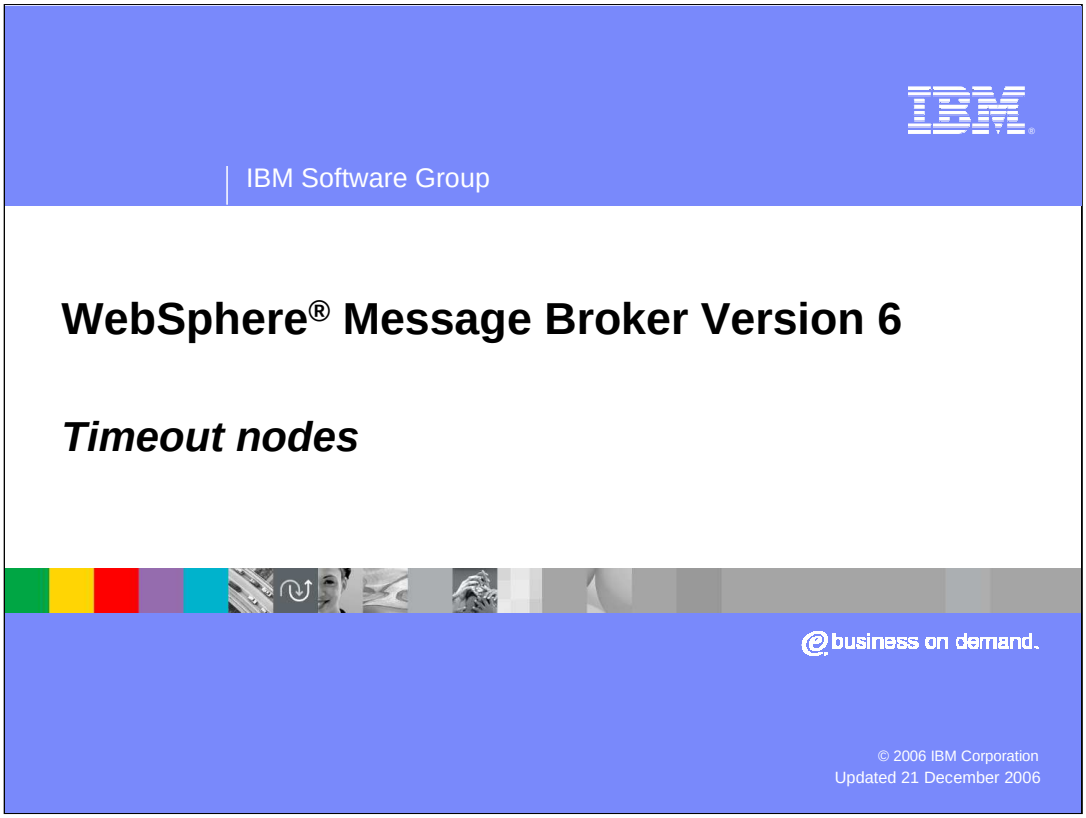


This session discusses the new timeout nodes, introduced in WebSphere Message Broker Version 6.

Agenda

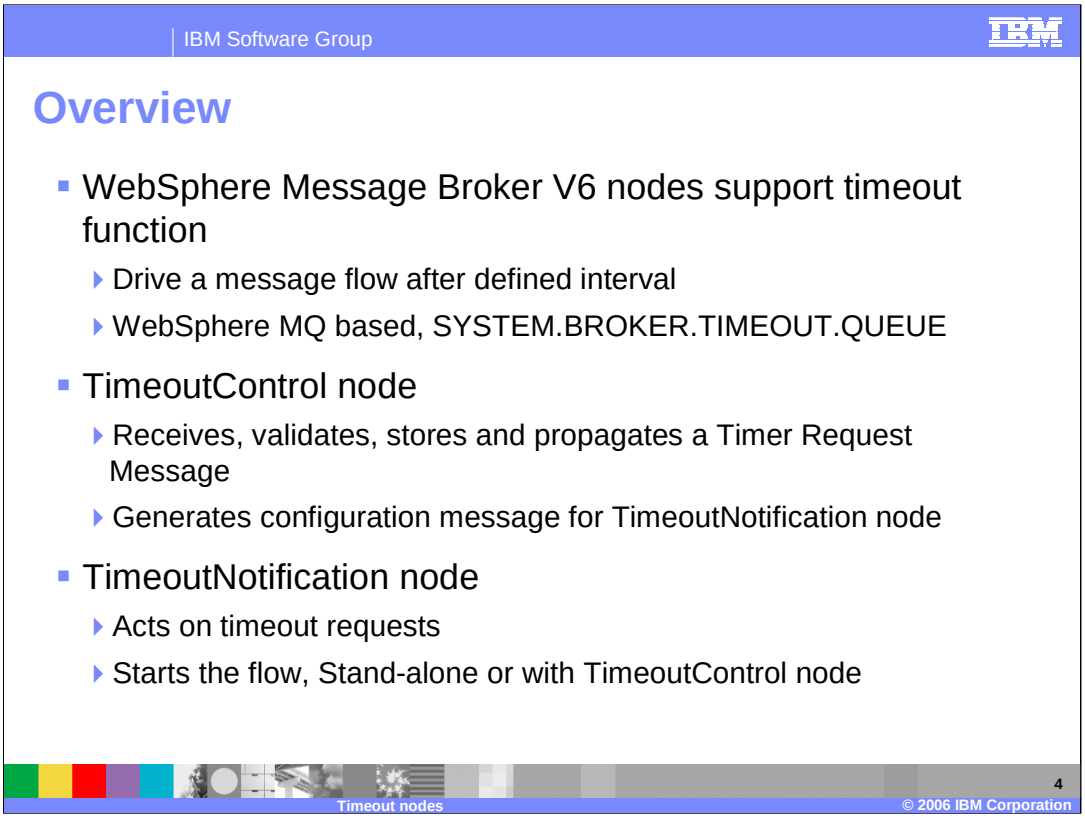
- Overview
- Implementation and configuration
- Summary

