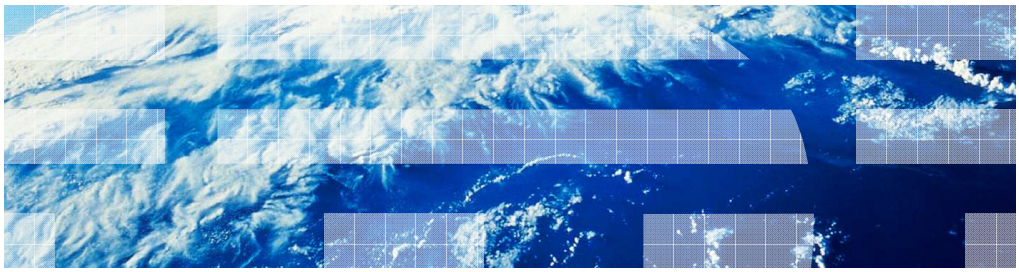
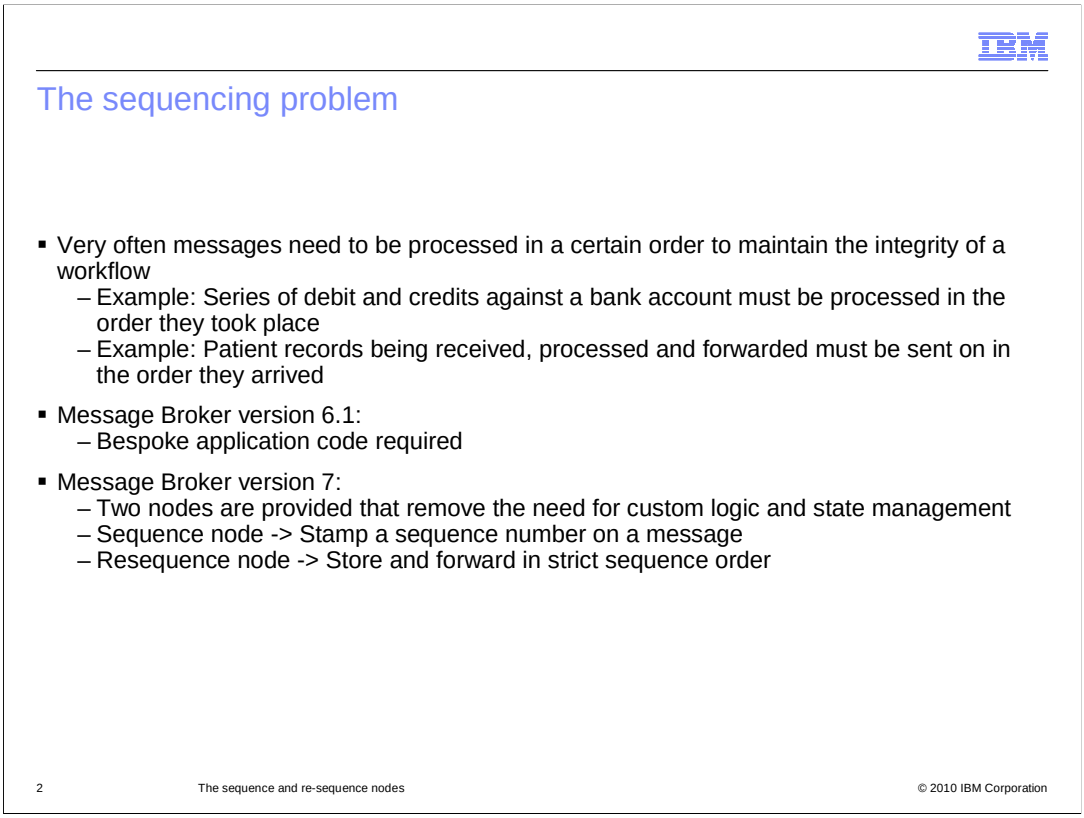
WebSphere Message Broker Version 7

The sequence and re-sequence nodes



WebSphere. software

This session discusses the new sequence and re-sequence nodes, introduced in Message Broker version 7.

