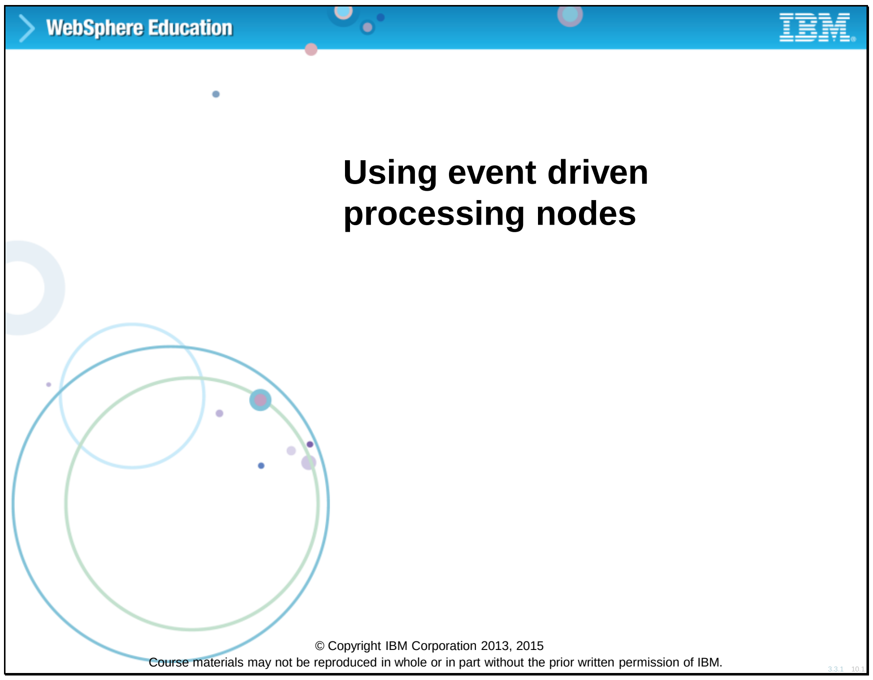
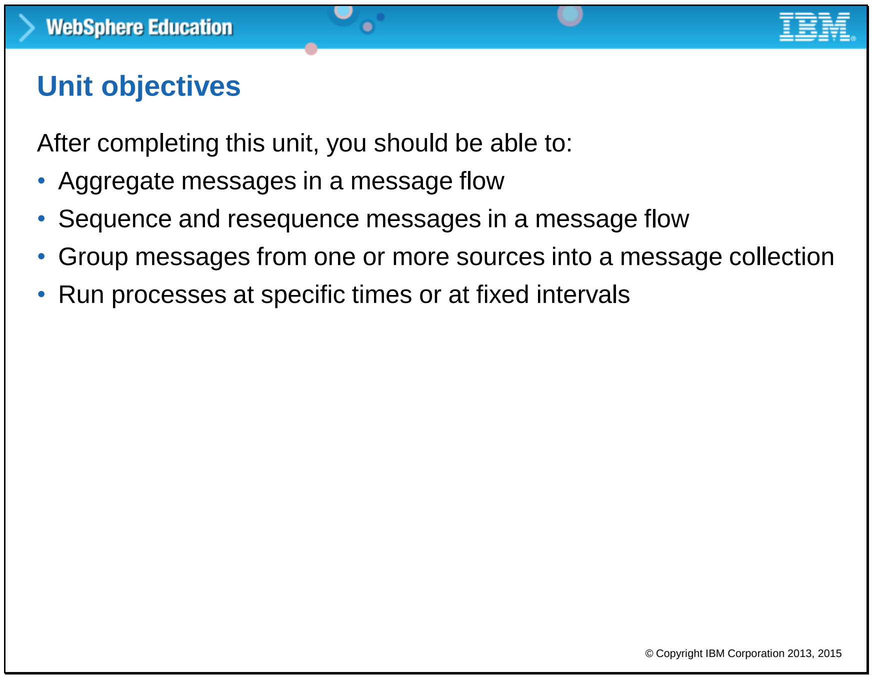
Slide 1



