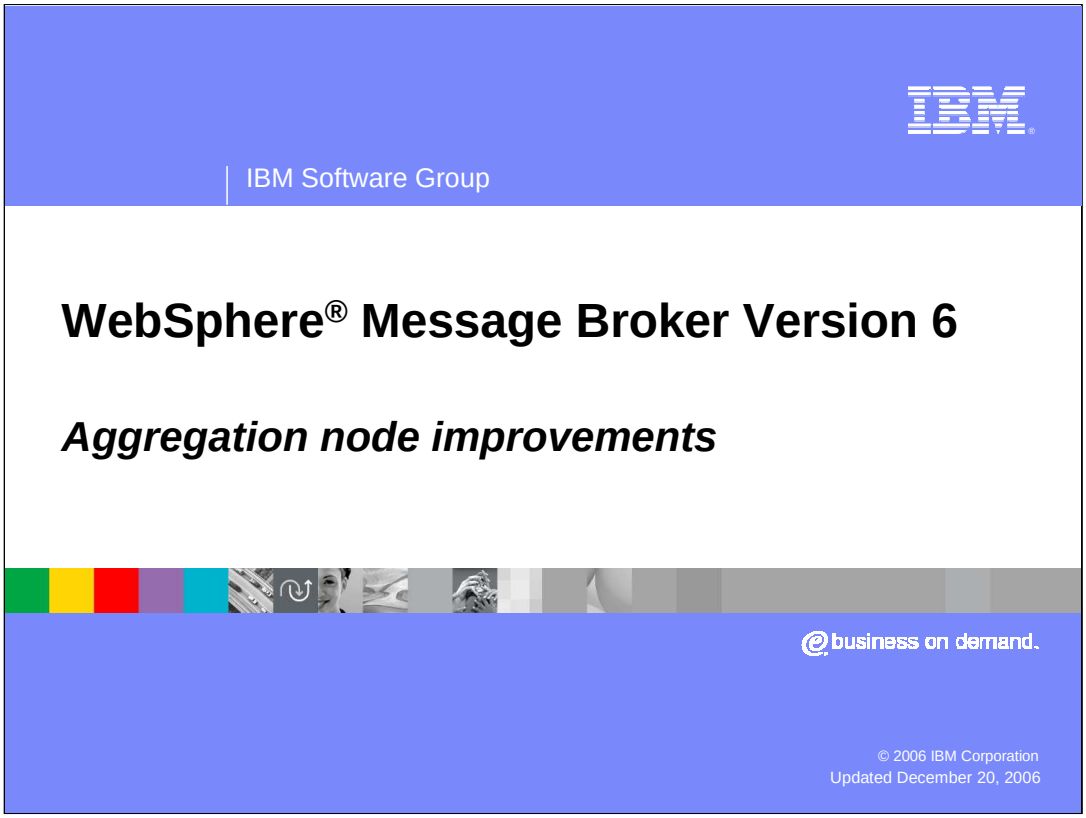


This session looks at the improvements that have been made for the Aggregation node in WebSphere Message Broker Version 6.

Agenda

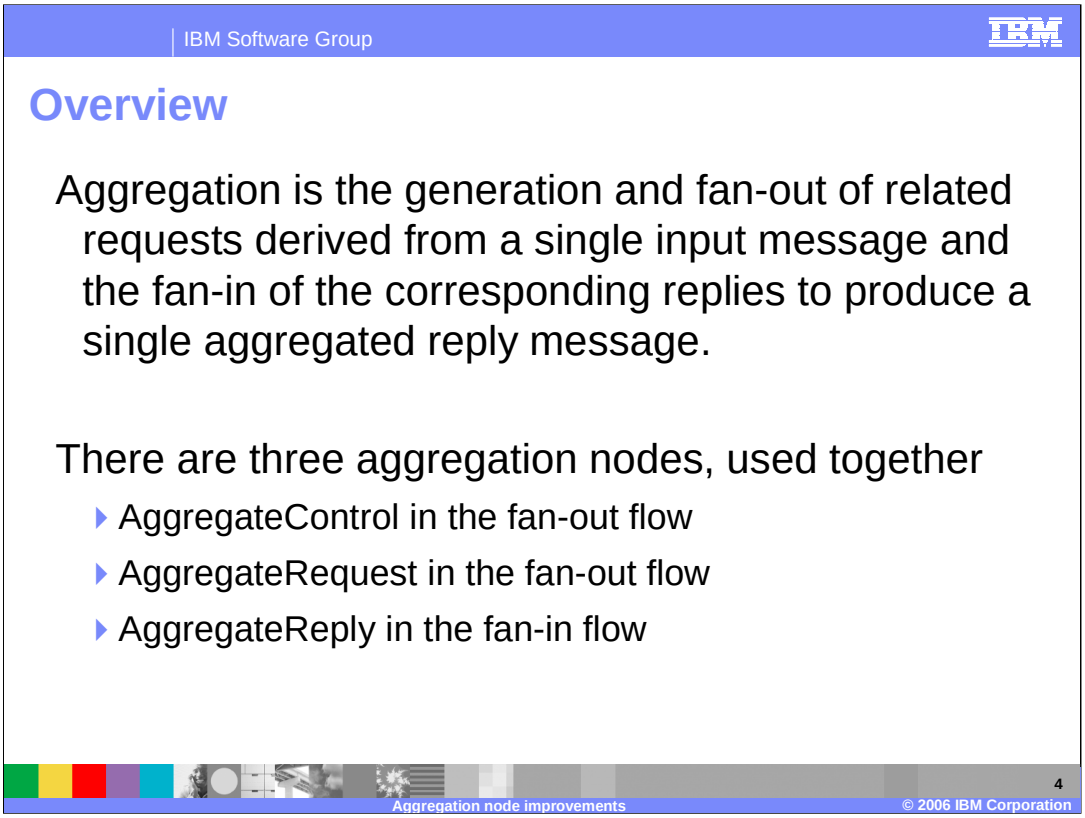
- Overview
- Configuration
- Migration
- Summary and References

