



IBM Software Group

Understanding Stuck Messages (Addendum)

Paul O'Donnell and
Ty Shrake



WebSphere® Support Technical Exchange



Agenda

- Introduction
- Overview of Message Flow
- Message Producers and Consumers
- Message Driven Bean Basics
- MDB Transactionality
- How Message Driven Beans Work
- Resource Adapters
- Inside onMessage()
- Message Accumulation
- Identifying the Problem
- Resolving the Problem
- Javacores
- Message States
- Summary



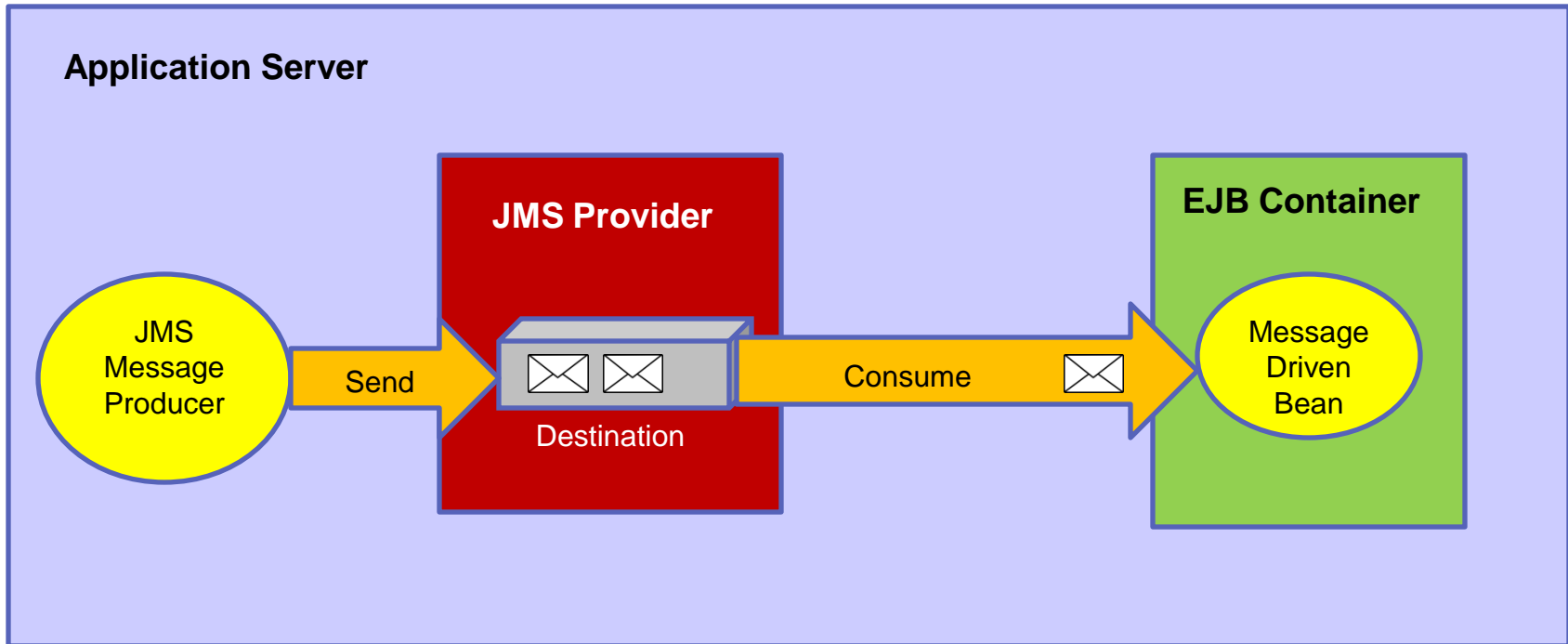
Introduction

Messages “stuck” on a destination is one of the most common problems customers encounter in messaging systems. In most cases this results in messaging stacking up or accumulating on a destination and a failure to process business data. This presentation will explain why message accumulation occurs and how to troubleshoot this type of problem.



Overview of Message Flow

When messages are stuck on a queue it is important to have a basic understanding of message flow in your environment. In 99% of all cases the message flow is very similar to the flow shown below:



Message Producers and Consumers

Messages are created in Producer applications. Once the message is created the application sends the message to a destination. This process usually works fine and is rarely problematic.

Most problems with messages involve the message Consumer application, which is supposed to 'consume' (remove) the message from the destination and process the business data in the message. Most investigations into message 'stuck' on a queue/destination should start with the Consumer application, not the Producer. In Websphere Application Server message consumers are almost always **Message Driven Beans**.



Message Driven Bean Basics

When investigating messages that are stuck on a queue or are accumulating on a queue it helps a great deal if you understand what Message Driven Beans are and how they work.

A **Message Driven Bean (MDB)** is a special case of an Enterprise Driven Bean (**EJB**). There are 3 types of EJBs: Session Beans, Entity Beans and Message Driven beans.

- **Session Beans:** Session beans are Java applications that service a single client. There are 2 kinds of Session beans: Stateless and Stateful.
- **Entity Beans:** An entity bean is simply a representation of a thing, such as a book or CD.
- **Message Driven Bean:** A Message Driven Bean (**MDB**) is a special case of EJB. It is usually a fairly small application *whose purpose is to receive (consume) messages from a messaging system*.

Note: All EJBs run inside the **EJB Container**. The container manages each bean and ensures the bean can access the services provided by the application server environment.