WebSphere Education

Using event driven processing nodes



WebSphere Education

Unit objectives

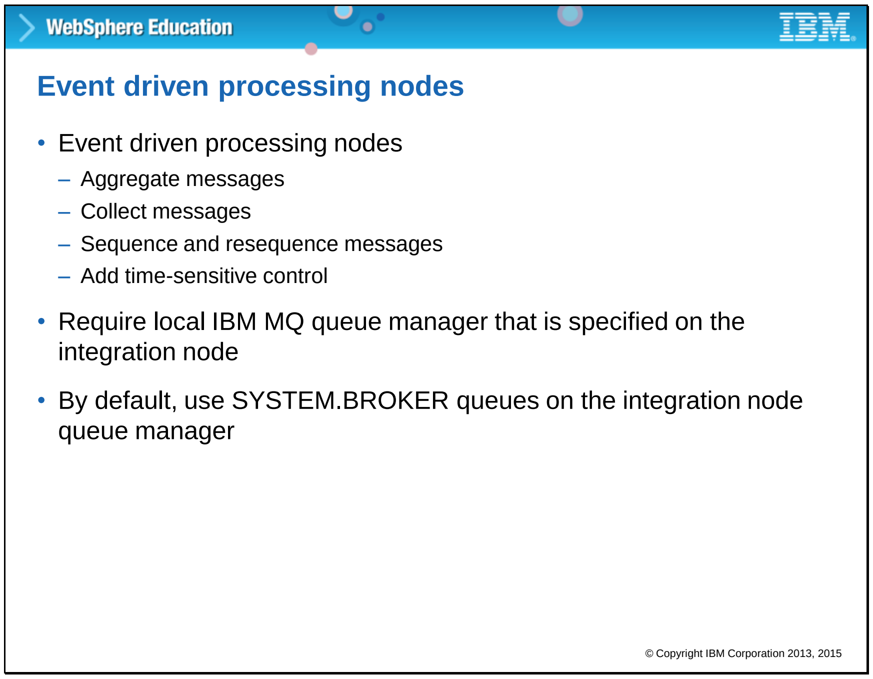
After completing this unit, you should be able to:
- Aggregate messages in a message flow
- Sequence and resequence messages in a message flow
- Group messages from one or more sources into a message collection
- Run processes at specific times or at fixed intervals

Unit objectives

Event-driven message processing nodes control the flow of messages through message flows by using aggregation, message collections, message sequences, and timeout flows. In this unit, you learn how to aggregate and control the sequence of messages in a message flow. You also learn how to use time-sensitive nodes to control when processes run.

After completing this unit, you should be able to:
- Aggregate messages in a message flow
- Sequence and resequence messages in a message flow
- Group messages from one or more sources into a message collection
- Run processes at specific times or at fixed intervals

Slide 3


WebSphere Education

Event driven processing nodes

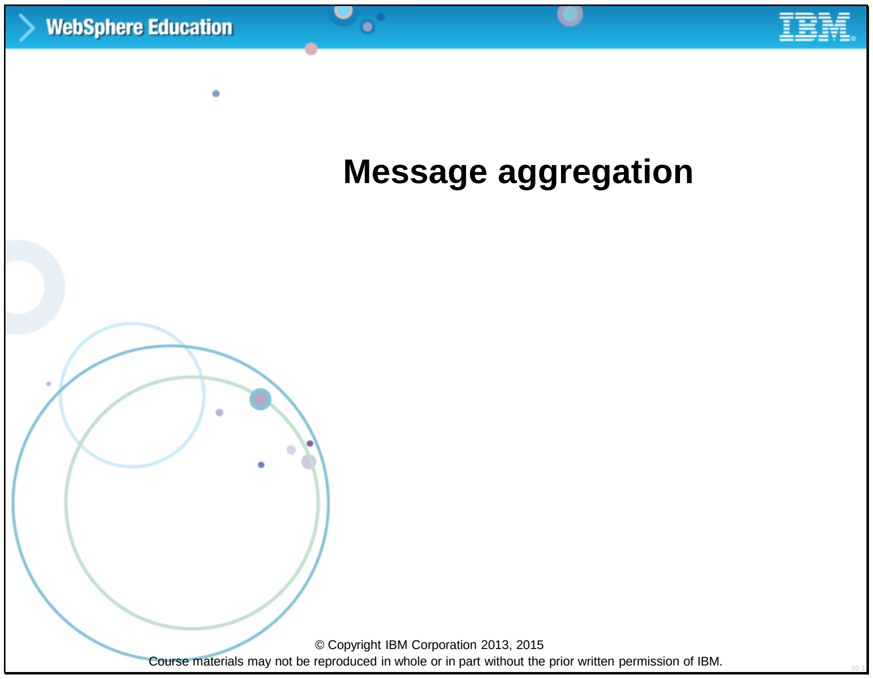
- Event driven processing nodes
  - Aggregate messages
  - Collect messages
  - Sequence and resequence messages
  - Add time-sensitive control
- Require local IBM MQ queue manager that is specified on the integration node
- By default, use SYSTEM.BROKER queues on the integration node queue manager

Event-driven processing nodes

The Integration Toolkit includes some event-driven processing nodes that are used for message aggregation, timeout, message collections, and message sequences.

Any message flows that contain event-driven processing nodes require that a local MQ queue manager is specified for the integration node.

The event-driven processing nodes also use special SYSTEM.BROKER queues, which the queue manager owns, to store information about in-flight messages. The Integration Bus installation includes command files that create the queues on the integration node’s queue manager.