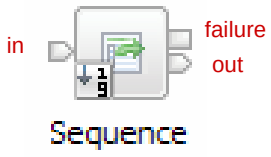


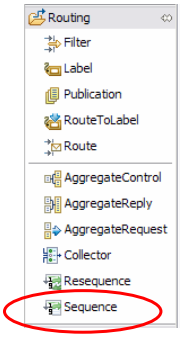
A common application problem is to generate or process a series of messages, where the order of the messages is important. It is often found that messages need to be processed in order, where the order of the messages is defined and controlled by a key or index held within the data of the message payload.

In Message Broker version 6, this type of processing required you to write your own application code to manage the state of the message processing, and to enforce the order of the messages.

In Message Broker version 7, the sequence and re-sequence nodes have been introduced to perform this processing using standard product capability.

Sequence node





- Allocates a sequence number to each received message
- Propagates the message stamped with the sequence number
- Multiple 'sequence groups' handled independently, in parallel
- Does not allocate next sequence number in group until current message in group has finished processing, to allow sequencing across multiple threads
- Sequence group state preserved across broker restarts
- Queues used:
  - SYSTEM.BROKER.SEQ.GROUP
  - SYSTEM.BROKER.SEQ.NUMBER

3 The sequence and re-sequence nodes

The sequence node is a new node that receives an input message, and allocates a monotomically increasing sequence number from a user-defined start point. It then stores the number in a location of the user's choosing in the message assembly. It does this for each message until the sequence ends, which is determined by a user-defined condition.

Messages arriving can be divided into independent 'sequence groups' based on an identifier in the message. Each group has its own sequence number and is handled completely independently.

Queues are used to store the current sequence number for each active sequence group. The queue names are shown on this slide.

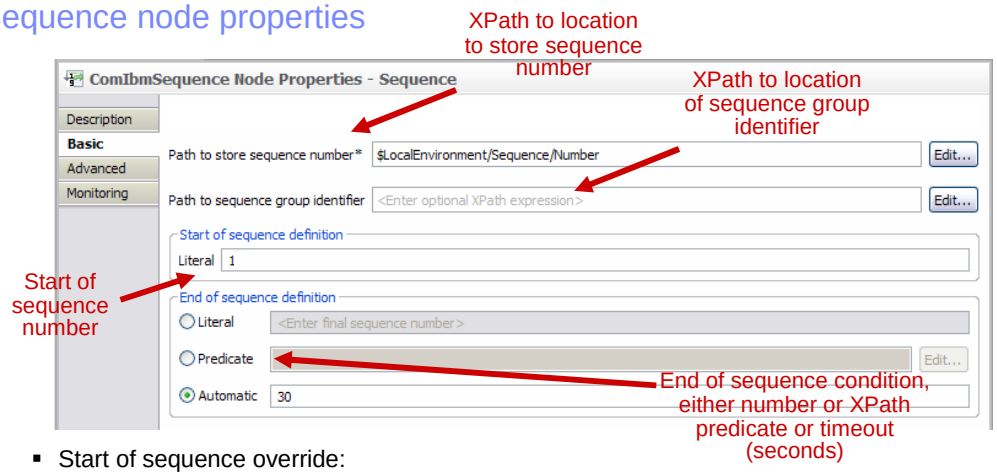
Sequence numbers can be persisted so that the sequence survives across queue manager restarts.

The node guarantees that the next sequence number for a given group will not be allocated until the current message for that group has been either committed or rolled back.

It is the use of the WebSphere MQ queue that provides this behavior. If another thread is processing a message in the same group, the thread will attempt to obtain the state message for the group, and will block if it is locked by another thread.

This ensures that sequencing is maintained for the group when there are multiple threads in the message flow.

Sequence node properties



- Start of sequence override:
  - **`LocalEnvironment.Sequence.StartOfSequenceNumber`**
- Other Local Environment variables:
  - **`LocalEnvironment.Sequence.Number`**
  - **`LocalEnvironment.Sequence.Group`**
  - **`LocalEnvironment.Sequence.Start`**
  - **`LocalEnvironment.Sequence.End`**

4 The sequence and re-sequence nodes

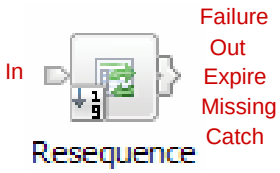
The start of the sequence must be specified by a literal number, but the value can be overridden by local environment, as shown on this slide.