This presentation will cover an overview of the Aggregation node, as well as configuration and migration.

Section

Overview

This section provides an overview of aggregation.



Aggregation is the generation and fan-out of related requests derived from a single input message and the fan-in of the corresponding replies to produce a single aggregated reply message. The initial request received by the message flow, representing a collection of related request items, is split into the appropriate number of individual requests to satisfy the subtasks of the initial request. This process is known as fan-out and is provided by a message flow that includes aggregation nodes. Replies from the subtasks are combined and merged into a single reply that is returned to the original requester (or another target application) to indicate the completion of the processing. This process is known as fan-in, and is also provided by a message flow that includes aggregation nodes.

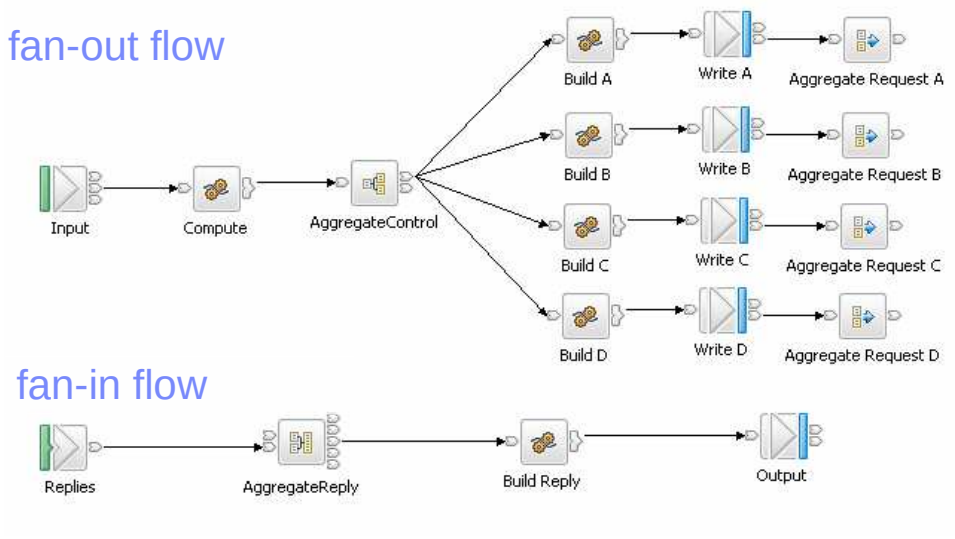
A typical scenario is a travel request message needing flight, car and hotel reservations from separate data sources returned in a single message. Aggregation also supports response timeout.

There are three aggregation nodes used together. They are:

- AggregateControl node used in the fan-out flow which controls the overall aggregation
- AggregateRequest node, one for each specific request needing a reply, in the fan-out flow
- AggregateReply node in the fan-in flow to process the responses

This is shown schematically on the next slide.

Aggregation schematic



When you include these nodes in your message flows, the multiple fan-out requests are issued in parallel from within a message flow. This is in contrast to the standard operation of the message flow in which each node performs its processing in sequence.

You can also use these nodes to issue requests to applications outside the broker environment; messages can be sent asynchronously to external applications or services, the responses retrieved from those applications, and the responses combined to provide a single response to the original request message.