Message Driven Bean Basics

An MDB is an application whose purpose is to receive (consume) messages from a messaging system.

Once the MDB consumes the message it will perform a small amount of work on the message and then pass the message data to a larger application (like a Session bean that performs the heavy duty work, such as storing the message data in a database, etc...) MDBs are specifically designed to take the messaging load off of larger applications that are busy with business logic, database updates, order confirmation, etc... And to simplify the coding of consumer applications.

Important Features of MDBs:

1. MDBs do not need any connection code to consume messages.
2. All MDBs have a method named **onMessage()**.
3. MDBs run inside an EJB Container



Message Driven Bean Basics

Most developers use Eclipse to create MDBs for WAS. Many times they use Rational Application Developer (RAD), which includes Eclipse and WAS.

When you create an MDB there are 2 basic parts:

- **onMessage()** method. This is the method (code) that will process the message that is passed to the bean. Many times this code will call other beans.
- The **Deployment Descriptor (DD)**. The DD is an XML file named **ejb-jar.xml**. This is the main configuration file for the MDB and holds various properties and behaviors for the MDB.

After you create your MDB you must **deploy** it (install it) into WAS. Once deployed the EJB Container will use the ejb-jar.xml file to associate (bind) the MDB to the correct JMS resources.



MDB Transactionality

The Deployment Descriptor of an MDB (**ejb-jar.xml**) holds important information about how the bean behaves and what resources it will use. One of the items in the DD is the transactionality the bean will use. There are 2 basic options:

1. **Container Managed** – This means that each message consumed by the MDB will be part of a transaction created and managed by the EJB container. This is the best option to use.
2. **Component or Bean Managed** - This means that the MDB itself is responsible for any transactions. This option should generally be avoided.

This is worth knowing because the State of a message on a destination is often determined by these choices.



How Message Driven Beans Work

When we talk about Message Driven Beans we often say that they “consume messages”. This is true but the situation is a bit more complex than this.

When discussing MDB message consumption we must break the process down into 3 parts: The Destination, the Resource Adapter and the MDB. *MDBs do not have any connection code in them the way normal, standalone JMS applications do.* Instead, a **Resource Adapter** connects to a destination on behalf of the MDB. **The Resource Adapter listens on the queue for messages and, when a message arrives, it passes it to the MDB.** When a message arrives on a destination the Resource Adapter gets a copy of the message from the destination and then calls the `onMessage()` method of the MDB. The `onMessage()` method takes a `Message` object as an argument (`onMessage(message)`). In other words, the Resource Adapter **delivers** the message to the MDB. So when we talk about MDB message consumption what we are really talking about is **message delivery** to an MDB. More on Resource Adapters later...

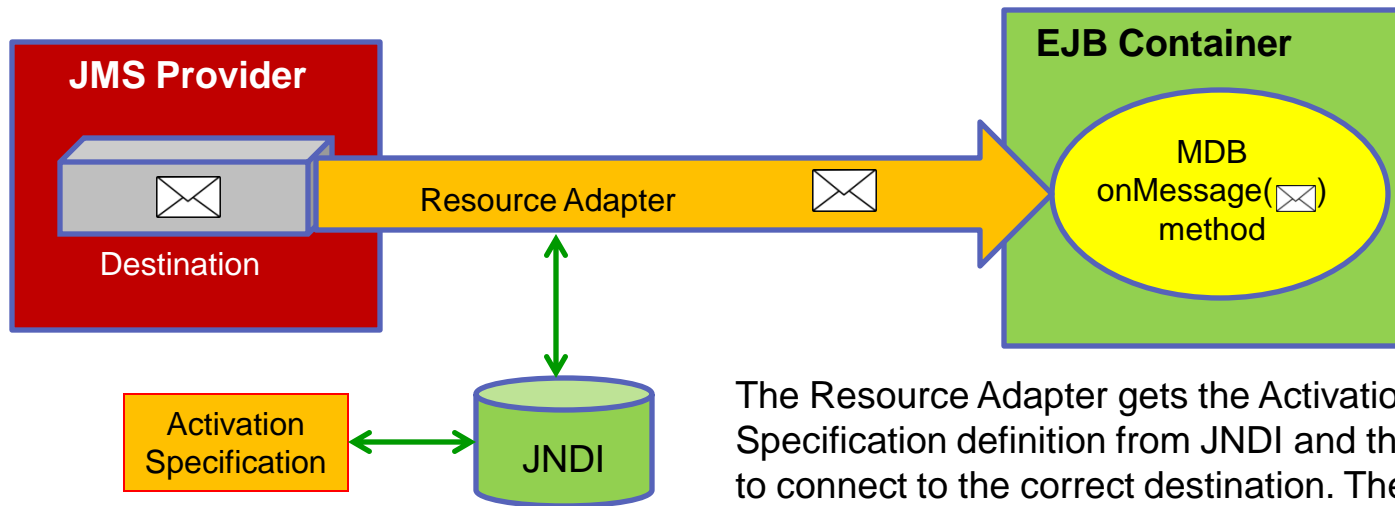
As noted before, once the MDB has the message it usually extracts the data (payload) and then passes the data to another bean for processing.



Resource Adapters

A **Resource Adapter (RA)** is a *program* that connects to a JMS destination, listens for messages arriving on that destination, and then passes the messages to the `onMessage()` method of the MDB using that RA.