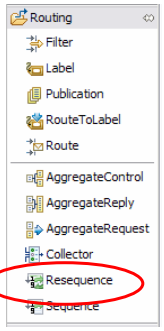
The end of the sequence is specified by either a literal number, an X-Path predicate that evaluates to true or false, or the time to wait for the next message.

The sequence number and sequence group identifier can be specified in two places. They can be specified in the node properties, as shown on this screen capture, using an X-Path expression. Alternatively, they can be specified in the local environment.

The Advanced tab has properties to control persistence and to specify a configurable service.

Resequence node





- Examines a sequence number in each received message
- Only propagates messages in sequence number order
- Out-of-sequence messages are stored until the missing messages arrive, or a missing message timeout occurs
- Out, Expire and Missing terminals to handle different use cases
- Multiple 'sequence groups' handled independently, in parallel
- Sequence group state preserved across broker restarts
- Node is a transaction break (like Collector node)
- Queues used:
  - SYSTEM.BROKER.EDA.EVENTS
  - SYSTEM.BROKER.EDA.COLLECTIONS

5 The sequence and re-sequence nodes

The re-sequence node is a new node that receives an input message, and checks a sequence number in a location of the user's choosing in the message assembly. It only propagates the message to the out terminal if it is the next message in the sequence.

If a message arrives that is out-of-sequence, the node stores the message until the missing messages arrive. For example, message number one arrives and is propagated, number three arrives and is stored, and number four arrives and is stored. Finally, number two arrives and is propagated, followed by number three and then number four.

A missing message timeout can be specified to prevent the node waiting forever for missing messages. When the timer expires, all stored messages are propagated in order, one at a time, to the expire terminal. If the missing messages eventually arrive they are propagated to the missing terminal. Any other messages that arrive continue to be propagated to the expire terminal.

Once a sequence has started, which is determined by a user-defined condition, re-sequence processing continues for each message until the sequence ends, which is also determined by a user-defined condition.

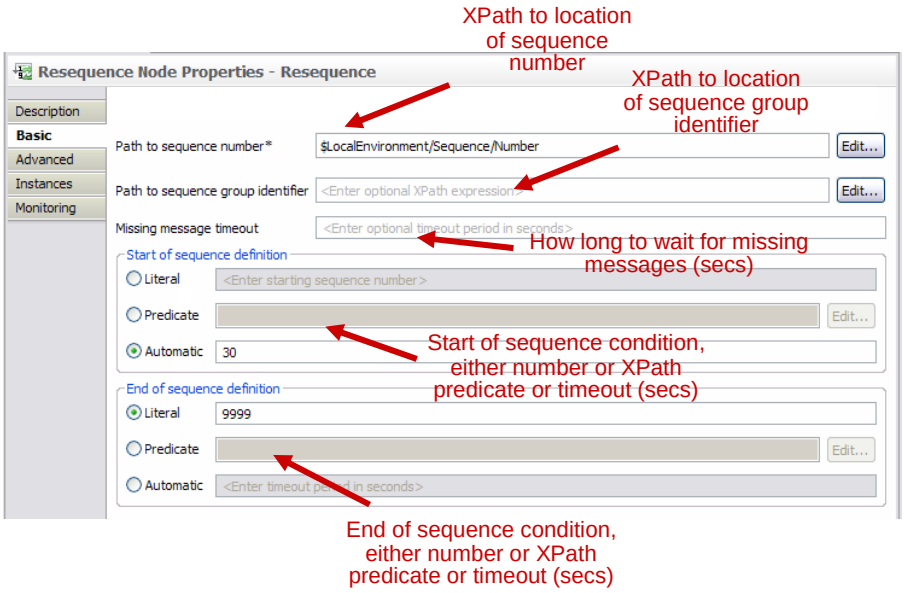
Messages arriving can be divided into independent 'sequence groups' based on an identifier in the message. Each group has its own sequence number and is handled completely independently, enabling parallel processing of groups.

WebSphere MQ queues are used to store the sequence group states, using the queue names shown on this slide. These are the same two queues as used by the collector node.

Sequence group state can be persisted so that the sequence survives across queue manager restarts.

The re-sequence node employs the same architecture as the collector node. It is therefore a transaction break, and all propagated messages are in a new unit of work. The node therefore has a catch terminal to catch any downstream exceptions. It has the same restrictions as collector node, in that the environment, local environment and exception list trees belonging to the received messages are not preserved.