The agenda for this presentation includes an overview and a discussion of implementation and configuration of timeout nodes.

Section

Overview

This section covers the overview.



WebSphere Broker V6 provides two new nodes to support a basic timeout function:

TimeoutControl node

TimeoutNotification node

Use these nodes in message flows to process timeout requests or to generate timeout notifications at specified intervals. These can be used to drive message flows automatically at regular intervals - these are known as *automatic timeouts*. Or you can provide *controlled timeouts* to applications on a per-message basis.

The timeout nodes use the WebSphere MQ queue, SYSTEM.BROKER.TIMEOUT.QUEUE, which holds the active timeout requests for a running broker.

The TimeoutControl node receives an input message that contains a timeout request. The node validates the request, stores the message, and propagates the message to the next node in the message flow.

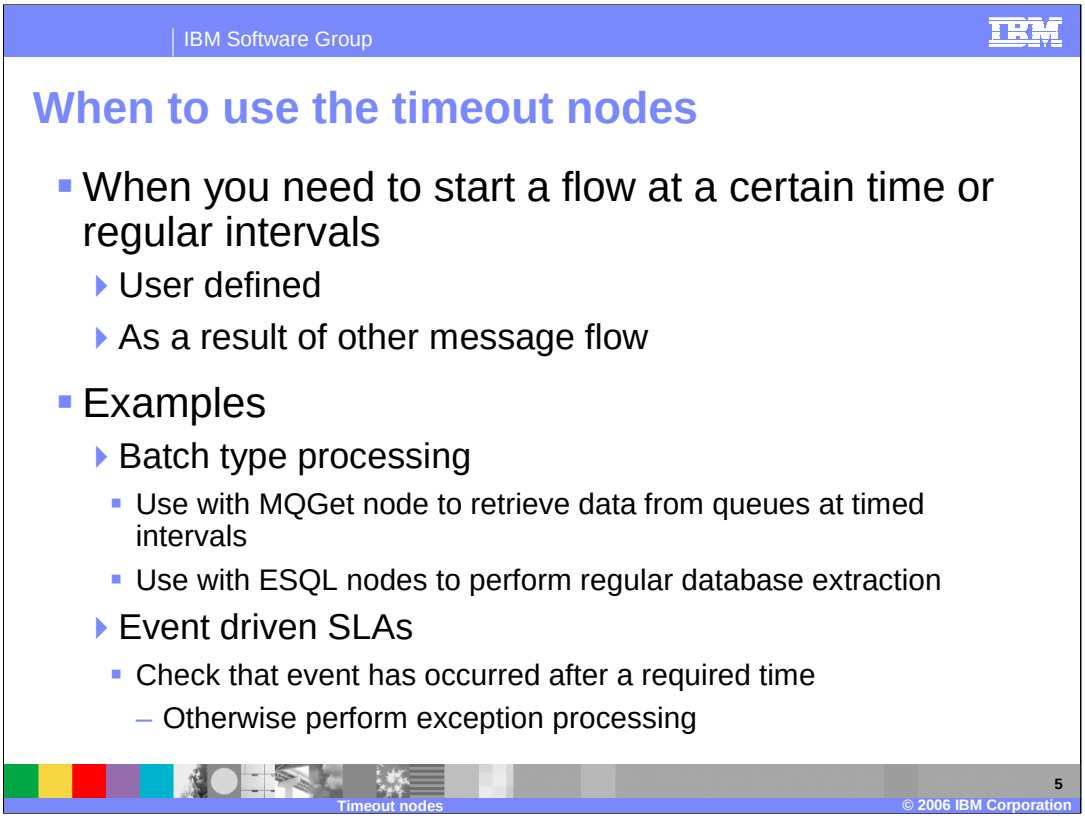
The TimeoutNotification node is an input node which reads from the SYSTEM.BROKER.TIMEOUT.QUEUE any timeout requests that relate to it, processes them and propagates the messages as required.

