properties - MQ Input_XML

Description: Specify the connection details to process a message on a queue for a local or remote queue manager.

Basic

Connection* Client channel definition table (CCDT) file

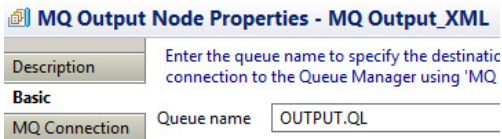
MQ Connection

Destination queue manager name IIB10_QM1

Configure the MQOutput node properties

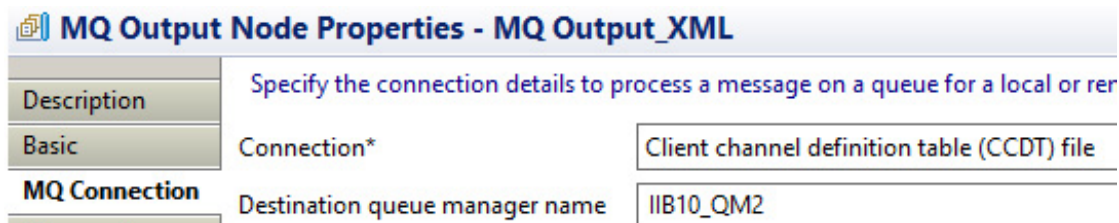
To configure the MQOutput node properties:

1. Update the OUTPUT.QL queue name:
 - a. Go to the MQ Output Node Properties window (Figure 18).
 - b. On the **Basic** tab, for **Queue name**, enter OUTPUT.QL.

Figure 18. Queue name for the MQOutput node


The screenshot shows the 'MQ Output Node Properties - MQ Output_XML' window. The 'Basic' tab is selected. The 'Queue name' field is set to 'OUTPUT.QL'. The 'Description' field contains the text: 'Enter the queue name to specify the destination connection to the Queue Manager using 'MQ'.

2. Update the connection details for the `IIB10_QM2` queue manager:
 - a. Go to the MQ Output Node Properties window (Figure 19).
 - b. On the **MQ Connection** tab, for **Connection**, enter `Client channel definition table (CCDT) file`. For **Destination queue manager name**, enter `IIB10_QM2`.

Figure 19. Connection details for the IIB10_QM2 queue manager


The screenshot shows the 'MQ Output Node Properties - MQ Output_XML' window. The 'MQ Connection' tab is selected. The 'Connection*' field is set to 'Client channel definition table (CCDT) file'. The 'Destination queue manager name' field is set to 'IIB10_QM2'. The 'Description' field contains the text: 'Specify the connection details to process a message on a queue for a local or remote queue manager.'

Register the CCDT file with the Integration node

Copy the `AMQCLCHL.TAB` CCDT file from the MQ Linux server to the Windows server by using the FTP, SFTP, or WINSXP utilities. Then, register the `AMQCLCHL.TAB` file with the Integration node:

```
mqsischangeproperties IIBV10 -o BrokerRegistry -n mqCCDT -v "C:\Users\MVP\Documents\AMQCLCHL.TAB"
```

Locate the CCDT file for the Integration node registry

Specify the location of the CCDT file for the Integration node registry:

```
C:\Program Files\IBM\IIB\10.0.0.0>mqsischangeproperties IIBV10 -o BrokerRegistry -n mqCCDT -v "C:\Users\MVP\Documents\AMQCLCHL.TAB"
```

You see the following message:

```
BIP8071I: Successful command completion
```

Check the CCDT file allocation

After you set the `mqCCDT` property, restart the Integration node for all the configurations to reflect the change. To check whether the `mqCCDT` property is registered to the `IIBV10` Integration node, you can use the following command:

```
mqsireportproperties IIBV10 -o BrokerRegistry -r BrokerRegistry
```

Listing 5 shows the command output.