Resequence node - properties

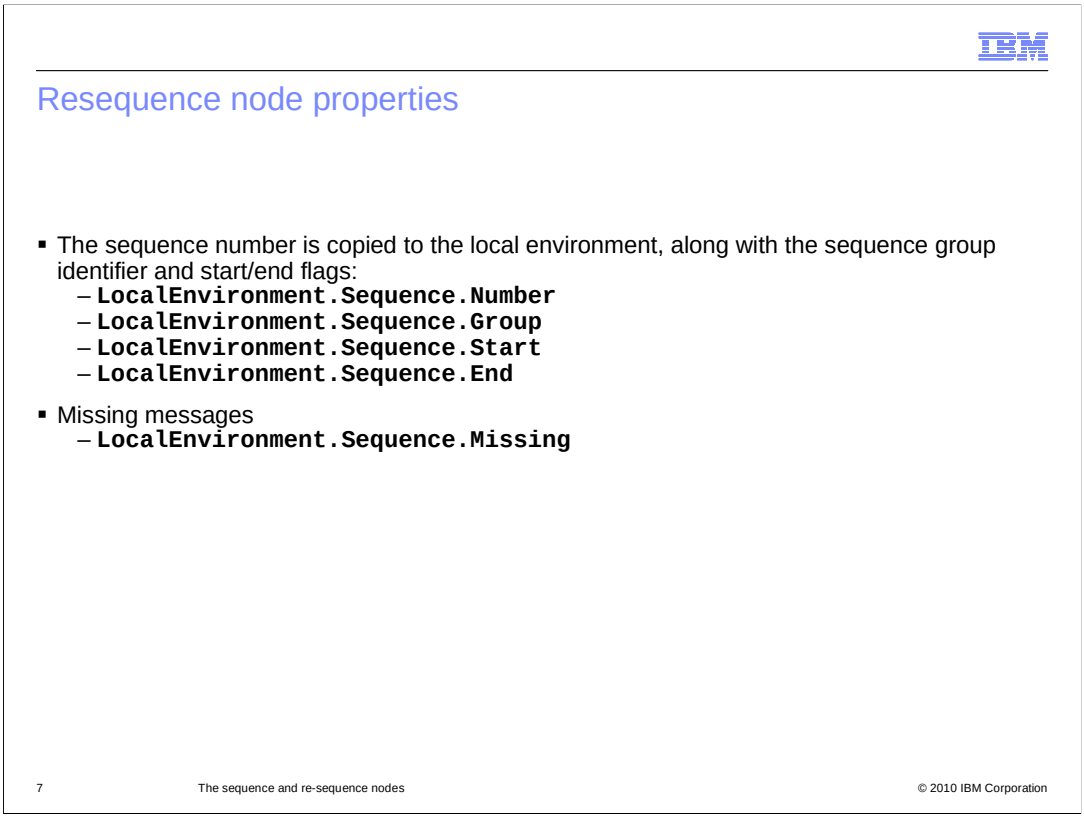


The start of the sequence is specified by either a literal number, an XPath predicate that evaluates to true or false, or a time to wait. For the latter, the node stores all messages that arrive and when the timer expires, it uses the lowest number as the start of the sequence.

The end of the sequence is specified by either a literal number, an XPath predicate that evaluates to true or false, or the time to wait for the next message.