V6 enhancements to aggregation

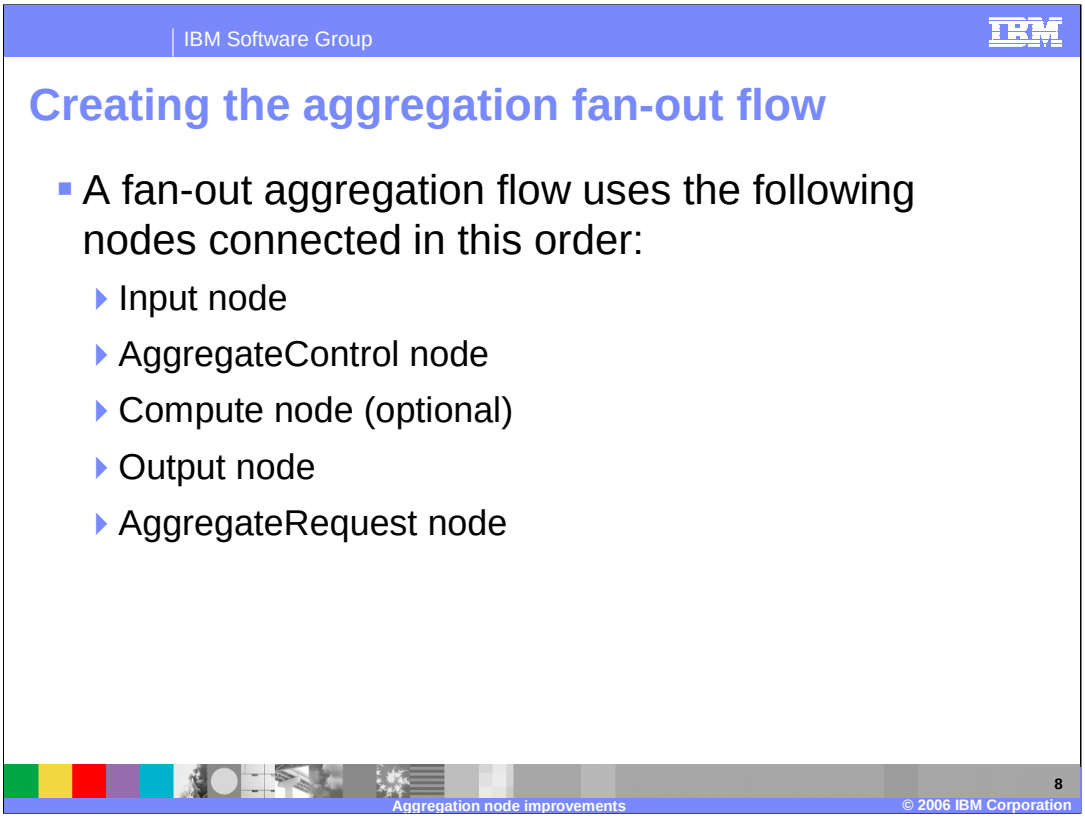
- Changes for WebSphere Message Broker V6
  - In previous versions, DB2® broker tables were used to persist aggregation
  - In V6, WebSphere MQ queues are used, improving performance
    - Aggregation can then be run in a non-persistent mode when persistence of aggregation requests is not required
    - Timeout properties can be used instead of Control terminals
  - External functions and properties remain unchanged

The primary change that was made for the aggregation nodes in WebSphere Message Broker Version 6 was to hold the aggregation state in a set of WMQ queues, instead of writing this data to the broker data base. This improves overall performance and throughput, as it does not incur the overhead of relational database access.

Section

Configuration

This section discusses configuration of aggregation nodes.



The aggregation fan-out flow receives the initial input message and restructures it to present a number of requests to a number of target applications.

Input node

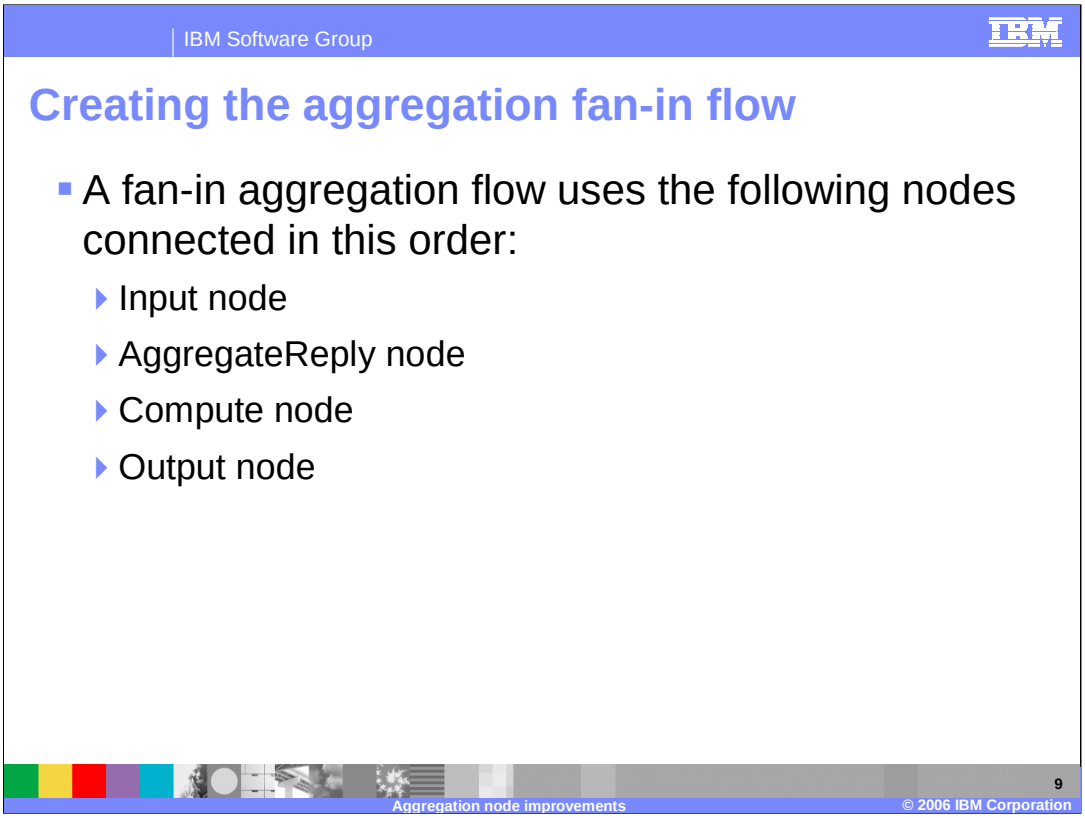
The input node receives an input message from which multiple request messages are generated. This can be any one of the built-in nodes, or a user-defined input node. Specify the source of input messages for this node. Connect the input node's out terminal to the in terminal of an AggregateControl node. This represents the simplest configuration; if appropriate, you can include other nodes between the input node and the AggregateControl node.