Listing 5. CCDT file allocated to the Integration node registry

```
uuid='BrokerRegistry'
brokerKeystoreType='JKS'
brokerKeystoreFile=''
brokerKeystorePass='brokerKeystore::password'
brokerTruststoreType='JKS'
brokerTruststoreFile=''
brokerTruststorePass='brokerTruststore::password'
brokerCRLFileList=''
httpConnectorPortRange=''
httpsConnectorPortRange=''
brokerKerberosConfigFile=''
brokerKerberosKeytabFile=''
allowSSLv3=''
mqCCDT='C:\Users\MVP\Documents\AMQCLCHL.TAB'
modeExtensions=""
```

Confirm the data flow engine connection

To confirm the data flow engine connection with the queue:

1. Start the MQSC commands for the `IIB10_QM1` queue manager:

```
-bash-4.1$ runmqsc IIB10_QM1
```

2. Display the queue status of the `IIB10_QM1` queue manager:

```
DIS CHS(*) all
```

The data flow engine (execution group) is listening, and the server-connection channel `IIBV10.SVRCONN` is running in the `IIB10_QM1` queue manager (Listing 6). These results demonstrate that the MQInput node is connected to the server-connection channel, indicating successful connectivity.

Listing 6. Flow connection to the IIB10_QM1 queue manager

```
AMQ8417: Display Channel Status details.
CHANNEL(IIBV10.SVRCONN)  CHLTYPE(SVRCONN)
BUFSRCVD(156)           BUFSSENT(5)
BYTSRCVD(10184)         BYTSSENT(2152)
CHSTADA(2015-10-05)     CHSTAI(12.20.00)
COMPHDR(NONE,NONE)      COMRMSG(NONE,NONE)
COMPRATE(0,0)           COMPTIME(0,0)
CONNNAME(192.168.112.1) CURRENT
EXITTIME(0,0)           HBINT(300)
JOBNAME(00000FBG00000IBF)
LOCLADDR(:::ffff:192.168.112.131(1025))
LSTMSGDA(2015-10-05)    LSTMSGTI(12.20.00)
MCASTAI(RUNNING)        MCAUSER(system)
MONCHL(OFF)             MSGS(3)
```

```
RAPPLTAG(erver\bin\DataFlowEngine.exe)
SECPROT(NONE)      SSLCERTI()
SSLKEYDA( )        SSLKEYTI()
SSLPEER( )         SSLRKEYS(0)
STAIUS(RUNNING)    STOPREQ(NO)
SUBSTAIE(RECEIVE)  CURSHCNV(1)
MAXSHCNV(10)       RVERSION(08000002)
RPRODUCT(MQCC)
```

3. Start the MQSC commands for the IIB10_QM2 queue manager:

```
-bash-4.1$ runmqsc IIB10_QM2
```

4. Display the queue status of the IIB10_QM2 queue manager:

```
dis chs(*) all
```

As shown in Listing 7, the data flow engine (execution group) is listening, and the server-connection channel IIBV10.QM2.SVRCONN is running in the IIB10_QM2 queue manager. These results demonstrate that the MQOutput node is connected to the server-connection channel, indicating successful connectivity.

Listing 7. Flow connection to the IIB10_QM2 queue manager

```
AMQ8417: Display Channel Status details.
CHANNEL(IIBV10.QM2.SVRCONN)  CHLTYPE(SVRCONN)
BUFSRCVD(10)      BUFSENT(9)
BYTSRCVD(2876)    BYTSENT(2844)
CHSTADA(2015-10-05)  CHSTAI(12.22.05)
COMPHDR(NONE,NONE)  COMRMSG(NONE,NONE)
COMPRATE(0,0)      COMPTIME(0,0)
CONNNAME(192.168.112.1)  CURRENT
EXITTIME(0,0)      HBINT(300)
JOBNAME(000010ZB00000000E)
LOCLADDR(:::ffff:192.168.112.131(1026))
LSTMSGDA(2015-10-05)  LSTMSGTI(12.22.05)
MCASTAI(RUNNING)    MCAUSER(sysam)
MONCHL(OFF)         MSGS(7)
RAPPLTAG(erver\bin\DataFlowEngine.exe)
SECPROT(NONE)      SSLCERTI()
SSLKEYDA( )        SSLKEYTI()
SSLPEER( )         SSLRKEYS(0)
STAIUS(RUNNING)    STOPREQ(NO)
SUBSTAIE(RECEIVE)  CURSHCNV(1)
MAXSHCNV(10)       RVERSION(08000002)
RPRODUCT(MQCC)
```

5. Check the CURDEPTH for the OUTPUT.QL queue:

```
DIS QL(OUTPUT.QL) CURDEPTH
```

The message flow picked the message from the INPUT.QL queue on the IIB10_QM1 queue manager and sent it to the OUTPUT.QL queue on the IIB10_QM2 queue manager. As shown in Listing 8, the CURDEPTH has now changed to 1.