The RA is configured by an **Activation Specification (AS)**. The AS is just a configuration object that tells the RA what destination to connect to and how to manage message delivery to the MDB. The AS is stored in JNDI. When the Resource Adapter starts it reads the AS connection information and uses that to actually connect to the destination.



The Resource Adapter gets the Activation Specification definition from JNDI and then uses that to connect to the correct destination. The AS also configures the message delivery options in the RA .

Inside onMessage()

MDB's that consume JMS messages have a method named **onMessage()** in their code. There are other methods as well (beyond the scope of this education) but onMessage() is the only method we are concerned with. The onMessage() method takes a Message object as its argument, as follows:

onMessage(your_msg)

In the MDB code the basic onMessage() method looks like this:

```
public void onMessage(javax.jms.Message msg) {
```

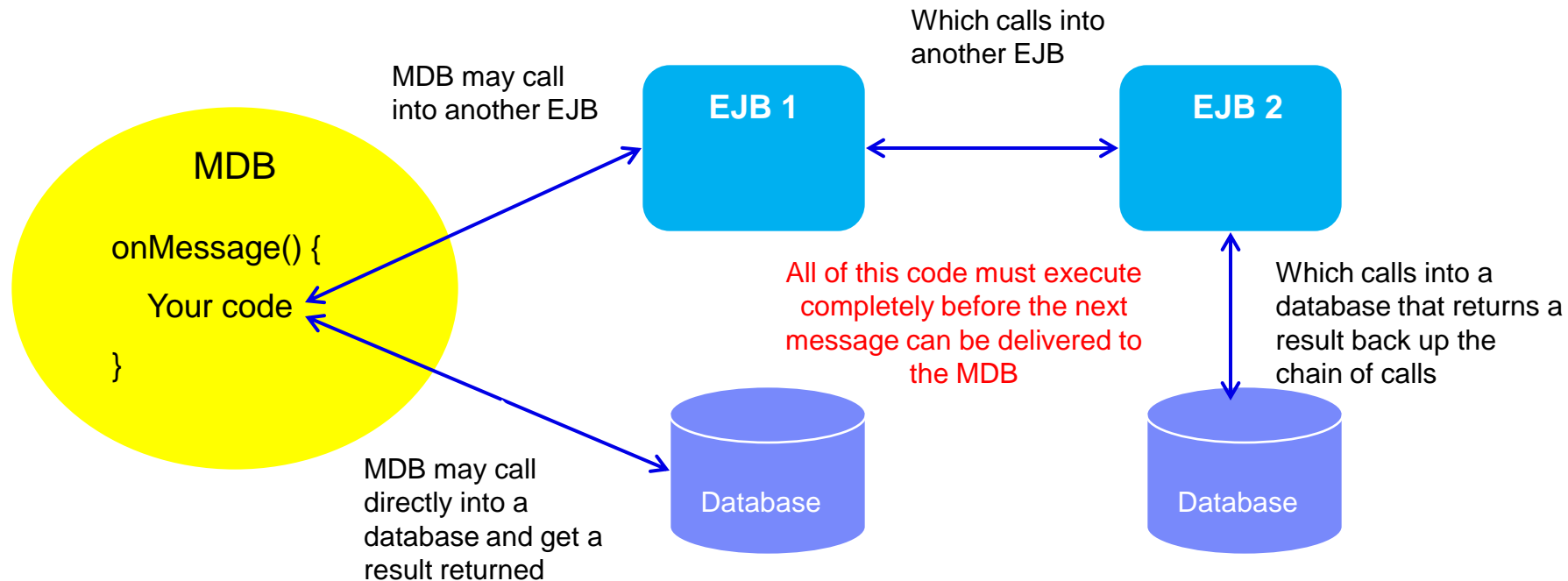
Your business code goes here...

```
} //end onMessage()
```



Inside onMessage()

The code inside the onMessage() method processes the message data. In most cases the data inside the message goes through a little processing and then other business objects (EJBs) are called for additional processing. For example...



Inside onMessage()

It is crucial to understand that once the code inside of `onMessage()` is executing the thread the MDB is running on is owned by that code. We are outside of SIB code at this point. When the message data is passed from the MDB to a database or another EJB or anything else all of that code must successfully execute and return (complete). Once all of that code completes then, and only then, will the `onMessage()` method return (complete). It is not possible to deliver another message to the MDB until `onMessage()` returns. If there are any problems anywhere in the chain of code then `onMessage()` may not return. If `onMessage` does not return, or takes a long time to return, then messages might accumulate on the destination the MDB consumes messages from.

Also note that if a problem occurs it may not be the MDB itself, **it might be a problem in an EJB that was called, or a database issue.**

As we'll see later in this presentation, Javacores can reveal if an MDB is having trouble processing a message.



Message Accumulation

Now that we know that an MDB has messages *delivered to it by a Resource Adapter*. And we also know that once the message is delivered to the MDB the MDB owns the message and the thread and that all processing inside of `onMessage()`, and anywhere called from inside of `onMessage()`, must complete before a new message can be delivered to the MDB. In a perfect system each message that arrives on the destination will be instantly consumed by an MDB. The message will be delivered to the MDB, the MDB will process the message and then return control back to the EJB Container, allowing the next message to be delivered. This usually happens in a matter of milliseconds.