AggregateControl node

The AggregateControl node updates the LocalEnvironment associated with the input message with information required by the AggregateRequest node. The *Aggregate Name* property of the AggregateControl node identifies this particular aggregation which is used later to associate this AggregateControl node with a specific AggregateReply node. Connect the out terminal of the AggregateControl node to the in terminal of one or more Compute nodes that provide the analysis and breakdown of the request in the input message that is propagated on this terminal.

Compute node

The Compute node extracts information from the input message and constructs a new output message. If the target applications that handle the subtask requests can extract the information that they require from the single input message, you do not need to include a Compute node to split the message. You can pass the whole input message to all target applications. If your target applications expect to receive an individual request, not the whole input message, you must include a Compute node to generate each individual subtask output message from the input message. Connect the out terminal of each Compute node to the in terminal of the



The aggregation fan-in flow receives the responses to the request messages sent out by the fan-out flow and constructs a combined response message containing all the responses received.

Input node

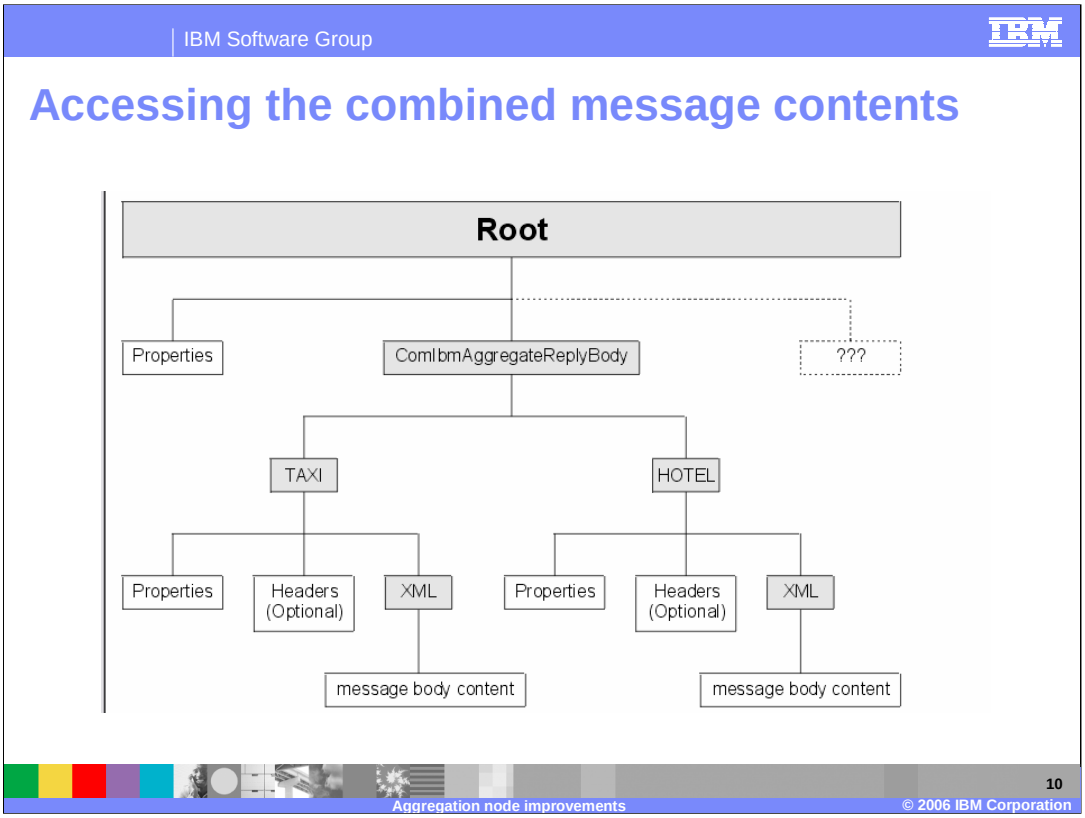
The input node receives the responses to the multiple request messages generated from the fan-out flow. This must be an input node that supports the request/reply model, such as an MQInput node, or a mixture of these nodes. The response received by each input node must be sent across the same protocol as the request to which it corresponds (for example, if you include an MQOutput node in the fan-out flow, the response to that request must be received by an MQInput node in this flow). In properties, specify the source of input messages for this node. Connect the input node's out terminal to the in terminal of an AggregateReply node. This represents the simplest configuration; if appropriate, you can include other nodes between the input node and the AggregateReply node.