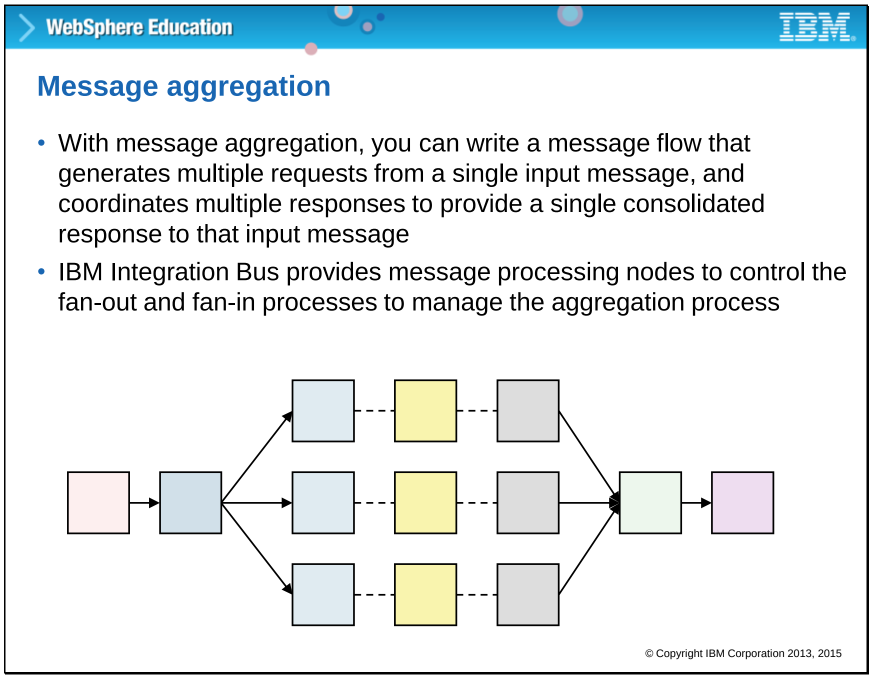


WebSphere Education

Message aggregation

Topic 1: Message aggregation

In this topic, you learn how to aggregate messages in a message flow by using the Aggregate Control, Aggregate Request, and Aggregate Reply nodes.


WebSphere Education
IBM

Message aggregation

- With message aggregation, you can write a message flow that generates multiple requests from a single input message, and coordinates multiple responses to provide a single consolidated response to that input message
- IBM Integration Bus provides message processing nodes to control the fan-out and fan-in processes to manage the aggregation process

Message aggregation

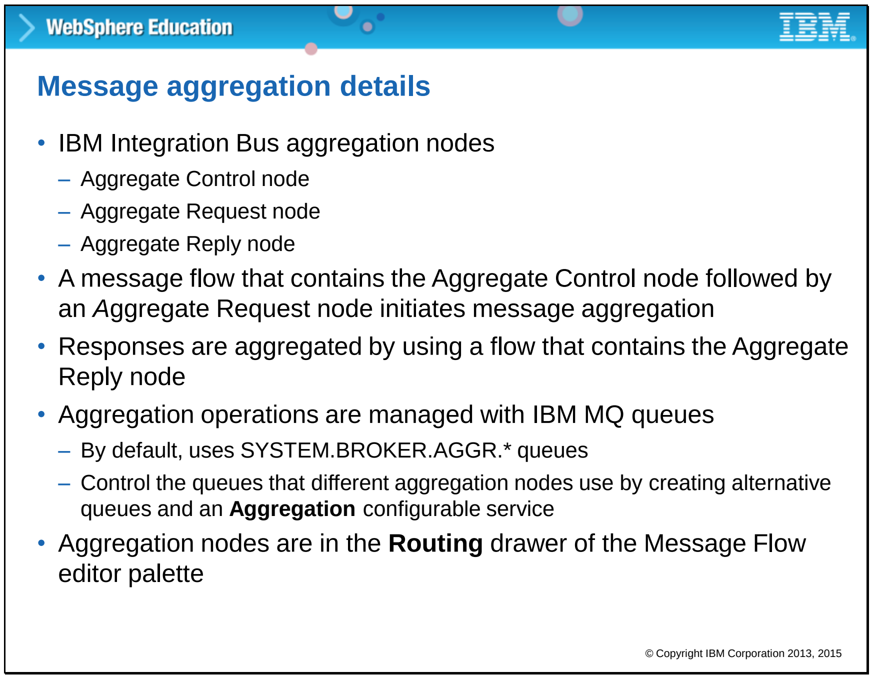
Message aggregation is an extension of the request-reply processing model. It combines the generation and fan-out of related requests with the fan-in of the corresponding replies, and compiles those replies into a single aggregated reply message.

The initial request that the message flow receives, 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 it is provided by a message flow that includes aggregation nodes.

Replies from the subtasks are combined and merged into a single reply, which is returned to the original requester (or another target application) to indicate completion of the processing. This process is known as fan-in, and it is also provided by a message flow that includes aggregation nodes.