The TimeoutNotification node can be used in one of two ways:

1. Paired with one or more TimeoutControl nodes.
2. Stand-alone. Generated messages are propagated to the next node in the message flow at timed intervals that are specified in the configuration of this node.

.

.



Use the TimeoutControl and TimeoutNotification nodes in message flows to process timeout requests or to generate timeout notifications at specified intervals.

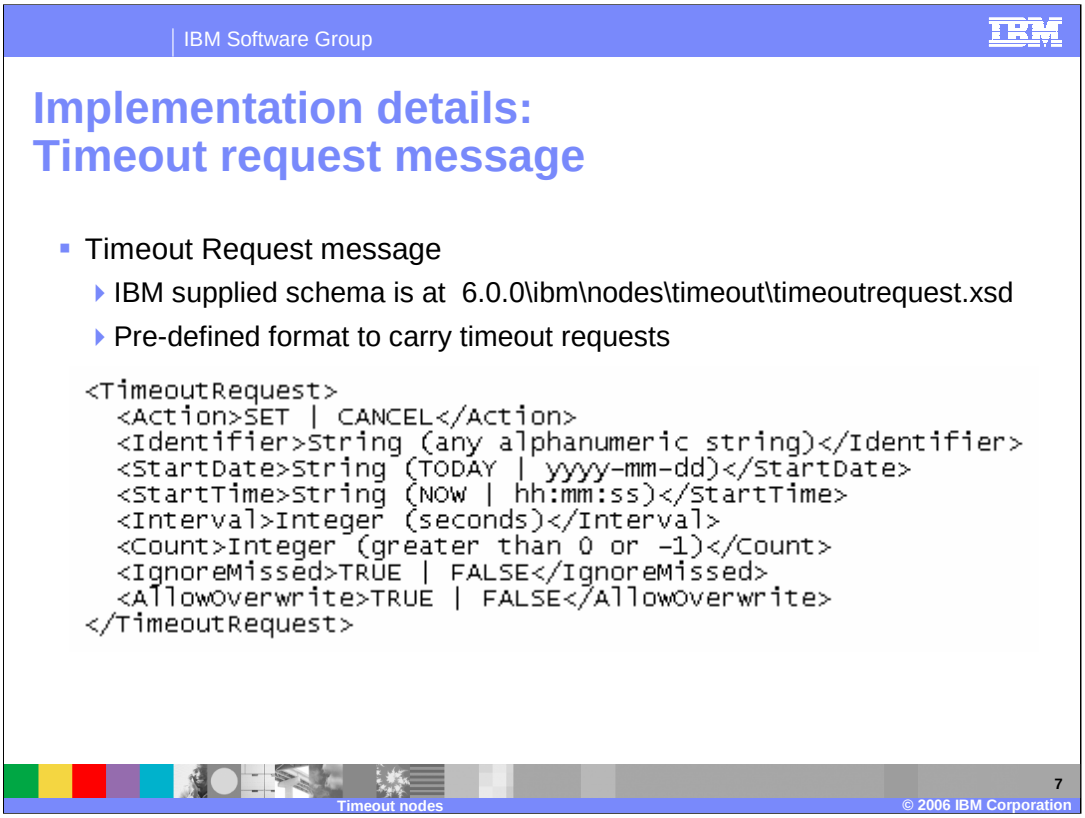
Here are examples of when you might want to use the timeout nodes in a message flow:

- You need to run a batch job every day at midnight.
- You want information about currency exchange rates to be sent to banks at hourly intervals.
- You want to confirm that important transactions are processed within a certain time period and perform some other specified actions to warn when a transaction has not been processed in that time period.

Section

Implementation and configuration

This section covers implementing and configuring Timeout nodes.



The XML format of a timeout request message is show on this slide. Any other format that is supported by an installed parser can be used instead of XML.

A Timeout Request is either SET or CANCELed. If CANCEL, the Identifier is required and must match the Identifier of the TimeoutRequest that is to be cancelled. TimeoutControl nodes and TimeoutNotification nodes have an individual identifier that is unique within the broker. Any requests that those nodes receive are indexed by the timeout node identifier and their own unique identifier embedded in the timeout request. If you want to set a timeout request you have to pass in an identifier; you can then pass this identifier in a subsequent message requesting a cancellation of the original request.

The “StartDate” and “StartTime” values are self explanatory; the default values are ‘Today’ and ‘Now’.

‘Count’ is the number of times a timeout request is going to start. So, you could start a timeout request immediately and request that it is initiated ten times. Those ten times will be separated by ‘Interval’ number of seconds. The default of ‘Count’ is ‘1’ and the default of ‘Interval’ is ‘0’, which means that if you put in a default message with just a ‘SET’ and an ‘Identifier’, you will get a message immediately straight from the timeout node, since it uses the defaults of ‘Today’, ‘Now’, ‘0’ and ‘1’.