But what if messages start to accumulate or stack up on the queue? *This is usually the first hint to a customer that something isn't working correctly.* There are generally 2 causes of message accumulation:

1. Message consumption by the MDB is slower than message production to the destination. In other words, messages are sent to the destination faster than they can be removed by the MDB. Alternatively the MDB may not be running at all.
2. The MDB is having trouble processing a message and is effectively stopped.



Identifying the Problem

In order to determine which of these 2 conditions is happening you must first view the messages on the destination. You can do this in the WAS Administration Console by navigating as follows:

Service Integration > Buses > YOUR_BUS > Messaging engines > YOUR_MESSAGING_ENGINE > Queue points > YOUR_QUEUE_POINT (Runtime tab) > Messages

Here you will find the list of messages currently on the queue (destination) you are interested in. Ideally there should be zero or nearly zero messages on the destination. Notice there are several columns displayed for each message. **Message State is the most important column.** The most common states are:

- ▶ **Available**
- ▶ **Removing**
- ▶ **Locked**
- ▶ **Unlocked**

There are other states as well that will be discussed at the end of this presentation.



Identifying the Problem

When you have messages stuck on a destination the most important things to observe are:

- **The first message at the top of the queue**
- **The value in the State column**
- **The value in the Transaction ID column (blank or non-blank)**

JMS messaging is First In, First Out, (FIFO) which means that the message displayed at the top of the message list is the next message to be consumed. In many cases this message will show some non-Available state while all other messages below it show Available. Focus on the top message.

The State column will show the current state of the message. Ideally this state should only last a few milliseconds. When messages show the same state over a long period of time it indicates a problem with the message consumer application (MDB).

There may or may not be a transaction ID (some consumers are transacted, some are not). The important point is that if a message is transacted and is stuck you may be able to manually roll back the transaction. It is unlikely that you will be able to manually commit it.



Identifying the Problem

In this example notice that the messages have a State of **Available** and there is **no Transaction ID**. This means the messages are available to be delivered to an MDB. If the list of messages changes over time then the message ARE being delivered. If the list does not change then it's likely there is a problem with the MDB itself.

Buses

[Buses](#) > [Bus1](#) > [Messaging engines](#) > [Cluster1.000-Bus1](#) > [Queue points](#) > [Q5@Cluster1.000-Bus1](#) > **Messages**

The messages on the message point.

⊞ Preferences

Delete Delete all Refresh Jump to page... ▾

⊞ ⊞ ⊞ ⊞

Select	Position ▾	System message ID ▾	State ▾	Transaction ID ▾
You can administer the following resources:				
<input type="checkbox"/>	001	0030B6115ECDD829 54507690	Available	
<input type="checkbox"/>	002	0030B6115ECDD829 54507691	Available	
<input type="checkbox"/>	003	0030B6115ECDD829 54507692	Available	

Two red arrows point to the 'State' and 'Transaction ID' columns, highlighting that all messages are 'Available' and have no transaction IDs.

Identifying the Problem

If the list is constantly changing (messages are being delivered to an MDB) but the list continues to grow in size (the total number of messages on the destination is increasing) then this indicates that messages are arriving on the destination faster than they can be removed. This can often be corrected by increasing the **Maximum concurrent MDB invocations per endpoint** in the Activation Specification used by the MDB. The default value is **10**.

Maximum concurrent MDB invocations per endpoint

Note: Changing this value will require a restart of your messaging cluster.

This means that up to 10 instances (copies) of the same MDB can run simultaneously. While 1 instance of the MDB is busy processing a message one another instances can consume the next message. This can greatly increase the rate of message consumption. Each MDB instance runs on a different thread. Using more than 1 MDB instance is sometimes called **MDB Throttling**.

Identifying the Problem

If the messages on the destination are not moving, in other words the same messages remain on the queue for long periods of time, then either the MDB is not running or it is running but has stopped processing messages.

1. If the MDB is not Started the messages will show a State of **Available**. The current status of the MDB can be checked in the WAS Administration Console here:

Applications > Application types > Websphere enterprise applications > YOUR_MDB

Check the Application Status column. If the MDB is not Started try manually starting it from this console panel.

2. If the MDB is Started then check the State of the messages on the destination.



Identifying the Problem

Many times, when an MDB is Started but messages are not being consumed from the destination, the list of messages will show at least 1 message in a **Removing** state and it will have a Transaction ID, as follows:

Buses

[Buses](#) > [Bus1](#) > [Messaging engines](#) > [Cluster1.000-Bus1](#) > [Queue points](#) > [Queue1@Cluster1.000-Bus1](#) > **Messages**

The messages on the message point.