IBM Integration Bus provides message processing nodes to control the fan-out and fan-in processes to manage the aggregation process.

Using an aggregation message flow is easier than writing flows with multiple callouts and an MQ Get node or the equivalent nodes in other message transports because Integration Bus manages the fan-out and fan-in processing.



WebSphere Education

Message aggregation details

- IBM Integration Bus aggregation nodes
  - Aggregate Control node
  - Aggregate Request node
  - Aggregate Reply node
- A message flow that contains the Aggregate Control node followed by an *A*ggregate Request node initiates message aggregation
- Responses are aggregated by using a flow that contains the Aggregate Reply node
- Aggregation operations are managed with IBM MQ queues
  - By default, uses SYSTEM.BROKER.AGGR.* queues
  - Control the queues that different aggregation nodes use by creating alternative queues and an **Aggregation** configurable service
- Aggregation nodes are in the **Routing** drawer of the Message Flow editor palette

Message aggregation details

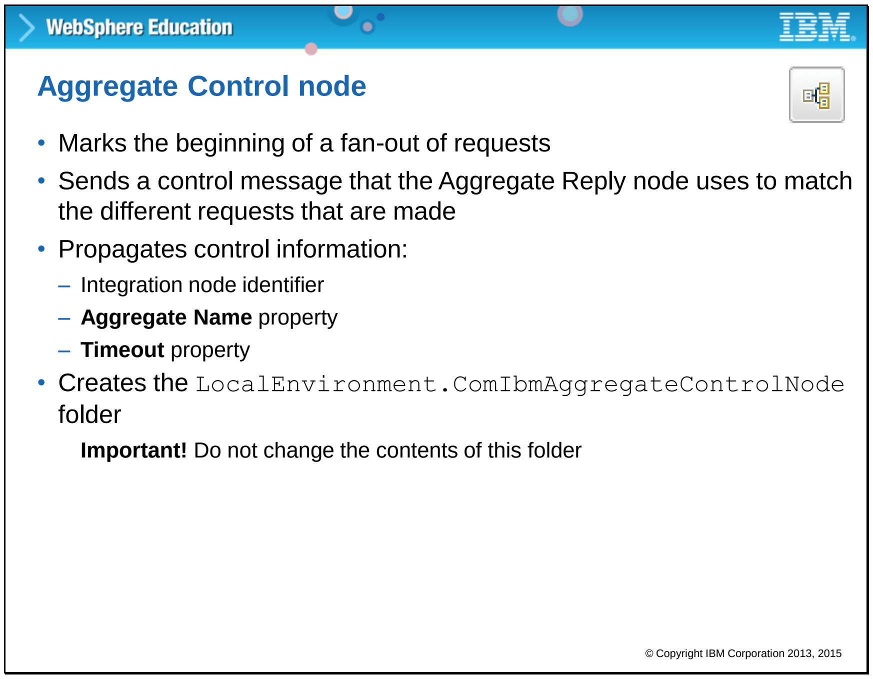
The Integration Toolkit provides three message processing nodes for message aggregation: Aggregate Control, Aggregate Request, and Aggregate Reply.

A message flow that contains the Aggregate Control node that is followed by an Aggregate Request node initiates message aggregation.

Responses are aggregated by using a flow that contains the Aggregate Reply node.

Aggregation operations are managed with MQ queues with the SYSTEM.BROKER.AGGR prefix. You can control the queues that different aggregation nodes use by creating alternative queues and creating an Aggregation configurable service.

Aggregation nodes are in the Routing drawer of the Message Flow editor palette.



Aggregate Control node

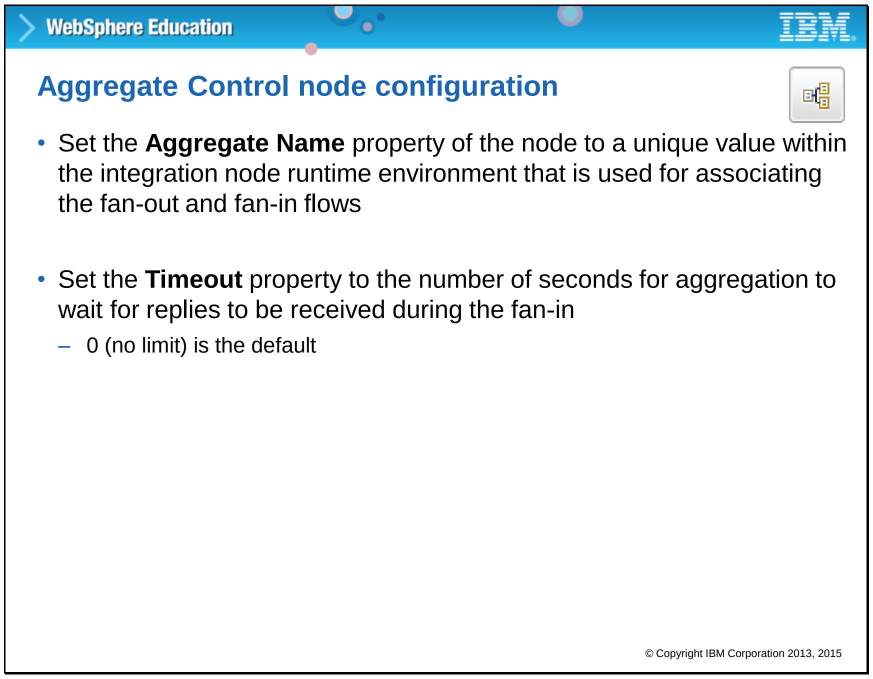
- Marks the beginning of a fan-out of requests
- Sends a control message that the Aggregate Reply node uses to match the different requests that are made
- Propagates control information:
  - Integration node identifier
  - **Aggregate Name** property
  - **Timeout** property
- Creates the `LocalEnvironment.ComIbmAggregateControlNode` folder