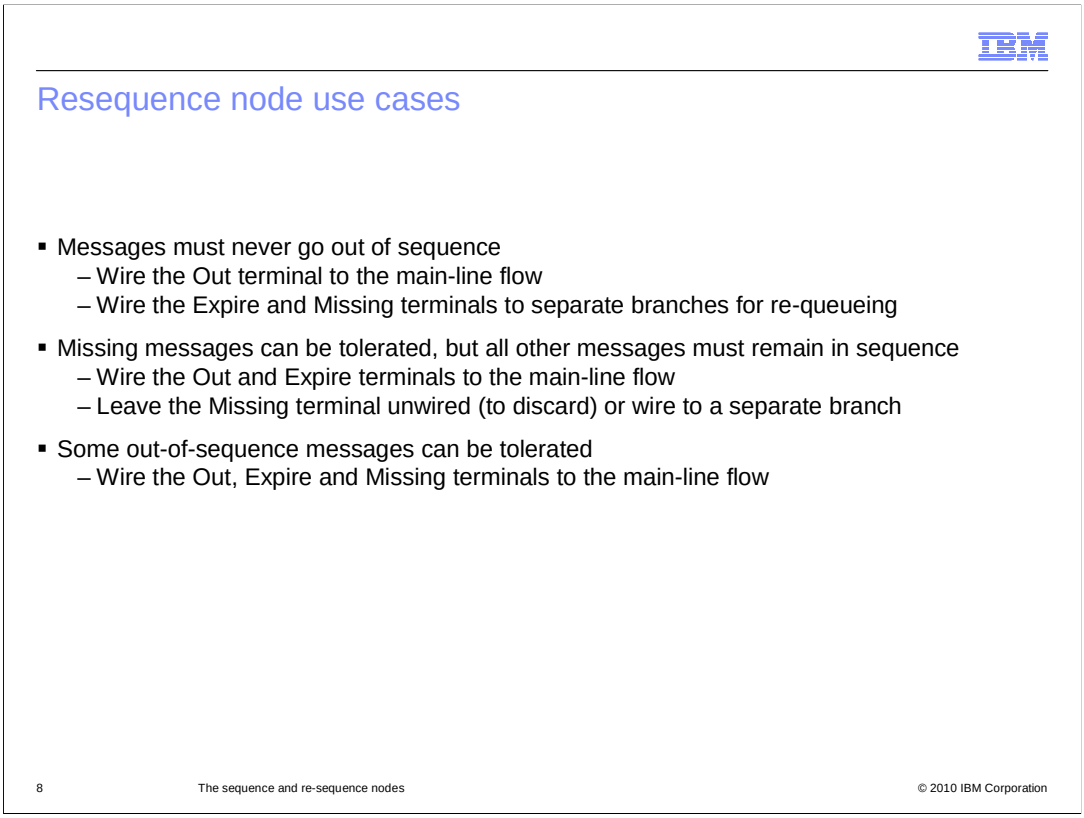
The Advanced tab has properties to control persistence and to specify a configurable service.

The Instances tab allows the re-sequence node to request extra threads from the message flow's thread pool, or to allocate a separate thread pool for its own use.



The sequence number is copied to the local environment, along with the sequence group identifier and the start and end flags.

If there was a gap caused by missing messages between the previously propagated message and the current message, the sequence number of the missing messages are stored in the local environment, one entry per missing number.



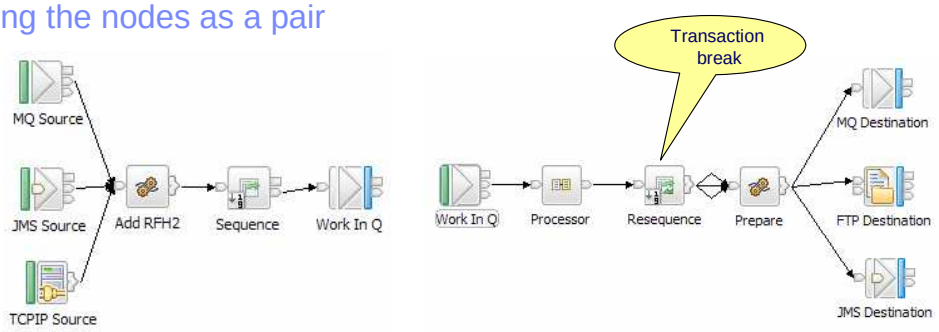
The following use cases are typical for the sequence and re-sequence nodes.

First, messages must never go out of sequence. If a message goes missing, route all subsequent messages, in sequence, if possible, to a temporary queue after a timeout period. To configure this behavior, wire the out terminal to the main-line flow and the expire and missing terminals to separate branches for re-queuing.

Secondly, missing messages can be tolerated, but all other messages must remain in sequence. If a message goes missing, skip over it and continue processing the rest of the sequence. If the missing message eventually arrives, either discard it, or process it separately from the main-line processing. To configure this behavior, wire the out and expire terminals to the main-line flow and leave the missing terminal unwired to discard the message, or wire it to a separate branch.

Finally, some out-of-sequence messages can be tolerated. The flow needs to process all the messages, preferably in sequential order, but in the interests of not holding up the flow for too long, some messages can be skipped and processed later. To configure this behavior, wire the out, expire and missing terminals all into the main-line flow.