AggregateReply node

The AggregateReply node receives the inbound responses from the input node through its in terminal. Each reply message received by the AggregateReply node is stored. When all the replies for a particular group of aggregation requests have been collected, the AggregateReply node creates an aggregated reply message and propagates this through the out terminal. Set the *Aggregate Name* property of the AggregateReply to identify this aggregation. Set this value to be the same value that you set for the *Aggregate Name* property in the corresponding AggregateControl node in the fan-out flow.

Compute node

The Compute node receives the message that contains the combined responses. It is unlikely that this combined message is in a format that is valid for output, so you



The AggregateReply node creates a folder in the combined message tree below Root, called ComIbmAggregateReplyBody. Below this, it creates a number of folders using the folder names that you set in the AggregateRequest nodes. The associated reply messages are put beneath them.

The request messages might have folder names such as:

TAXI

HOTEL

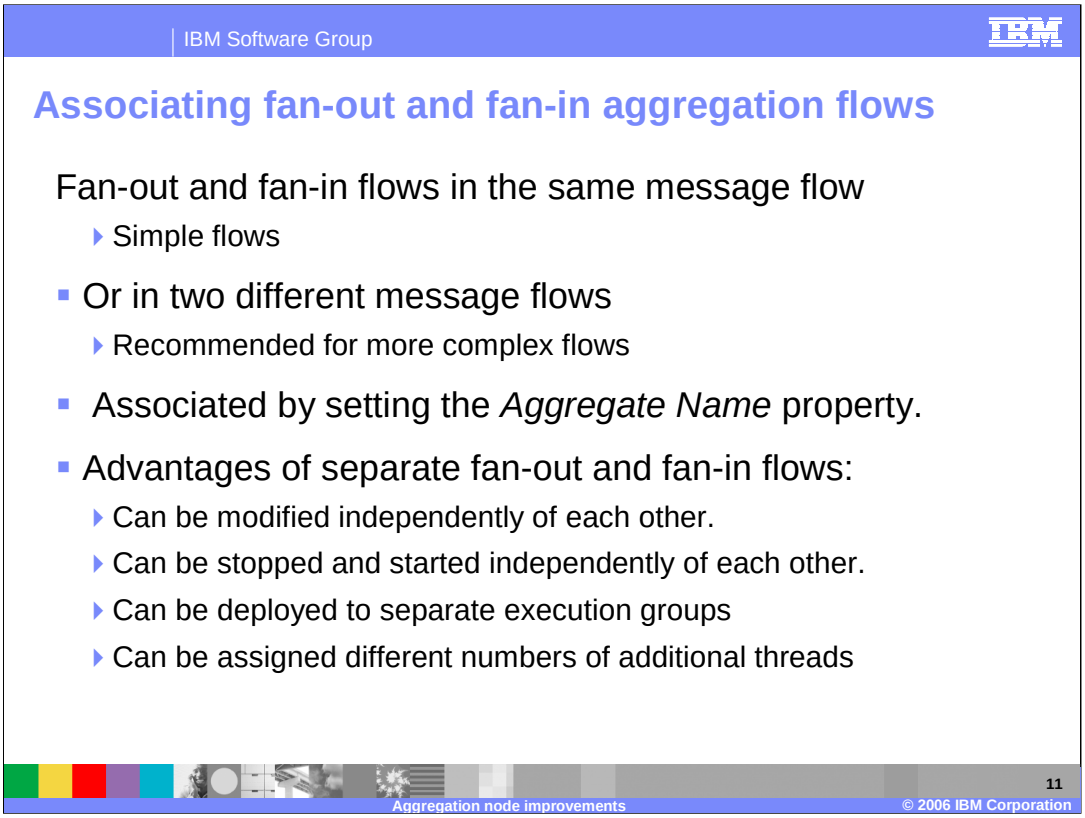
The resulting aggregated reply message created by the AggregateReply node might have a structure similar to that shown here.