‘IgnoreMissed’ controls whether timeouts that occur while either the broker or the timeout notification flow is stopped, are processed the next time that the broker or timeout notification flow is started. The default value is TRUE which means that missed timeouts are ignored by the TimeoutNotification node when the broker or message flow is started. If this value is set to FALSE then the missed timeouts are all immediately processed by the TimeoutNotification node when the flow is started.

‘AllowOverwrite’ controls whether subsequent timeout requests with a matching *Identifier* can overwrite this timeout request. This is used to allow a SET request with the same Identifier as a running timeout application to overwrite the original. The default value is TRUE; FALSE throws an exception for any attempt to overwrite a running timeout message.

Details – TimeoutControl node

- TimeoutControl node
- Validates incoming requests
  - Timeout requests embedded in incoming message
- Stores validated requests
  - Stores all or part of incoming message to timeout queue
- Cancels stored requests
  - Deletes stored requests from timeout queue
- Passes input message through unchanged

The TimeoutControl node is a standard data flow node; it receives incoming messages and validates the Timeout Requests embedded in them. If they are validated, it then stores them to the SYSTEM.BROKER.TIMEOUT.QUEUE. It can store all or part of the incoming message. The default is to store the entire incoming message so that the message that the TimeoutNotification node propagates is what is generated by the TimeoutControl node. Alternatively, if you have a very large message, you can store just part of this message.

The TimeoutControl node can also cancel stored requests. If a cancellation request is received, it just deletes the message from Timeout Queue. The input message passes through unchanged; if you connect the output terminal, the message is simply propagated to the next node in the flow.

If the TimeoutControl node gets a failure during processing, it throws an exception. Examples are:

- if the request could not be validated
- if the Identifier matches an existing request
- If the Timeout Request has been badly formatted

Details – TimeoutControl node

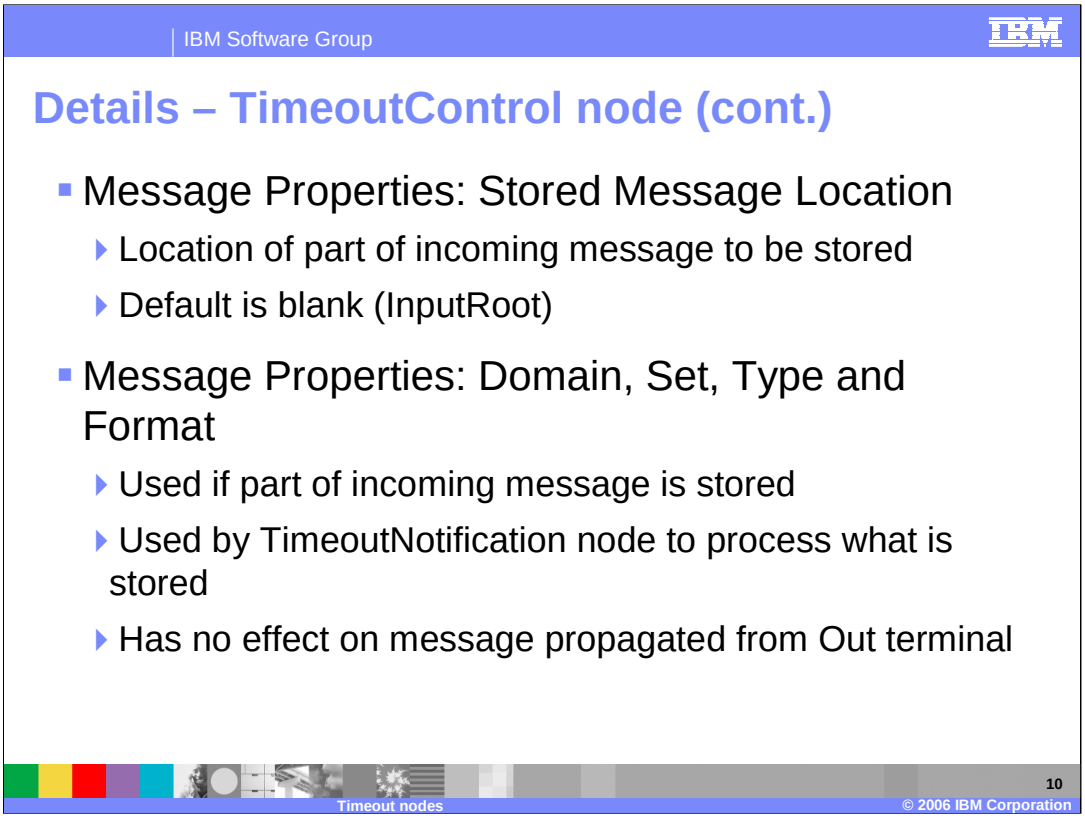
- Basic Properties: Unique Identifier
  - Unique name within a broker
  - Used to associate with exactly one TimeoutNotification node
- Basic Properties: Request Location
  - Location of timeout request in incoming message
- Basic Properties: Request Persistence
  - Controls persistence of messages stored on timeout queue
  - Yes, No or Automatic (as incoming message)

When you have put an instance of the TimeoutControl node into a message flow, you can configure its properties. Unique Identifier is the only mandatory property. It does not have a default value. Its value must be unique within the broker. The equivalent property of the TimeoutNotification node with which it is paired must have the same value. The maximum length of this identifier is 12 characters.