Using the nodes as a pair



- Scenario: Messages without a sequence number arrive from multiple sources
  - Their order must be preserved when they are sent to their destinations
  - Intermediate processing depends on size and type of message and must be highly parallel to achieve required throughput
- Solution: Use a sequence group per source
  - Use a receiver flow with a Sequence node to establish the sequence
  - Add a Resequence node to the processing flow to preserve the sequence
  - Ensure sequence start and end conditions tie up

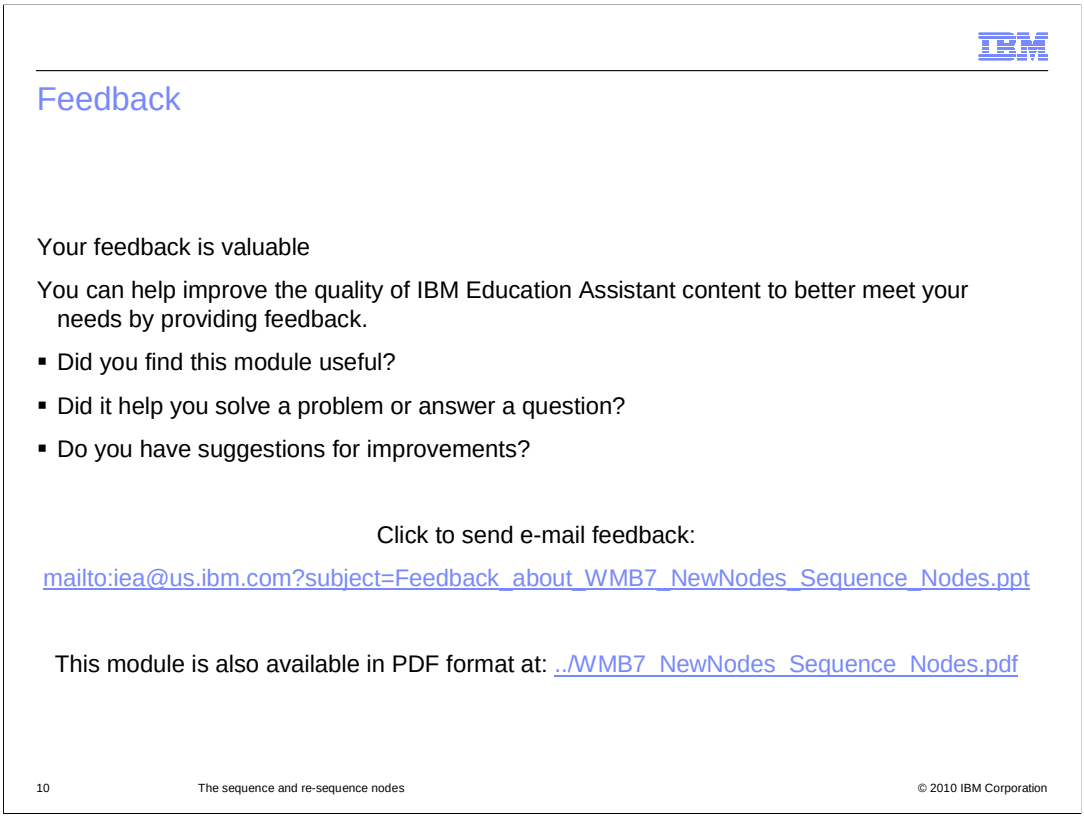
9 The sequence and re-sequence nodes

In this scenario, the sequence node is used to establish the sequence based on order of arrival. The re-sequence node is used to re-establish the sequence in case it got out-of-step, because of the length of time taken during the main processing phase of the flow.

As a reminder, the re-sequence node is a transaction break; it takes three units of work to process a message in this scenario. Also, you can use separate thread pools for the left side and right side of the processing flow to enable the required throughput, without starving the right side.

In the screen capture, the re-sequence node out, expire and missing terminals are all wired to the next node in the flow for convenience only.

The two nodes must detect the start and end of the sequence in step. One way to do this is for the re-sequence node to use the LocalEnvironment start/end flags set by the sequence node.



You can help improve the quality of IBM Education Assistant content by providing feedback.

Trademarks, disclaimer, and copyright information