You can use a Compute node to access the reply from the taxi company using the following correlation name:

InputRoot.ComIbmAggregateReplyBody.TAXI.xyz

The folder name does not have to be unique. If you have multiple requests with the folder name TAXI, you can access the separate replies using the array subscript notation, for example:

InputRoot.ComIbmAggregateReplyBody.TAXI[1].xyz
InputRoot.ComIbmAggregateReplyBody.TAXI[2].xyz



You can either create the fan-out and fan-in flows in the same message flow, or in two different message flows. In either case, the two parts of the aggregation are associated by setting the *Aggregate Name* property.

The advantages of creating separate fan-out and fan-in flows are:

•The two flows can be modified independently of each other.

•The two flows can be stopped and started independently of each other.

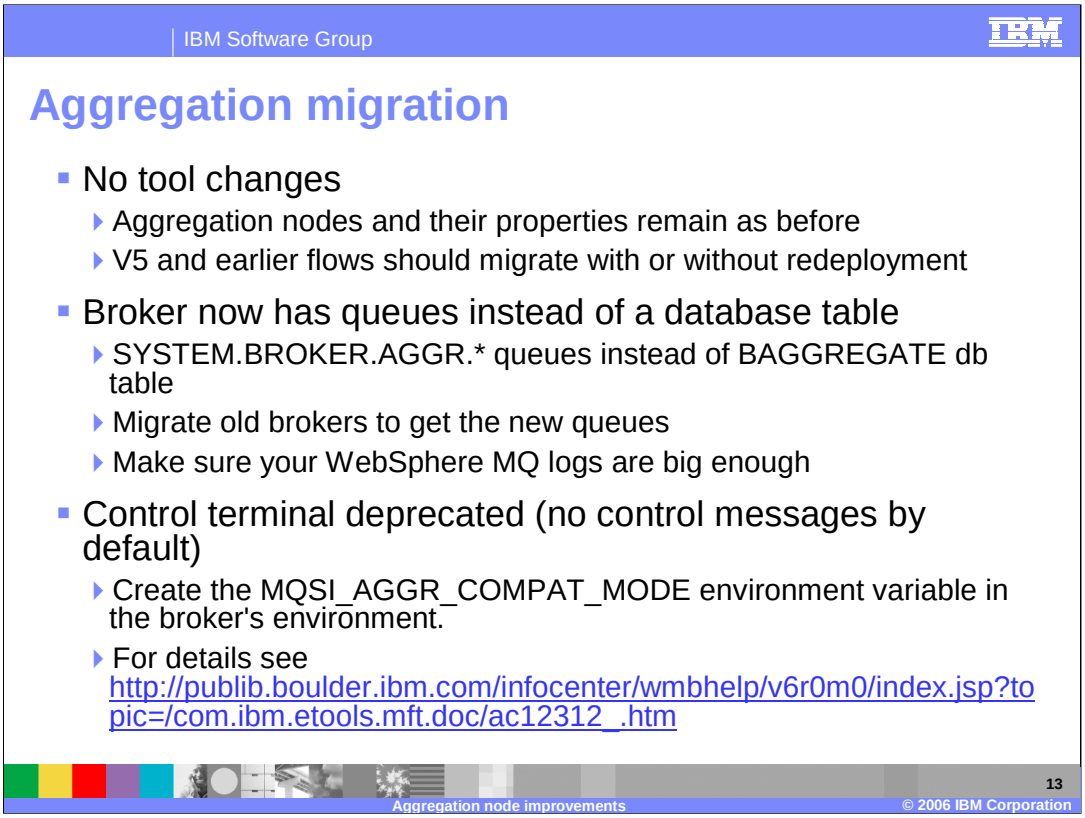
•The two flows can be deployed to separate execution groups to take advantage of multiprocessor systems, or to provide data segregation for security or integrity purposes.

•The two flows can be assigned different numbers of additional threads as appropriate to maintain an appropriate processing ratio.

Section

Migration

This section discusses migration of aggregation flows.