Each Timeout Request has a unique identifier which associates it with exactly one TimeoutNotification node. However, multiple TimeoutControl nodes can be associated with one TimeoutNotification node; several different TimeoutControl nodes can store messages to be processed by a single TimeoutNotification node.

The Request Location property describes where to find the timeout request information. This must be a valid location in the message tree. This is validated at runtime. If no request location is specified, LocalEnvironment.TimeoutRequest is assumed. The value can be stored there using a Compute node, or you can embed it using the supplied XSD within your message tree and specify its precise location within the message tree.

The Request Persistence property determines whether an incoming timeout request survives a broker or message flow restart. The value of this property can be Automatic, Yes, or No. If the value is Automatic, the Persistence setting in the Properties folder of the incoming message is used.

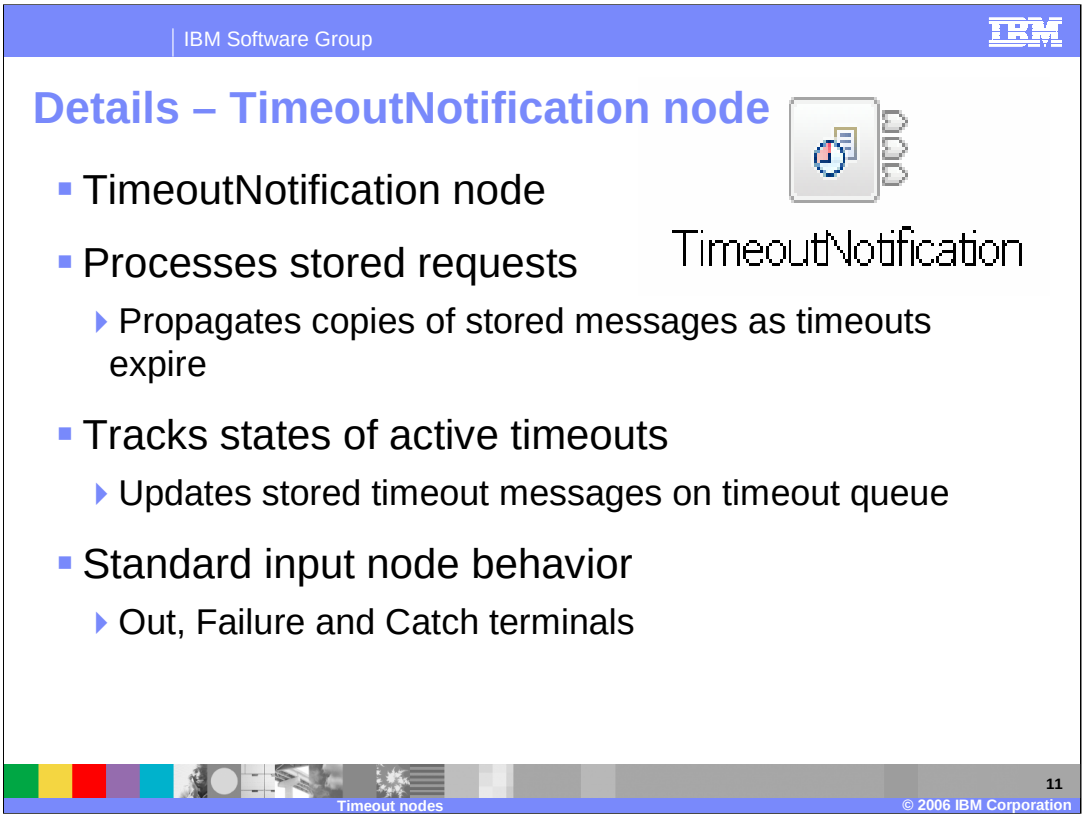


Additional TimeoutControl node properties need to be considered in the context of the TimeoutNotification node.

The ‘Stored Message Location’ provides the location of the part of the request message that you want to store for propagation by the TimeoutNotification node with which this node is paired. If you do not specify a value, the entire message is stored.

In *Message Domain*, select the name of the parser that you are using from the drop-down list. This value, and the three corresponding values in *Message Set*, *Message Type*, and *Message Format*, are used by the TimeoutNotification node with which it is paired when it rebuilds the stored message for propagation. If you have stored the entire request message (by leaving *Stored Message Location* blank), do not specify any values here. If you choose to store part of the request message, you must specify values here that reflect the stored request message fragment as if it was the entire message, which is the case when it is processed by the TimeoutNotification node.

Since a TimeoutNotification node can process multiple TimeoutControl nodes, the values specified in the TimeoutControl node are important as these properties would not necessarily be the same in the various TimeoutControl nodes.



