WebSphere Application Server Messaging Configuring and using MQ and SIBus

Carl Farkas (With thanks to Matt Leming for much of the material)
SW IOT TechWorks zWebSphere Application Integration Consultant
IBM France D/2708
Paris, France
Internet: farkas@fr.ibm.com
Notes: Carl Farkas/France/IBM @ IBMFR
Agenda

- Quickie review of MQ and SIBus
- Configuration options
- Configuration examples
- Bibliography
Preamble

- WebSphere MQ (ex-MQSeries) has been sold by IBM since 1993. It is in production by over 10,000 customers world-wide. It is estimated that MQ transports between 65-80% of the ”business messages” today.
- WebSphere Application Server v6 arrived in early 2005 with its own built-in messaging engine, the “SIBus” (sometimes called « Default Messaging Provider » or « WAS Messaging Bus » or « Platform messaging » or etc. ! ;)

Which product provides the right messaging for you?

There most probably is no correct answer for this question in all circumstances, but this presentation aims to provide some information on how to use these messaging products separately as well as together, and to provide some elements of comparison.
Transport layers

- WHY DO PEOPLE SEND MESSAGES, DAD?
- SO THEY WON'T HAVE TO TALK TO EACH OTHER.
Review: what is WebSphere MQ?

- A simple, efficient **API** for sending/receiving data messages
- Enables **fast, reliable asynchronous** messaging from application to application
- An industry standard for Message-Oriented-Middlewares (MOM) with over 65-80% of the market.
- Can be used on over 45 **different platforms** (Windows, Linux, AIX, Solaris, HP-UX, iSeries-AS/400, etc., etc., and of course…… System Z !)
- Can be used from all major **programming languages** (C, C++, COBOL, Fortran, BAL, PL/I, Java/JMS, VB, RPG, etc.)
- Includes support for the **MQ Client** – a remote API accessing the queue manager resources.
- Other unique features of MQ…
  - Fully transactional with its own 2PC transaction manager for distributed MQ and working with RRS on z
  - Support for high-availability and load-balancing via MQ Cluster as well as Sysplex on Z
  - Support for point to point and pub/sub messaging
  - Support for message grouping and segmentation
  - Support for SSL
  - Support for message compression
  - Etc.

```java
ioQueue = session.createQueue("MaFile");
QueueSender queueSender = session.createSender(ioQueue);
queueSender.send(outMessage);
```
Review: What is the WAS SIBus?

- SIBus provides *native* support in WAS for JMS messaging
  - Available with WAS v6 on all platforms
  - Full support for JMS 1.1 for WAS applications
  - Full support for point-to-point and pub/sub messaging models

- SIBus names are known throughout the WAS cell
- An SIBus has 1-n **Bus Members**, which are Application Servers, WAS Clusters and/or WAS “MQ Servers”
- Queues and Topics are defined on Bus Members.
- At Runtime, each SIBus Bus Member starts 1-n **Messaging Engines** (MEs); MEs manage the runtime resources (eg. Queues, topics)
  - The ME can store messages in memory, a database (eg. Cloudscape, DB2…) or a file.
  - An SIBus can connect to the outside world (other SIBuses, MQ, etc.)
JMS configuration choices – there are several!

- JMS configuration with SIBus objects
  1. SIBus “pure” – the messages stay in the SIBus world
  2. SIBus peer-to-peer with MQ : MQ Link
  3. SIBus front to MQ : MQ Server
- JMS configuration with MQ objects
  4. MQ direct connection to a local Queue Manager, called “bindings”
  5. MQ network connection to a Queue manager, called MQ “client” mode
- JMS configuration with other providers also possible (but why? ;>)
How do I use JMS from WAS?