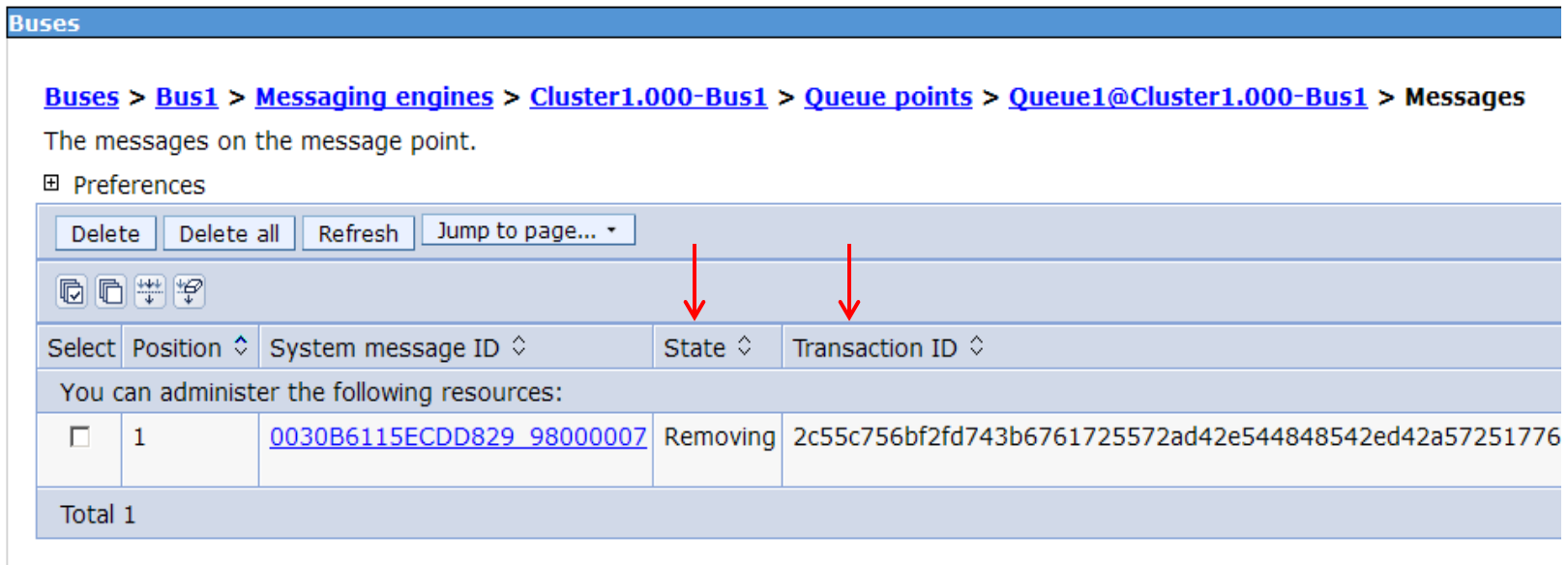
▣ Preferences

Delete Delete all Refresh Jump to page... ▾

☑ ☐ ⬆ ⬇ ⬇ ⬆

Select	Position ▾	System message ID ▾	State ▾	Transaction ID ▾
You can administer the following resources:				
<input type="checkbox"/>	1	0030B6115ECDD829 98000007	Removing	2c55c756bf2fd743b6761725572ad42e544848542ed42a57251776

Total 1



This means that the message is currently being removed from the destination under a transaction that hasn't completed. **This often indicates a problem with the MDB.**

Identifying the Problem

Sometimes the list of messages will show at least 1 message in a **Locked** state with no Transaction ID, as follows:

Buses

[Buses](#) > [Bus1](#) > [Messaging engines](#) > [Cluster1.000-Bus1](#) > [Queue points](#) > [Queue1@Cluster1.000-Bus1](#) > **Messages**

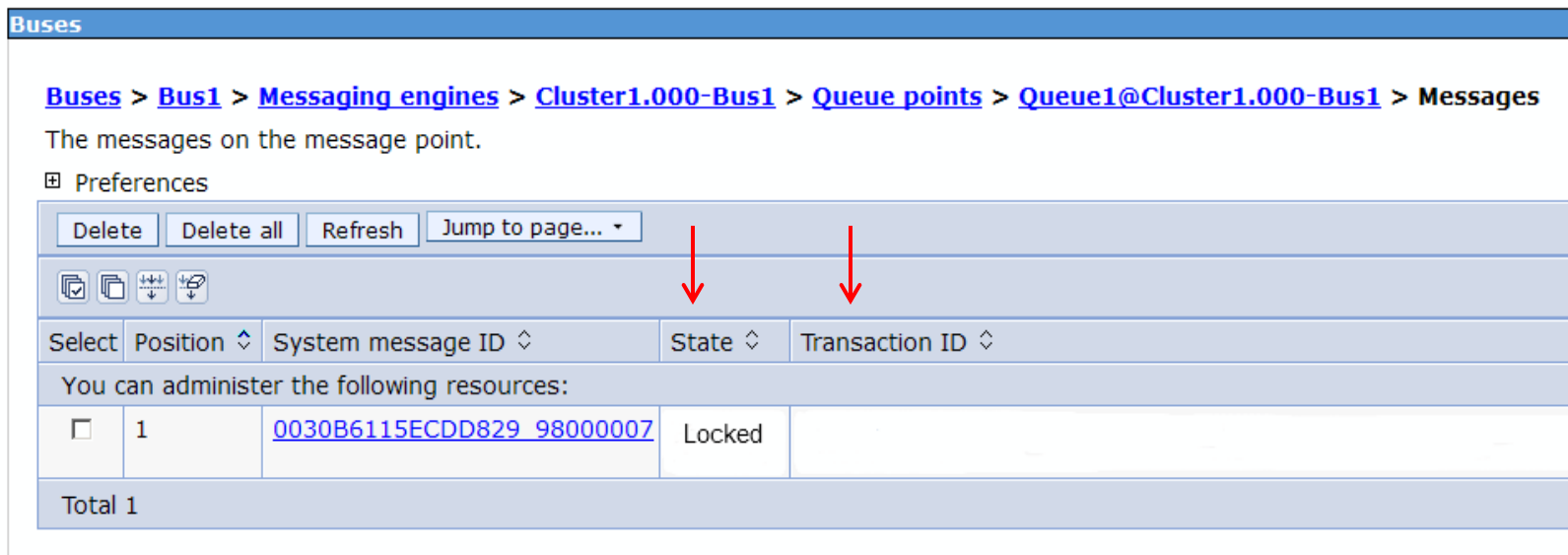
The messages on the message point.