The TimeoutNotification node is an input node, which looks very much like an MQInput node. However, the input queue for the TimeoutNotification node is the internal SYSTEM.BROKER.TIMEOUT.QUEUE. Messages on the SYSTEM.BROKER.TIMEOUT.QUEUE which have a Correlation ID (from the Unique Identifier property of the TimeoutControl node) equal to the TimeoutNotification node Unique Identifier are processed.

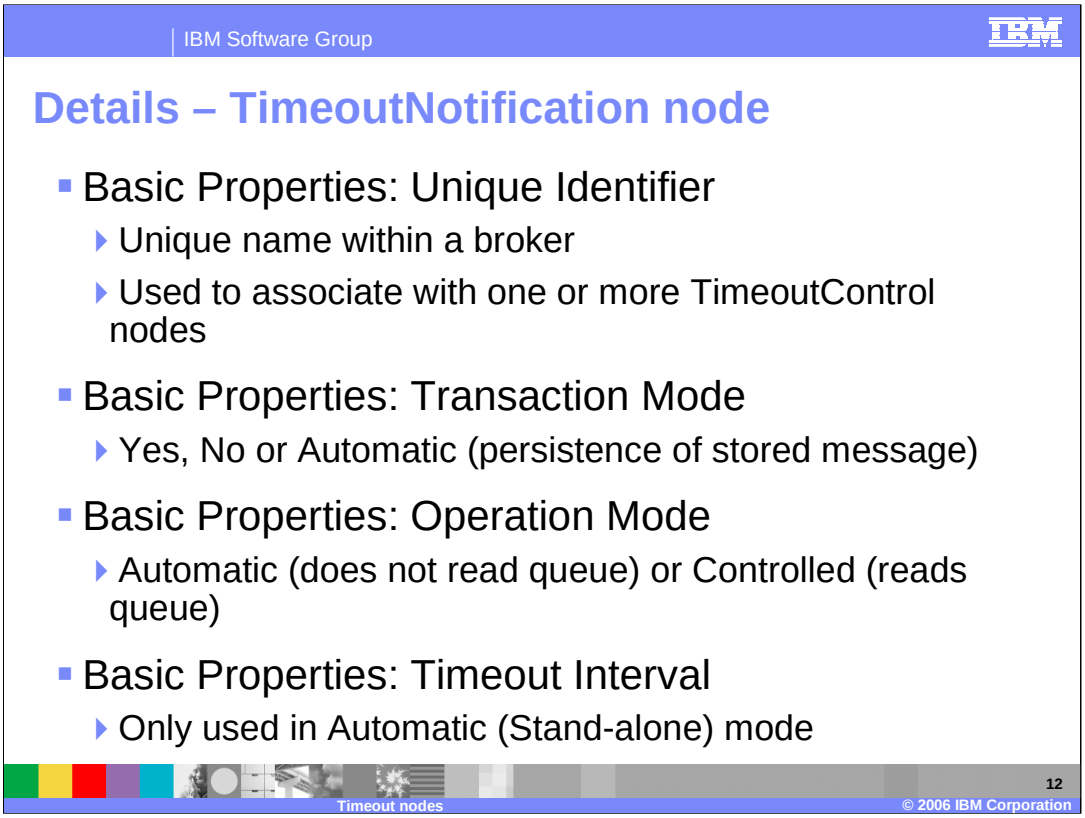
Each broker has a single SYSTEM.BROKER.TIMEOUT.QUEUE which holds all of the messages for different TimeoutNotification nodes from different TimeoutControl nodes, all with different Unique Identifiers. The Correlation ID of the stored messages matches the Unique Identifiers of the nodes.

The broker keeps track of the state of its active timeouts by keeping those messages updated as it sends out timeout events. Hence, for example, the broker knows how many events have been processed, since it updates the status message every time an event is propagated.

The Timeout Notification node has standard input node behavior with Out, Failure, Catch Terminals. If you get an Exception downstream from the TimeoutNotification node, it goes to the 'Catch' terminal. If the catch terminal hasn't been connected, it rolls back.

If you get a failure during normal processing inside the node, it goes to the Failure terminal, If you haven't connected Failure, it rolls back, exactly as the MQInput node does. The only difference in this case is that the message flow should actively cancel the message which caused the problem in the first place. If this is not done, all subsequent Timeout events will result in the same repeated error.

Note that when a TimeoutNotification node is started as a result of the broker, or the



In common with other nodes, the TimeoutNotification node has several properties.