- Write a Java program that uses the JMS class libraries
  - This is done by the developer
- Define the logical resources
  - These JMS “pointers” are accessed via JNDI (not strictly necessary)
  - They are independent of the program
  - They are defined by a WAS namespace administrator
- Define the physical resources (eg. Queues)
  - They are defined by a WAS or MQ messaging administrator
SIBus – pure WAS messaging

- For messaging purely in the WAS environment
  - Can be extended beyond the WAS cell also… Other SIBuses, or to MQ world

A. Define the “physical resources”
   1. First create the SIBus
      ✓ Only needs to be done once per WAS Cell (although multiple buses are certainly allowed)
   2. Add Bus Members (eg. Application servers, cells…)
   3. Create your Destinations

B. Define the JMS “pointers”
   4. Define the Connection Factory
   5. Define the Destination pointer (eg. Queues or Topics)
1. SIBus - Create the SIBus

- **Service integration > Buses > New**  then.... Save
- The simplest “wizard” in the world!
2. SIBus - Add the Bus member

- Bus members define the members of the SIBus
- Messaging engines (MEs) are instantiated to represent the Bus members

*Buses > Your_SIBus > Bus members > Add*
2. SIBus - Add the Bus member, cont’d

- Adding a (WAS) Cluster is the key to High-availability
- “WebSphere MQ Server” option to be discussed later
- The “File store” option was added with WAS v6.1
  - Simpler to configure
  - Faster performance
  - A JDBC database probably more robust for HA
- Be sure to Save and Restart your WAS
3. SIBus - Create the Destination

Service integration > Buses > Your_SIBus > Destinations > New....
3. SIBus – Create the Destination, cont’d

- A Destination is “owned” by a Bus member; the BM owns the resources
- Alias, Foreign (as well as MQ Server Queues) create pointers only
  - Similar to a MQ Alias and Remote Queue
  - Probably best to just use the Alias
  - A JMS program can optionally directly address a queue without using a pointer
4. JMS – Define a Connection Factory

- The Connection Factory is the JNDI-administered object that a program connects to and provides the interface to the messaging system
  - Creates a “handle”; in MQ terms, this is like an MQCONNN (although strictly speaking, it’s actually the createSession that does the MQCONNN)

- Careful! There are 3 choices!
  - Connection Factory – if your JMS application uses JMS 1.1 (nice work!)
  - Queue and Topic Connection Factories – if your JMS application is older (time to update!)

- One CF per messaging provider is often sufficient

Resources > JMS > Connection Factory > New
4. JMS – Define a Connection Factory, cont’d

- Other provider choices can also appear (e.g. WAS “v5 default messaging”, xyz messaging, etc)
- You must select the SIBus or QM that you want to work with here
- Other connection factors (e.g. Security, transactionality, performance pooling) are here
5. JMS – Define the destination pointer

Resources > JMS > Queues > New > Default messaging provider

Specify the JNDI name used by Java application

After selecting the SIBus to be queried....

.... the known queue names will be proposed. Cool!
2. **SIBus MQ Link**

- A WAS “MQ Link” emulates a WMQ Queue Manager.
  - The MQ Queue Manager thinks that it’s another MQ on the other side!
  - A simple peer-to-peer relationship

- Unidirectional channels connect the Messaging Engine with the QM
- Standard store ‘n forward messaging between the SIBus and the QM

A. Define the “physical resources”
   1. First define a Foreign Bus on the SIBus
   2. Next define the MQ Link
   3. Create your Destinations - Alias

B. Define the JMS “pointers”
   4. Define the Connection Factory
   5. Define the Destination pointer (eg. Queues or Topics)
MQ Link – SIBus networks

- An MQ Link can be used as a “gateway” into the MQ world
- Or as a router between different SIBus worlds
- Or as a router between different MQ worlds
- Messages are fully preserved between worlds
  - By MQMD properties when possible
  - Otherwise kept in SIBus folder in RFH2
1. MQ Link – Define the Foreign Bus

- The Foreign Bus is a “proxy” within the SIBus to represent another bus (SIBus, MQ, etc)
- Bus “Name” should typically be that defined by the partner

Service integration > Buses > your_Bus_name > Foreign buses
2. MQ Link – Define the MQ Link

- The MQ Link gets defined on the runtime Messaging Engine of an SIBus
- Be sure that the SIBus is defined and WAS restarted

Service integration > Buses > Your_SIBus > Messaging en
WebSphere MQ links > New
2. MQ Link – Define the MQ Link, cont’d

Service integration > Buses > Your_SIBus > Messaging engines > Your_ME then....