Listing 8. OUTPUT.QL CURDEPTH

```
AMQ8409: Display Queue details.
QUEUE(OUTPUT.QL)  TYPE(QLLOCAL)
CURDEPTH(1)
```

6. Browse the sample message on the OUTPUT.QL queue that is on the IIB10_QM2 queue manager:

```
-bash-4.1$ ./amqsget OUTPUT.QL IIB10_QM2
```

You see the following message:

```
Sample AMQSGETO start  
message <XML>I am using CCDT File</XML>
```

Challenge with the CCDT setup

If the CCDT file configurations, such as the Channel Name or Location of the CCDT file are incorrectly defined, you might encounter an error like the one shown in Figure 20.

Figure 20. Incorrect CCDT configurations

(default_EG) Failed to make a CCDT connection to Queue Manager "IIB10_QM1" : MQCC=2; MQRC=2058

Failed to connect to the Websphere MQ queue manager "IIB10_QM1" . The return and reason codes returned by the MQCONN are as displayed. A reason code MQRC-2058 indicates that the CCDT file cannot be located, that the queue manager name is not defined in the CCDT file, or that there is a Websphere MQ connection error. The connection parameters are MQCHLLIB="C:\Users\MVP\Documents" and MQCHLTAB="AMQCLCHL".

Check the Websphere MQ completion and reason codes in the Websphere MQ Application Programming Reference online help to determine the cause of the error, and take the appropriate action. Ensure that the CCDT file location is set in the integration node property 'mqCCDT', and that the integration node is restarted after setting this property

This error message indicates a failure in making a CCDT connection to the queue manager. To correct this error, you must check the following areas:

- MQ Client Channel definitions
- Port numbers
- Location of the CCDT file: After you create the CCDT file, you must transfer the file in Binary mode.
- Client/server connection channel names: The names must match before you create the CCDT file.
- Location of the CCDT File

Important: Do not open or modify the CCDT file. Modifying the file can cause the CCDT not to work and can result in errors.

Conclusion

This tutorial explained the configuration and connections between the IBM Integration Bus v10 and MQ v8. It explained three options to connect an integration node to the MQ queue manager.

These options included the local queue manager, the MQ client connection properties, and the Client Channel Definition Table. The scenarios demonstrated how to create queue managers, MQ channels by using CCDT, and the IBM Integration node. They also explained how to configure IBM Integration Bus message flows by using the MQ Connection properties.

Acknowledgments

A special thank you to Ganesh Gopalakrishnan, Rajish E. Pattavalapil, and Arundeeep B. Veerabhadraiah for their assistance and valuable input provided during the creation of this tutorial.

Resources

Learn more. Develop more. Connect more.

The new [developerWorks Premium](#) membership program provides an all-access pass to powerful development tools and resources, including 500 top technical titles (more than 125 for Java developers alone) through Safari Books Online, deep discounts on premier developer events, video replays of recent O'Reilly conferences, and more. [Sign up today](#).

IBM MQ

- IBM Knowledge Center:
 - [IBM MQ Version 8 documentation](#)
 - [Client channel definition table \(CCDT\) topic](#)
 - [Creating server-connection and client-connection definitions on different platforms topic](#)
- [IBM MQ forum](#) on mqseries.net for user questions, answers, and tips.
- [IBM MQ articles and tutorials](#) in the IBM developerWorks WebSphere zone

IBM Integration Bus

- IBM Knowledge Center
 - [IBM Integration Bus Version 10 documentation](#)
 - [Configuring a local connection to MQ topic](#)
- [IBM Integration Community](#)
- [IBM Integration Bus articles and tutorials](#) in the developerWorks WebSphere zone

© Copyright IBM Corporation 2016

(www.ibm.com/legal/copytrade.shtml)

Trademarks

(www.ibm.com/developerworks/ibm/trademarks/)