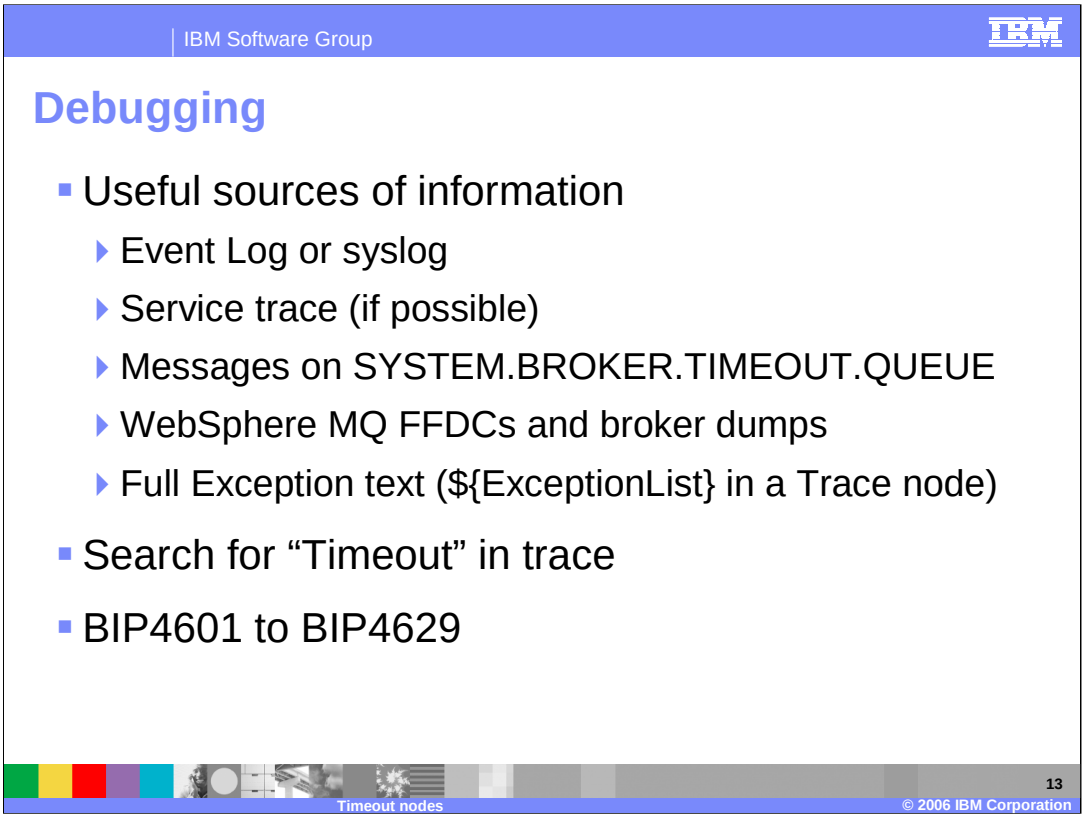
The ‘Unique Identifier’ allows you to associate it with one or more TimeoutControl nodes.

‘Transaction Mode’ is set ‘Yes’, ‘No’ or ‘Automatic’ based on the persistence of the stored message. The scope of the transaction includes any updates that are made to the stored messages on the Timeout Queue as well, so if this is done within a Sync-Point, then the whole transaction is fully coordinated.

If the ‘Operation Mode’ is set to ‘Control’, then the TimeoutNotification node is being controlled by TimeoutControl nodes; in this case, it is reading messages from the SYSTEM.BROKER.TIMEOUT.QUEUE that the TimeoutControl node is putting there.

If it is in ‘Automatic Mode’, then it is working in isolation. It simply executes, acting like a heart beat, driving the attached flow, without reading or looking at the SYSTEM.BROKER.TIMEOUT.QUEUE.

It uses the Timeout Interval to determine how often to generate an event. So the ‘Timeout Interval’ is only required in ‘Automatic’ mode.