WebSphere MQ links > New

- Name of SIBus virtual QM
- Must match MQ RECEIVER channel definition!
- Must match MQ SENDER channel definition!
2. MQ Link – Define the MQ Link, cont’d

- And on the MQ side, it looks like this
- Note that the channel names must match *exactly* on both sides!
- Use the port number SIB_MQ_ENDPOINT_ADDRESS found at Application servers > your_server > Ports to define the MQ Sender Connection name (port)

SIBus → MQ Receiver

MQ Sender → SIBus

**Must match the SIBus MQLink Sender channel**

**Must match the SIBus MQLink Receiver channel**

**Must match the SIBus MQLink Receiver channel**
3. MQ Link - Create the SIBus Destination

Service integration > Buses > Your_SIBus > Destinations > New....
3. MQ Link - Create the SIBus Destination, cont’d

- Alias and Foreign options create pointers only
  - Similar to a MQ Alias and Remote Queue
  - Probably best to just use the Alias
  - A JMS program can optionally directly address a queue without using a pointer
- More sophisticated (complex!) Target Identifier’s can be used (eg. MyQueue@QMgry) to route via a gateway Target bus
4. JMS – Define a Connection Factory
5. JMS – Define a destination pointer

- These two steps are essentially identical to what we saw earlier for a “pure” SIBus definition.
- As before, create a Connection Factory to “point to” the SIBus
- As before, create a destination pointer (eg. Queue or Topic) for the particular resource that we just created above.

Resources > JMS > Queues > New > Default messaging provider

Name referred to by Java JMS program
Choose the queue name defined on the SIBus in the prior step
**3 SIBus MQ Server**

- A WAS “MQ Server” connects to a WMQ Queue Manager using a… “MQ Client” (!!)  
  - Only supported by WebSphere MQ on z/OS  
  - Can also use MQ “bindings” mode (if WAS running locally with MQ)
- Bidirectional connection between the Messaging Engine with the QM yielding “local-like” interface to MQ (putting and getting messages)
- Fully exploits MQ as the queue resource holder (eg. Shared queue access)

A. Define the “physical resources”
   1. Define a MQ Server on the SIBus
   2. Add the MQ Server as an SIBus member
   3. Create your Destinations - Alias

B. Define the JMS “pointers”
   4. Define the Connection Factory
   5. Define the Destination pointer (eg. Queues or Topics)
1. MQ Server – Define the MQ Server

Servers > WebSphere MQ server > New

- Must match the QM name
- If you’re lucky enough to have WAS on z/OS!
2. MQ Server – Add MQ Server as Bus Member

Service integration > Buses > Your_bus > Bus members > Add
3. MQ Server - Add the SIBus Destination

- You must add the queue definition (and not an Alias), but this isn't the “physical” queue definition; you still must use MQ administration to create the queue.

Service integration > Buses > Your_SIBus > Destinations > New....
3. MQ Server - Add the SIBusDestination, cont’d

- After selecting the MQ Server as the Bus member, the WAS Admin Console wizard automatically presents the MQ queue names available.

**Important for MQ "legacy" compatibility**
4. JMS – Define a Connection Factory
5. JMS – Define a destination pointer

- These two steps are essentially identical to what we saw earlier for a “pure” SIBus definition.
- As before, create a Connection Factory to “point to” the SIBus
- As before, create a destination pointer (eg. Queue or Topic) for the particular resource that we just created above.

*Resources > JMS > Queues > New > Default messaging provider*
MQ direct connection

- Requires a WebSphere MQ Queue Manager installed locally on the same machine as WAS
- Takes full advantage of MQ store ‘n forward messaging
- Minimises WAS resources – no need to create an SIBus at all
- Uses MQ-supplied class libraries over native (JNI) interface
- Reduced “visibility” from WAS

A. Define the “physical resources”
   1. Done on MQ by the MQ administrator

B. Define the JMS “pointers”
   2. Define the Connection Factory
   3. Define the Destination pointer (eg. Queues or Topics)
2. JMS – Define a Connection Factory

- This is identical to what was seen earlier for addressing the SIBus resources, except the WebSphere MQ messaging provider is chosen.
- Many other connection factors (eg. Security, transactionality, performance pooling, pub/sub interface, etc.) are here.

- Resources > JMS > Connection Factory > New

This is the critical option to indicate a direct, local MQ connection.
3. JMS – Define the destination pointer

- Again, this is identical to the JMS definition seen earlier for addressing the SIBus resources, except the WebSphere MQ messaging provider is chosen.