⊞ Preferences

Delete Delete all Refresh Jump to page... ▾

⊞ ⊞ ⊞ ⊞

Select	Position ▾	System message ID ▾	State ▾	Transaction ID ▾
You can administer the following resources:				
<input type="checkbox"/>	1	0030B6115ECDD829_98000007	Locked	
Total 1				



This message is most likely being consumed by a non-transacted MDB. **If this message stays in this state for a long period the MDB is the most likely cause.**

Resolving the Problem

The following slides will discuss solving stuck messages in 6 common scenarios:

- Messages are **Available** but **moving slowly**
- Messages are **Available** and **not moving**
- Messages are in a **Removing** state **with Transaction ID** but **moving slowly**
- Messages are in a **Removing** state **with Transaction ID** but **not moving**
- Messages are in a **Locked** state with **no Transaction ID** but **moving slowly**
- Messages are in a **Locked** state with **no Transaction ID** but **not moving**
- Messages are in a **Locked** state with **Transaction ID** but **moving slowly**
- Messages are in an **Unlocked** state with **no Transaction ID** but **not moving**



Resolving the Problem

- Messages are **Available** but moving slowly:

Interpretation: Messages are being consumed by the MDB but at a slow rate.

Possible Solution:

- ▶ Try increasing Maximum MDB Concurrency (see slide 18)
- ▶ Check MDB performance by adding timestamps to the MDB source code or gather 3 Javacores at about 30 second intervals to see where the MDB is spending time.



Resolving the Problem

- Messages are **Available** and not moving:

Interpretation: The MDB may not be started.

Possible Solution:

- ▶ Check the state of your MDB and make sure it is in a Started state.
- ▶ Manually stop and restart your MDB
- ▶ If the procedure above does not work and the MDB is Stared, gather 3 Javacores against the PID of the server hosting the MDB.



Resolving the Problem

- Messages are in a **Removing** state **with Transaction ID** but moving slowly:

Interpretation: Messages are being consumed by the MDB but at a slow rate.

Possible Solution:

- ▶ Increase Maximum MDB Concurrency (see slide 18)
- ▶ Manually stop and restart the MDB
- ▶ Check MDB performance by adding timestamps to the MDB source code or gather 3 Javacores at about 30 second intervals to see where the MDB is spending time.



Resolving the Problem

- Messages are in a **Removing** state **with Transaction ID** but not moving:

Interpretation: Messages are being consumed by the MDB but at a slow rate.

Possible Solution:

- ▶ Increase Maximum MDB Concurrency (see slide 18)
- ▶ Manually stop and restart the MDB
- ▶ Check the state of the transaction and try to manually complete the transaction by navigating in the WAS Administration Console as follows:

Servers > Server Types > WebSphere application servers > [Content pane] *server_name* > [Container Settings] Container Services > Transaction Service > Runtime > Manual transactions - Review

Resolving the Problem

To list the resources used by a transaction, click the transaction local ID in the list displayed.

To act on one or more of the transactions listed, select the check boxes next to the transactions that you want to act on, then use the buttons provided. You can either manually Commit or Rollback the transaction.

- ▶ If the procedure above does not work gather 3 Javacores, at 1 minute intervals, against the PID of the server hosting the MDB.



Resolving the Problem

- Messages are in a **Locked** state **with no Transaction ID** but moving slowly:

Interpretation: Messages are being consumed by the MDB but at a slow rate.

Possible Solution:

- ▶ Increase Maximum MDB Concurrency (see slide 18)
- ▶ Manually stop and restart the MDB
- ▶ If the procedures above do not work gather 3 Javacores against the PID of the server hosting the MDB.



Resolving the Problem

- Messages are in a **Locked** state **with no Transaction ID** but not moving:

Interpretation:

- ▶ The message has been passed to a consumer, and that consumer has not yet completed.
- ▶ The message has been passed to a consumer, and that consumer has completed but we haven't tried to remove the message yet.
- ▶ The message has been passed over the network to a client application pre-emptively (up to 1 per client with read-ahead disabled, and many more if read-ahead is enabled) - the client has not yet consumed it.

Possible Solution:

- ▶ Increase Maximum MDB Concurrency (see slide 18)
- ▶ Manually stop and restart the MDB
- ▶ If the procedures above do not work gather 3 Javacores against the PID of the server hosting the MDB.



Resolving the Problem

- Messages are in a **Locked** state **with Transaction ID** but moving slowly:

Interpretation:

- ▶ An application has consumed the message with a transaction, and is performing some processing that is taking a long time to complete (or it may have deadlocked).
- ▶ A transaction has become in-doubt (one of the resources managers failed during the prepare phase or between prepare and commit), and the transaction manager has not been yet able to contact the messaging engine and tell it whether to commit/rollback the transaction.

Possible Solution:

- ▶ Increase Maximum MDB Concurrency (see slide 18)
- ▶ Manually stop and restart the MDB
- ▶ If the procedures above do not work gather 3 Javacores against the PID of the server hosting the MDB.