  Important! Do not change the contents of this folder

Aggregate Control node

The Aggregate Control node starts an aggregation fan-out flow. It creates a folder in the LocalEnvironment tree that manages the aggregation process and contains the control information such as the integration node identifier, aggregation name, and a timeout property.

Do not change the contents of the Aggregation Control node branch of the LocalEnvironment tree.



WebSphere Education

Aggregate Control node configuration

- Set the **Aggregate Name** property of the node to a unique value within the integration node runtime environment that is used for associating the fan-out and fan-in flows
- Set the **Timeout** property to the number of seconds for aggregation to wait for replies to be received during the fan-in
  - 0 (no limit) is the default

Aggregate Control node configuration

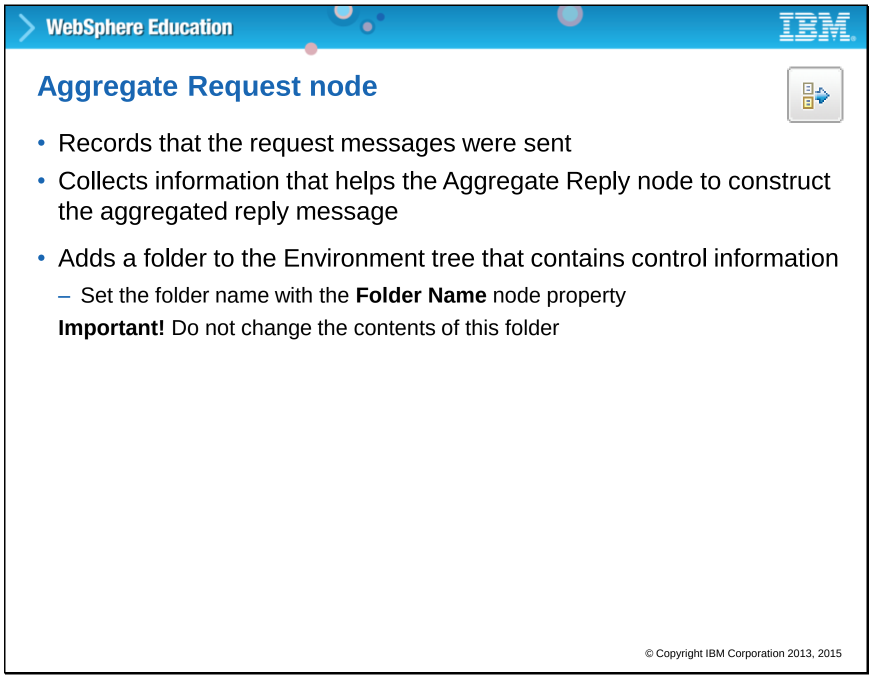
When you use Integration Bus to provide message aggregation, you associate the fan-out flow and the fan-in flow.

The **Aggregate Name** property associates the fan-out and fan-in flows for a specific aggregation flow. Set the **Aggregate Name** property to a value that is unique within the integration node runtime environment. Do not use the same value for this property that another aggregation message flow that runs in the same integration node runtime environment uses.

The **Timeout** property sets the number of seconds for aggregation to wait for replies from the fan-in.
If a timeout is not set on the Aggregate Control node, then aggregate requests that are stored internally are not removed unless all aggregate reply messages return. This situation might lead to a gradual build-up of messages on the internal queues. To avoid this situation, set the timeout to a value other than zero so that when the timeout is reached the requests are removed and the queues do not fill up with redundant requests.

It is a good practice to set the timeout value to a large value; for example, 86400 seconds (24 hours), so that the queues clear old aggregation messages even if timeouts are not required or expected.