*Resources > JMS > Queues > New > WebSphere MQ messaging provider*
3. JMS – Define the destination pointer, cont’d

- This is where most default message-specific properties are defined

Specify the JNDI name used by Java application

Specify the queue name defined on MQ

Determines if MQ RFH2 used for JMS properties
MQ network connection

- Requires a WebSphere MQ Client which connects to a Queue Manager typically installed remotely on a different machine than the WAS (CAF required on MQ z/OS)
- Uses a single, bi-directional MQ channel providing remote MQ API access
- Takes full advantage of MQ store ‘n forward messaging
- Minimises WAS resources – no need to create an SIBus at all
- Uses MQ-supplied class libraries
- Reduced “visibility” from WAS
- Lower throughput and greater WAS overhead than the MQ direct connection

A. Define the “physical resources”
   1. Done on MQ by the MQ administrator

B. Define the JMS “pointers”
   2. Define the Connection Factory
   3. Define the Destination pointer (eg. Queues or Topics)
2. JMS – Define a Connection Factory

- This is *almost* identical to what was seen earlier for the MQ direct connection, except the WebSphere MQ Queue Manager host connection parameters must be supplied.
- Many other connection factors (eg. Security, transactionality, performance pooling, pub/sub interface, etc.) are here.

Resources > JMS > Connection Factory > New

Specify the exact QM name along with the IP address and port.

This is the critical option to indicate a network MQ connection.
3. JMS – Define the destination pointer

- This is identical to the JMS definition seen earlier for addressing the MQ direct connection resources

Resources > JMS > Queues > New > WebSphere MQ messaging provider
Message Driven Beans (MDB)

- Message Driven Beans (MDBs) are used to “wake up” a message-receiving application
- The application (Java class) is pre-loaded by WAS
- You must tell WAS which queue to monitor
  - For SIBus (pure JCA 1.5), this is done via an Activation specifications
  - For MQ, this is done via a Listener Port
- Link to application is pre-configured in EAR, or by Enterprise applications menu > your_app > MDB listener bindings

```java
public void onMessage(javax.jms.Message msg) {
    System.out.println("==> Woke up with :" + msg);
}
```
Message Driven Bean (MDB) with the SIBus

Resources > JMS > Activation specifications > New > Ok

Application specification refers to this JNDI (JMS) name

JNDI (JMS) name of queue
Message Driven Bean (MDB) with MQ

Application servers > your_app_server > Messaging > Message Listener Service > Listener Ports > New

Application specification refers to this name

JNDI (JMS) name of queue and CF
Message Driven Bean – application link - port

- Application link to listener typically in the .jar with the .ear

```xml
<ejbBindings xmi:type="ejb:MessageDrivenBeanBinding" xmi:id="MessageDrivenBeanBinding_1108036949665"
  listenerInputPortName="LP">
  <enterpriseBean xmi:type="ejb:MessageDriven" href="META-INF/ejb-jar.xml#testMDB"/>
</ejbBindings>
```

- Otherwise… it can be defined by
  
  Enterprise Applications > your_WAS_Appli > MDB listener bindings
To be added.... Quand j’aurai le temps

- Administration
- Sécurité
- Qqs exemples Pub/Sub
- Plus d’info sur la conversion formats MQ-JMS
Bibliography

- WAS product information
  - http://www-306.ibm.com/software/webservers/appserv/was/
- WAS V6.1 InfoCenter
- IBM RedBooks
  - http://www.redbooks.ibm.com
    - WebSphere Version 6 Web Services Handbook Development and Deployment, SG24-6461-00
    - WebSphere Application Server V6.1: System Management & Configuration, SG24-7304-00
    - WebSphere Application Server V6 Problem Determination for Distributed Platforms, SG24-6798-00
    - WebSphere Application Server V6 Scalability and Performance Handbook, SG24-6392-00
    - WebSphere Application Server V6.1 Security Handbook, SG24-6316-01
    - Patterns: SOA with an Enterprise Service Bus in WebSphere Application Server V6, SG24-6494-00
    - WebSphere Application Server V6.1: Technical Overview, REDP-4191-00
    - WebSphere Application Server V6 Default Messaging Provider Problem Determination, REDP-4076
    - WebSphere Application Server V6.1: Planning and Design, SG24-7305-00
    - WebSphere Application Server V6.1: Installation Problem Determination, REDP-4305-00
- IBM developerWorks
  - (Searching on “Service Integration Bus” returns a number of interesting articles)