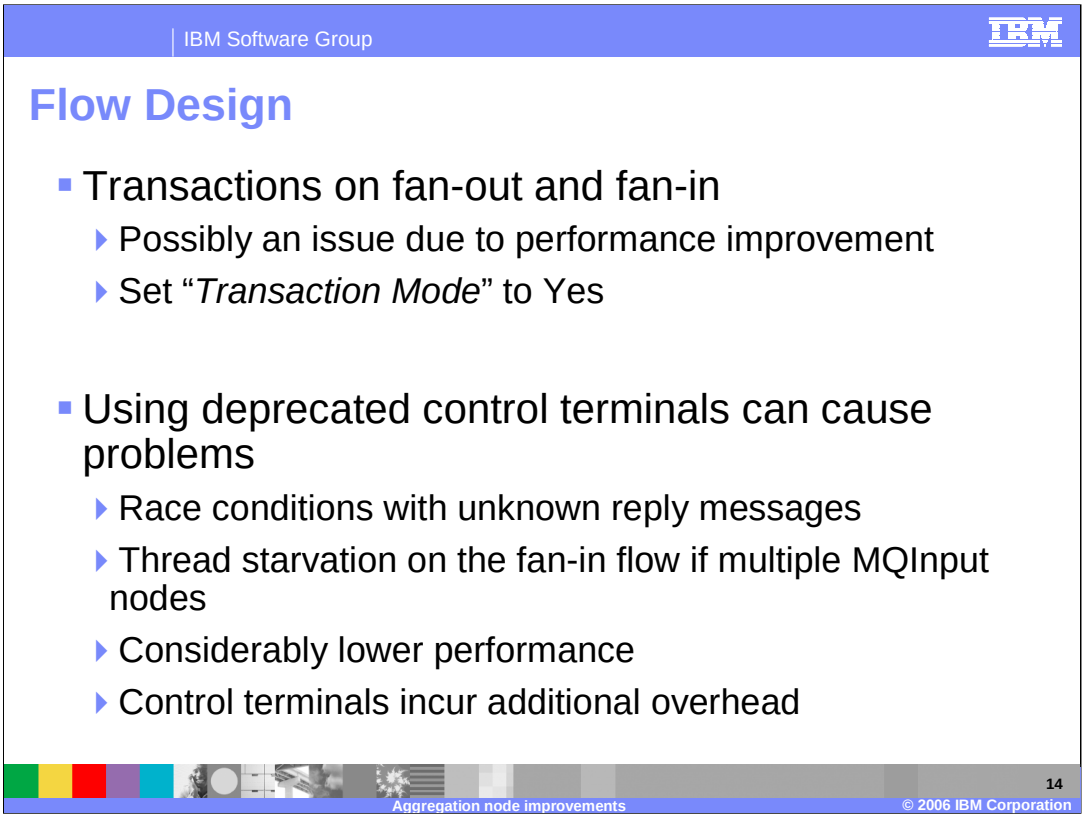


In Version 6 the aggregation nodes and their properties have not changed; therefore message flows from Version 5 should migrate easily to Version 6.

The aggregation state for each active message flow is now stored in a set of WMQ queues. These are prefixed SYSTEM.BROKER.AGGR. There are five queues, with the suffices: CONTROL, UNKNOWN, TIMEOUT, REQUEST, and REPLY. These replace the previous implementation of aggregation in the broker which used a database table called BAGGREGATE.

To allow the possibility of migration to Version 6, while the table contained live aggregation data, the BAGGREGATE table is not updated during the migration process. This allows the option to rollback, complete the unfinished aggregation, and proceed with the migration when no aggregation is active. One point to note is that since the aggregation function uses WMQ in Version 6, you may need to increase the size of your WMQ logs to handle the increased usage of WMQ.

The control terminal of the AggregationControl node has been deprecated in Version 6 even though the terminal is still available. By default, in WebSphere Message Broker Version 6.0 any connections from the control terminal of the AggregateRequest node to the AggregateReply node are ignored. This is to maximize the efficiency of aggregation flows and does not damage the reliability of aggregations. This is the optimum configuration. However, if you do want a control message to be sent from the AggregateControl node to the AggregateReply node, you must connect the control terminal to the corresponding AggregateReply node on the fan-in flow. For these



Since the aggregation function now performs significantly faster than in Version 5, there are some important considerations when designing your messages flows.

First, on the fan-out flow, you should ensure that the aggregation request messages are put under syncpoint. This will ensure that the AggregationReply node cannot receive replies to the aggregation request before it has received the notification of the request from the fan-out flow. You should do this by setting the Advanced property "Transaction Mode" to Yes.

Using the deprecated control terminals can result in problems, as a result of the improved performance of the aggregation nodes. First, it is possible to observe race conditions, where unknown reply messages arrive in a non-deterministic fashion. This means that a flow may work well for one invocation, but on a subsequent invocation it may receive an aggregation timeout. This inconsistent result will be difficult to manage within your message flow.

Secondly, you may experience thread starvation on the fan-in flow if you have multiple MQInput nodes feeding a single AggregationReply node. This only applies if the Control terminal of the AggregateControl node in your fan-out flow is connected to output control messages to a queue.

Finally, performance is impacted if you use the deprecated control terminals. This is because it results in the overhead of building an XML Control Message, sending it out

Aggregation timeout

- Set the Timeout property of the AggregateControl node (in seconds)
  - Default is 0 – no timeout
- AggregateReply node sends a partial response message to its timeout terminal if not all replies received in specified time (from AggregateControl node)
- AggregateReply node sends unexpected replies to its unknown terminal
  - Set the Unknown Message Timeout property
- In previous versions, control terminals of AggregateControl node and AggregateReply node were used for timeout processing.
  - V6 uses WebSphere MQ Expiry time

In certain situations you might need to receive an aggregated reply message within a certain time. Some reply messages might be slow to return, or might never arrive. Set the *Timeout* property of the AggregateControl node to specify how long (in seconds) the broker must wait for replies. By default, this property is set to 0, which means that there is no timeout and the broker waits indefinitely.