Slide 9


WebSphere Education IBM

Aggregate Request node

- Records that the request messages were sent
- Collects information that helps the Aggregate Reply node to construct the aggregated reply message
- Adds a folder to the Environment tree that contains control information
  - Set the folder name with the **Folder Name** node property

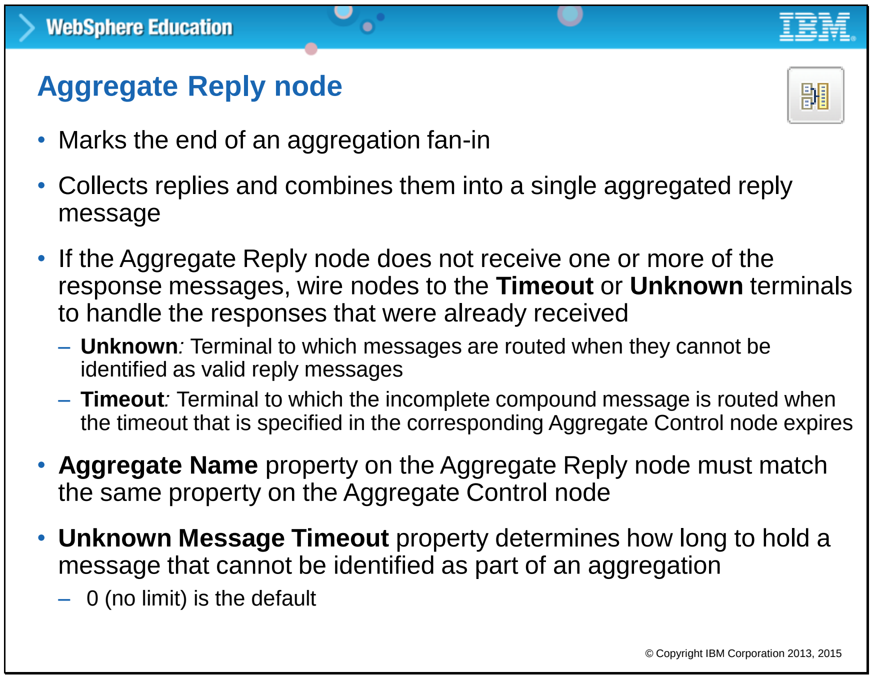
  Important! Do not change the contents of this folder

Aggregate Request node

The Aggregate Request node records information about the request messages that are sent to the various services. It stores this information in the Environment tree, in a folder that you specify in the **Folder Name** property of the node.

Set the **Basic** property **Folder name** to a value that identifies the type of request that is sent. This value is used by the Aggregate Reply node to match up with the reply message when it is received in the fan-in flow. The folder name that you specify for each request that the fan-out flow generates must be unique.

The Aggregate Request node writes a record in MQ for each message that it processes. This record enables the Aggregate Reply node to identify which request each response is associated with. If your output nodes are non-transactional, the response message might arrive at the fan-in flow before this MQ update is committed.



WebSphere Education

Aggregate Reply node

- Marks the end of an aggregation fan-in
- Collects replies and combines them into a single aggregated reply message
- If the Aggregate Reply node does not receive one or more of the response messages, wire nodes to the **Timeout** or **Unknown** terminals to handle the responses that were already received
  - **Unknown**: Terminal to which messages are routed when they cannot be identified as valid reply messages
  - **Timeout**: Terminal to which the incomplete compound message is routed when the timeout that is specified in the corresponding Aggregate Control node expires
- **Aggregate Name** property on the Aggregate Reply node must match the same property on the Aggregate Control node
- **Unknown Message Timeout** property determines how long to hold a message that cannot be identified as part of an aggregation
  - 0 (no limit) is the default

Aggregate Reply node

The Aggregate Reply node ends a fan-in operation.

The Aggregate Reply node receives the inbound responses from the input node through its In terminal. The Aggregate Reply node stores each reply message for subsequent processing. When all the replies for a particular group of aggregation requests are collected, the Aggregate Reply node creates an aggregated reply message, and propagates it through the Out terminal.

The Aggregate Reply node **Unknown Message Timeout** property sets the amount of time that “invalid” messages are held until they are propagated to the **Unknown** terminal. This action covers the case in which the first replies arrive, while not all of the requests were sent out.

To explicitly handle unrecognized messages, connect the Unknown terminal to another node, or sequence of nodes. If you do not connect this terminal to another node in the message flow, messages that are propagated through this terminal are discarded.