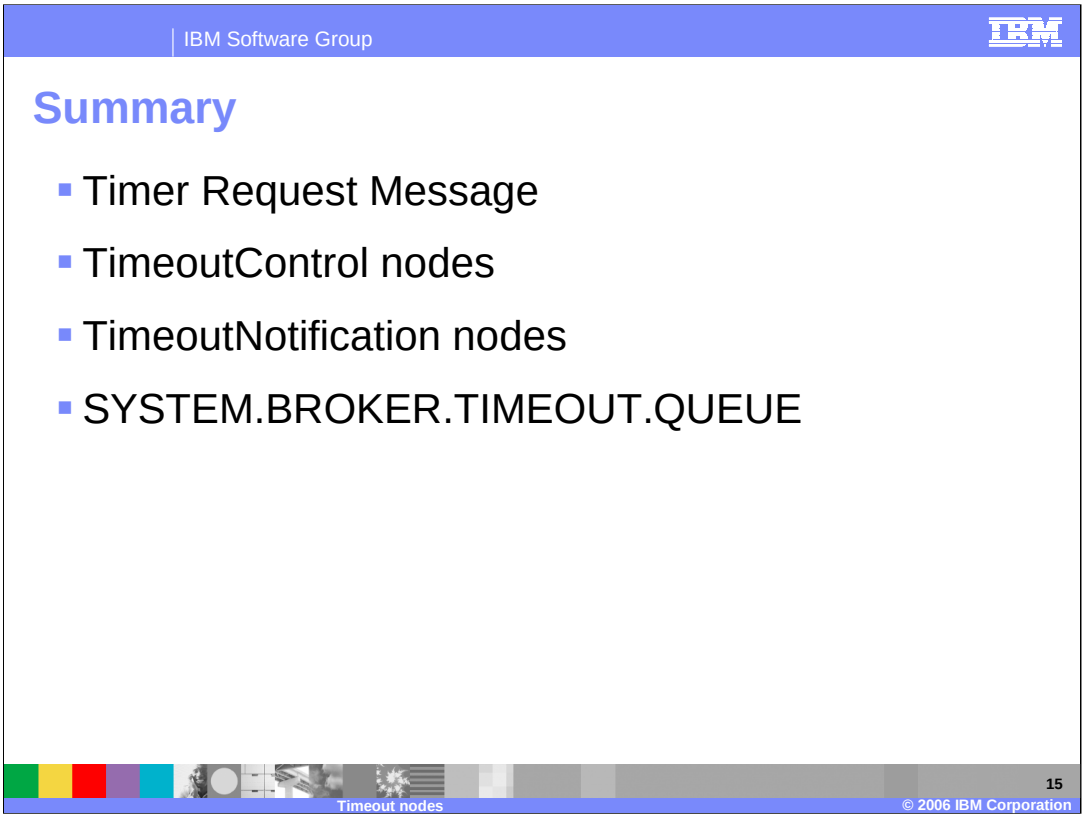
Debugging should use the same tools as for other nodes. The Event log on Windows or the service log on UNIX® will contain the usual diagnostic information. In some cases, the Service Trace may be useful as well. If you are familiar with the structure of the Timeout Messages, you can look at the messages on SYSTEM.BROKER.TIMEOUT.QUEUE. The content of these messages may provide some useful information to aid problem determination. Since the Timeout messages are stored in WebSphere MQ queues, it may also be useful to look for MQ “FFDC”s.

When reading the trace files, search for the ‘timeout’ string, which will help you to focus on the right area of the trace.

Section

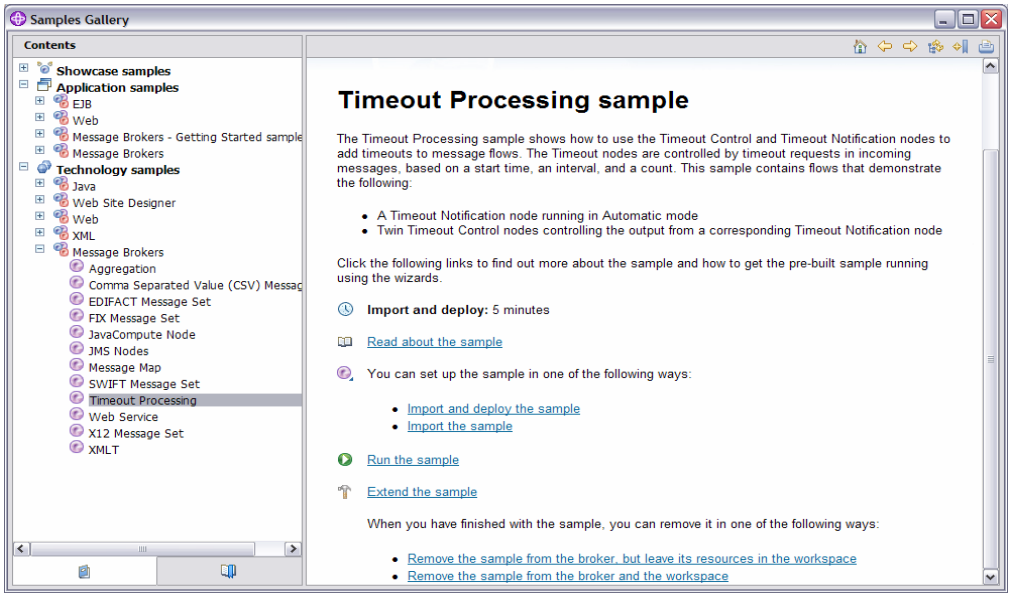
Summary and references

And, in summary…



This presentation discussed the components, capabilities and interactions of the WMB V6 timeout functions. A TimeoutControl node will read Timer Request information that has been imbedded in a message. It will validate this information and put it on the SYSTEM.BROKER.TIMEOUT.QUEUE for subsequent processing by a TimeoutNotification node. The TimeoutNotification node will read the SYSTEM.BROKER.TIMEOUT.QUEUE for messages matching its unique identifier property and propagate the message. Alternatively, the TimeoutNotification node may run stand-alone. In this case, generated messages are propagated to the next node in the message flow at timed intervals that are specified in the configuration of this node.

Samples



Samples for the timeout nodes are provided in the Samples Gallery; access through the Broker Toolkit, Samples Gallery.

References

- WebSphere Message Broker library:

http://www-306.ibm.com/software/integration/wbimessagebroker/library/

- WebSphere Message Broker Information Center:

http://publib.boulder.ibm.com/infocenter/wmbhelp/v6r0m0/index.jsp

Trademarks, copyrights, and disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM WebSphere

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

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements or changes in the products or programs described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (for example, IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.