Resolving the Problem

- Messages are in an **UnLocked** state **with no Transaction ID** but not moving:

Interpretation:

- ▶ No thread is listening for messages on this destination.
- ▶ The match criteria of all the consumers for this destination do not match the message.
- ▶ All consumers for this destination are pointing at a different partition of the destination in a WLM environment.
- ▶ All consumers are maxed out processing messages and can't accept new ones (this would normally happen when we have a mix of **locked** and **unlocked** messages).



Resolving the Problem

Possible Solution:

- ▶ Make sure the Activation Specification for the MDB is configured correctly
- ▶ Increase Maximum MDB Concurrency (see slide 19)
- ▶ Manually stop and restart the MDB
- ▶ If the procedures above do not work gather 3 Javacores against the PID of the server hosting the MDB.



Javacores

In situations where messages are not moving off of a destination (customer often describe them as being “stuck” on the queue) it is usually a good idea to gather Javacores. Javacores are a ‘snapshot’ of the Java Virtual Machine (JVM) that show what each thread in the JVM is doing. For stuck messages Javacores should be gathered against the JVM running the MDB. A sequence of 3 javacores, about 1 minute apart, should be gathered. On Unix systems the basic command to gather a Javacore is:

```
kill -3 PID
```

More detailed instructions can be found here:

Windows:

http://www-01.ibm.com/support/docview.wss?rs=0&context=SSCMPB9&context=SSCMP9J&q1=MustGatherDocument&uid=swg21138203&loc=en_US&cs=utf-8&lang=en



Javacores

Solaris:

http://www-01.ibm.com/support/docview.wss?rs=0&context=SSCMPB9&context=SSCMP9J&q1=MustGatherDocument&uid=swg21115625&loc=en_US&cs=utf-8&lang=en

AIX:

http://www-01.ibm.com/support/docview.wss?rs=0&context=SSCMPB9&context=SSCMP9J&q1=MustGatherDocument&uid=swg21052641&loc=en_US&cs=utf-8&lang=en



Javacores

A good tool for reading Javacores is the **IBM Thread and Monitor Dump Analyzer for Java**. You can find it here:

<https://www.ibm.com/developerworks/mydeveloperworks/groups/service/html/communityview?communityUuid=2245aa39-fa5c-4475-b891-14c205f7333c>

When you use this tool, and you suspect that an MDB may be hung or unresponsive, you will be looking for a thread that has **onMessage()** in the call stack. The following example shows that after a message is delivered to `onMessage()` the MDB becomes unresponsive because it is waiting on a database. This is a typical example of why messages get stuck (and accumulate) on a destination. If the MDB has to wait forever on the database no new messages can be delivered to the bean. In the example the important highlights are in **red**. The slide after that will walk you through the analysis of this call stack.

The pattern you are looking for is this: **A thread that shows the `onMessage()` method is stuck in exactly the same spot in all 3 javacores. That almost always means the MDB is hung.**



"Default : 0" (TID:0x000000001A774400, sys_thread_t:0x0000000019544630, state:B, native ID:0x0000000000003566) prio=5

22 at com/company3/common/jdbc/**SimpleDataSource.popConnection**(SimpleDataSource.java:566(Compiled Code)) < HANG

21 at com/company3/common/jdbc/SimpleDataSource.getConnection(SimpleDataSource.java:222)

20 at com/company3/sqlmap/ernie/transaction/jdbc/JdbcTransaction.init(JdbcTransaction.java:48)

19 at com/company3/sqlmap/ernie/transaction/**jdbc**/JdbcTransaction.getConnection(JdbcTransaction.java:89(Compiled Code))

18 at com/company3/sqlmap/ernie/mapping/statement/MappedStatement.executeQueryForObject(MappedStatement.java:120)

17 at com/company3/sqlmap/ernie/impl/SqlMapExecutorDelegate.queryForObject(SqlMapExecutorDelegate.java:518)

16 at com/company3/sqlmap/ernie/impl/SqlMapExecutorDelegate.queryForObject(SqlMapExecutorDelegate.java:493)

15 at com/company3/sqlmap/ernie/impl/SqlMapSessionImpl.queryForObject(SqlMapSessionImpl.java:106)

14 at com/company3/sqlmap/ernie/impl/SqlMapClientImpl.queryForObject(SqlMapClientImpl.java:82)

13 at com/company1/foo/data/service/impl/company3/company2DataServiceImpl.upsertServiceRequestStatus(Bytecode PC:120)

12 at com/company1/foo/services/messaging/tasks/company2/company2SrTask.refreshSR(Bytecode PC:127)

11 at com/company1/foo/services/messaging/tasks/company2/company2SrTask.doWork(Bytecode PC:292)

10 at com/company1/foo/services/messaging/service/XmlTextMessageService.onMessage(Bytecode PC:121)

9 at com/company1/foo/services/messaging/ejb/**TheConsumerBean.onMessage**(Bytecode PC:108) <<< Enter onMessage() !!!