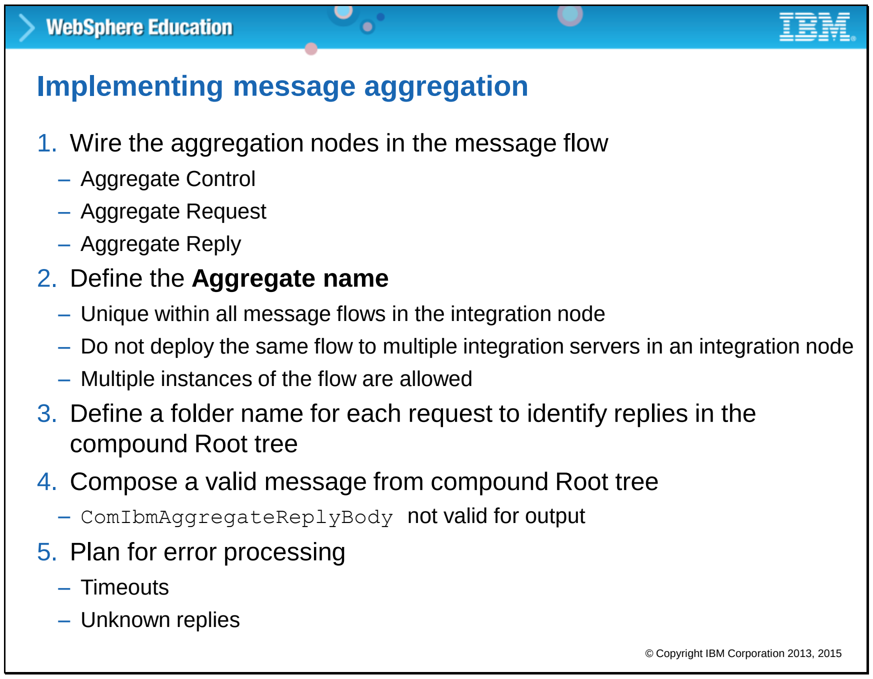
The **Timeout** property on the Aggregate Control node sets the duration to wait for the Aggregate Reply node to receive all aggregation replies. If that **Timeout** property is exceeded, the message assembly is sent to the **Timeout** terminal.

If you specified a **Timeout** value for this aggregation in the Aggregate Control node, and you want to explicitly handle timeouts that expire before all replies are received, connect the

Timeout terminal to another node, or sequence of nodes. If the timer expires, the partially complete aggregated replies are sent to the Timeout terminal. If you do not connect this terminal to another node in the message flow, messages that are propagated through this terminal are discarded.

Set the **Aggregate name** property of the Aggregate Reply node to identify this aggregation. Set this value to be the same value that you set for the **Aggregate name** property in the corresponding Aggregate Control node in the fan-out flow.

Slide 11



WebSphere Education

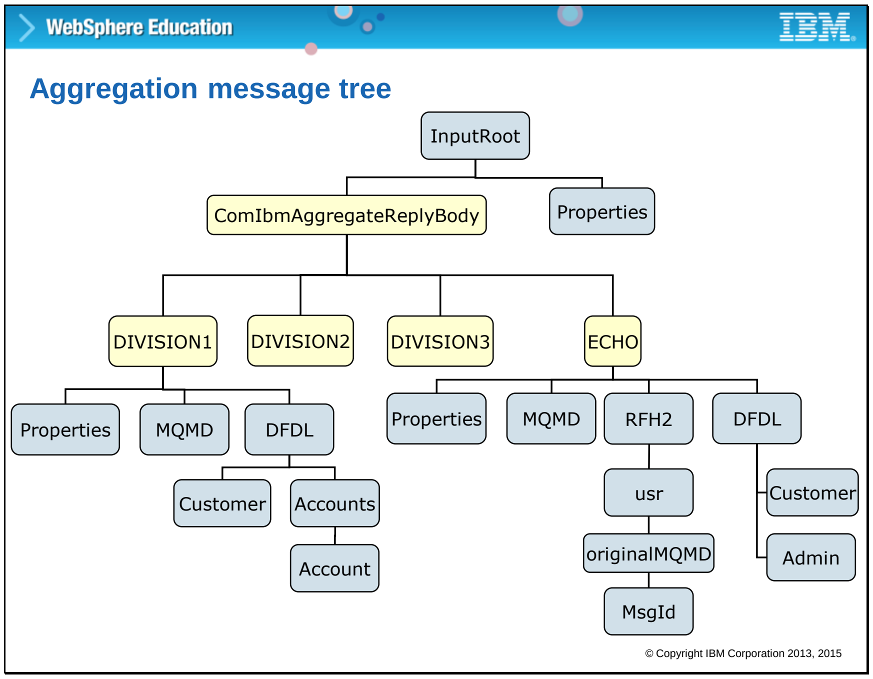
Implementing message aggregation

1. Wire the aggregation nodes in the message flow
   - Aggregate Control
   - Aggregate Request
   - Aggregate Reply
2. Define the **Aggregate name**
   - Unique within all message flows in the integration node
   - Do not deploy the same flow to multiple integration servers in an integration node
   - Multiple instances of the flow are allowed
3. Define a folder name for each request to identify replies in the compound Root tree
4. Compose a valid message from compound Root tree
   - `ComIbmAggregateReplyBody` not valid for output
5. Plan for error processing
   - Timeouts
   - Unknown replies

Implementing message aggregation

This figure summarizes the steps for implementing message aggregation.