If the timeout interval passes without all the replies arriving, the replies that have arrived are turned into an aggregated reply message by the corresponding AggregateReply node, and propagated to its timeout terminal. If you choose, you can process this partial response message in the same way as a complete aggregated reply message. If you prefer, you can provide special processing for incomplete aggregated replies.

When a message arrives at the in terminal of an AggregateReply node, it is examined to see if it is an expected reply message. If it is not recognized, it is propagated to the unknown terminal. You might want the broker to wait for a given period of time before doing this, because:

•The reply message might arrive before the work performed by the AggregateRequest node has been transactionally committed.

•The reply message might arrive before the control message. This situation can be avoided by leaving the control terminal of the AggregateControl node unconnected.

Set the *Unknown Message Timeout* property on the AggregateReply node. When you set this property, a message that cannot be recognized immediately as a valid reply is held persistently within the broker for the number of seconds that you specify for this property.

If the unknown timeout interval expires, and the message is recognized, it is processed. The node also checks to see if this previously unknown message is the last reply needed to make an aggregation complete. If it is, the aggregated reply message is constructed and propagated.

Debugging

- Lost aggregation data during migration from V5 to V6
  - Allow aggregations to finish before stopping broker
- Control messages not sent from AggregateControl node
  - Deprecated behavior. Can be used, but caveat emptor
- Broker queue manager runs out of log space
  - Increase the WMQ logs. Aggregation now uses WMQ instead of DB2
- Above average unknown message generation
  - Check transactional control of fan-out flow
  - Review “Extending the Sample” in the Aggregation Sample
- Basic WMQ Errors
  - Queue full, message too big, etcetera

This slide discusses some possible error scenarios.

When migrating from Version 5 to Version 6, you should ensure that all active aggregations have completed. This will avoid the possibility of losing any of this data during migration.

The Control terminal is now deprecated. In Version 6, control messages are not sent from the AggregateControl node, even if this is connected in the message flow. If this function is still required when running the flow in Version 6, you will need to set the environment variable accordingly. Be aware that this function may be removed from future versions of Message Broker.

If the Broker Queue Manager runs out of queue space, increase the WMQ logs size. This is a result of the increased usage of WMQ for this function.

You might see an increase in the rate of arrival of unknown messages. These should be handled through your normal timeout processing. Some might be avoided by setting the transaction context in the fan-out flow.

Finally, if you’ve got basic WMQ errors such as the queue is full, the message is too big for the queue, etc, make the appropriate alterations to the queue definitions.

Section

Summary and references

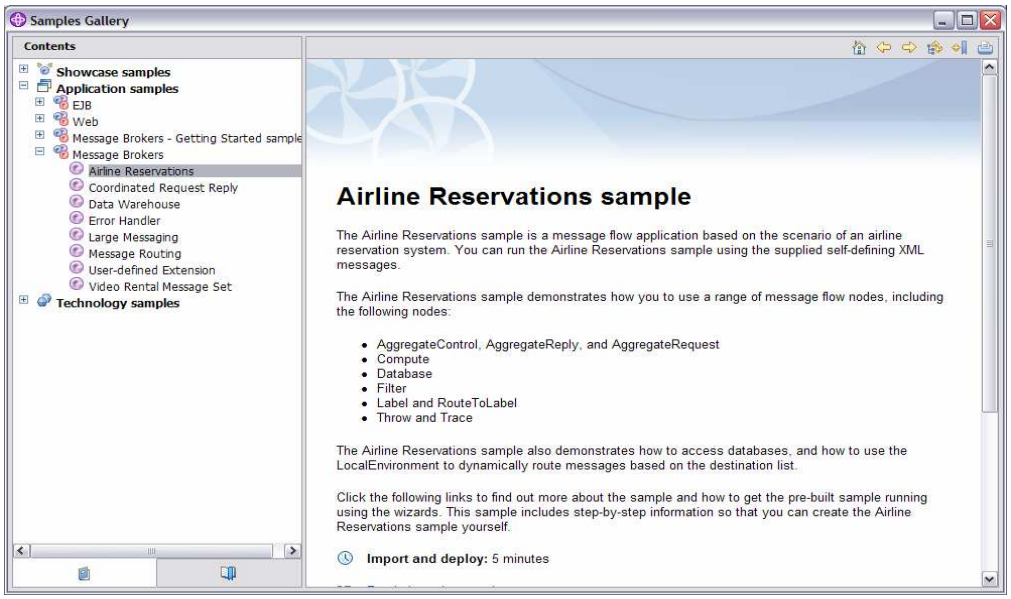
This section contains a summary and references.

Summary

- Overview
- Configuration of aggregation flows
- Aggregation migration

This session looked at aggregation nodes, key items to be considered when configuring the nodes, and how to migrate existing aggregation flows.

Samples



The supplied samples contain good examples of how to use the aggregation function. Use of this is fully demonstrated in the “Airlines” sample, access from 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:

DB2 WebSphere

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.

Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.