8 at **com/ibm/ejs/container/MessageEndpointHandler.invokeMdbMethod**(MessageEndpointHandler.java:1014(Compiled Code))

7 at **com/ibm/ejs/container/MessageEndpointHandler.invoke**(MessageEndpointHandler.java:747(Compiled Code))

6 at \$Proxy7.**onMessage**(Bytecode PC:18)

5 at com/ibm/ws/sib/api/jmsra/impl/JmsJcaEndpointInvokerImpl.invokeEndpoint(JmsJcaEndpointInvokerImpl.java:201)

4 at com/ibm/ws/sib/ra/inbound/impl/**SibRaDispatcher.dispatch**(SibRaDispatcher.java:768(Compiled Code))

3 at com/ibm/ws/sib/ra/inbound/impl/SibRaSingleProcessListener\$SibRaWork.run(SibRaSingleProcessListener.java:584)

2 at com/ibm/ejs/j2c/work/WorkProxy.run(WorkProxy.java:419(Compiled Code))

1 at com/ibm/ws/util/ThreadPool\$Worker.run(ThreadPool.java:1473(Compiled Code))

Read from bottom to top!

Javacore Analysis:

We start from the bottom and read up:

- On line 4 notice the **SibRaDispatcher**. This is the SIB Resource Adapter.
- On line 6 we see the first mention of **onMessage()**.
- On lines 7 and 8 we see the EJB Container prepare to call onMessage() of the customer MDB.
- On line 9 we see that **onMessage()** is called. The method is in a customer class (MDB) named **TheConsumerBean**. *We are now in customer code!*
- Notice that from lines 10 to 13 the package name is “com/company1” and we are no longer in “com/ibm” code. This means we are now in code from the company1 company (company1.com).
- Notice from lines 14-22 the package name is “com/company3”. This again is not IBM code.
- Notice that on line 19 “**jdbc**” appears in the call stack. JDBC stands for **J**ava **D**atabase **C**onnectivity. This means the customer code is now connecting to a database.
- In line 22 the code is now in **popConnection()**, which is non-IBM code. *This is where the hang occurs.*

If at least one other javacore shows this same thread in the exact same code (if they show exactly the same stack with popConnection() at the top), this MDB (TheConsumerBean**) likely isn't consuming messages. The MDB is hung in **popConnection()** (at the top of the call stack). The MDB is probably waiting for a response from the database, so interaction with the database, not the MDB, is the real problem. *The connection to the database should be investigated.***

Message States

The following is a list of ALL possible message states, although the states we have already discussed are the most common. The most common states are in **red**.

- **Available** The message is available for consumption.
- **Locked** The message is currently unavailable. The message is in this state temporarily, possibly because it is being consumed by a non-transacted consumer.
- **Unlocked** There is no message listener running. The match criteria of all the consumers for this destination do not match the message. All consumers for this destination are pointing at a different partition of the destination in a WLM environment. All consumers are maxed out processing messages and can't accept new ones (this would normally happened when we have a mix of **locked** and **unlocked** messages).
- **Remote lock** The message is currently locked to a consumer attached to another, remote, messaging engine in the bus. The message will remain locked until the remote messaging engine responds with a decision on the message. If the remote messaging engine is stopped, the message will remain locked until the messaging engine is restarted. A corresponding message request for a "known remote queue point" will identify the remote messaging engine that is making the request.



Message States

- **Removing** The message is currently being removed under a transaction. The message will be in this state until the transaction commits or rolls back. If this state persists, investigate the state of the transaction identified by Transaction ID.
- **Committing** The message is currently being added under a transaction. The message will be in this state until the transaction commits or rolls back. If this state persists, investigate the state of the transaction identified by Transaction ID.
- **Pending retry** The message is currently unavailable before being eligible for a retry. This might be because a message-driven bean is configured to delay failing message retries.
- **Blocked** This message is currently unavailable because the message point is blocked by the first message on the queue. The first message has reached its maximum failed delivery limit but no exception destination is configured. Identify the first message and resolve the problem that is preventing it from being consumed.



Message States

One additional, though rare, message state that you may see is **Pending acknowledgement**. This state can occur on a Remote Publication Point when a message is sent between 2 Messaging Engines. This state means that the sending Messaging Engine is waiting on the receiving Messaging Engine for acknowledgement that the message was received.

If this state occurs make sure all messaging engines are Started and operating correctly. If necessary you may be able to solve the problem by stopping and restarting the Messaging Engines, one at a time, in the messaging cluster.



Summary

This presentation is designed to help you understand and investigate why messages are stuck or accumulating on a destination. A basic understanding of how MDBs work was presented, including an overview of `onMessage()` and the importance of understanding what `onMessage()` does. This was followed by a discussion of message states and how to interpret them.

Lastly, Javacores and their importance was discussed, including an example that shows a typical hung MDB scenario and how to interpret the call stack.

It is hoped that this presentation will help you deal with stuck messages in the future!



Additional WebSphere Product Resources

- Discover the latest trends in WebSphere Technology and implementation, participate in technically-focused briefings, webcasts and podcasts at:
<http://www.ibm.com/developerworks/websphere/community/>
- Learn about other upcoming webcasts, conferences and events:
http://www.ibm.com/software/websphere/events_1.html
- Join the Global WebSphere User Group Community: <http://www.websphere.org>
- Access key product show-me demos and tutorials by visiting IBM Education Assistant:
<http://www.ibm.com/software/info/education/assistant>
- View a Flash replay with step-by-step instructions for using the Electronic Service Request (ESR) tool for submitting problems electronically:
<http://www.ibm.com/software/websphere/support/d2w.html>
- Sign up to receive weekly technical My support emails:
<http://www.ibm.com/software/support/einfo.html>



Questions and Answers