1. Add and wire the aggregation nodes in the fan-out and fan-in message flows.
2. Define the unique Aggregate name.
3. Define a folder name for each request to identify replies in the compound Root tree.
4. Use a Compute node or Java Compute node to compose a valid message from compound Root tree.
5. Plan for error processing by setting timeout values and connecting all output terminals.



Aggregation message tree

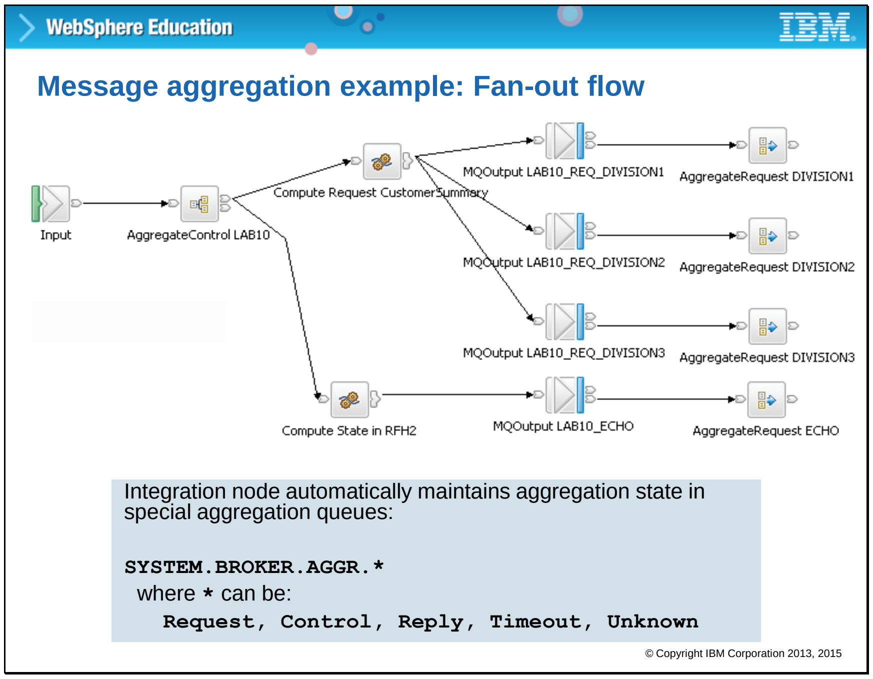
Aggregation message tree

The Aggregate Reply node creates a logical message tree with multiple bodies. Each reply is contained in a separate body folder, which the request name identifies. In the figure, the body folders are DIVISION1, DIVISION2, DIVISION3, and ECHO.

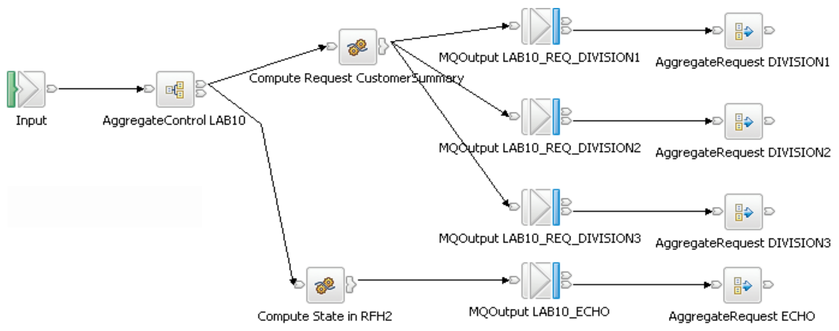
Typically, the format of this combined message is not valid for output because the aggregated reply message has an unusual structure and cannot be parsed into the bit stream required by some nodes, for example the MQ Output node. You must include a Compute node in the message flow and configure this node to create a valid output message that can be sent to the destination application.

Next, you see an example of message aggregation flows. You also create message aggregation flows in the exercise for this unit.

Slide 13


WebSphere Education
IBM

Message aggregation example: Fan-out flow


Input
AggregateControl LAB10
Compute Request CustomerSummary
MQOutput LAB10_REQ_DIVISION1
AggregateRequest DIVISION1
MQOutput LAB10_REQ_DIVISION2
AggregateRequest DIVISION2
MQOutput LAB10_REQ_DIVISION3
AggregateRequest DIVISION3
Compute State in RFH2
MQOutput LAB10_ECHO
AggregateRequest ECHO

Integration node automatically maintains aggregation state in special aggregation queues:

`SYSTEM.BROKER.AGGR.*`
where **`*`** can be:
`Request, Control, Reply, Timeout, Unknown`

Message aggregation example: Fan-out flow

This slide shows an example of a message aggregation fan-out flow.

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.

The Aggregate Control node starts the aggregation fan-out flow. It creates a folder in the LocalEnvironment tree that manages the aggregation process and contains the control information such as the aggregation name.

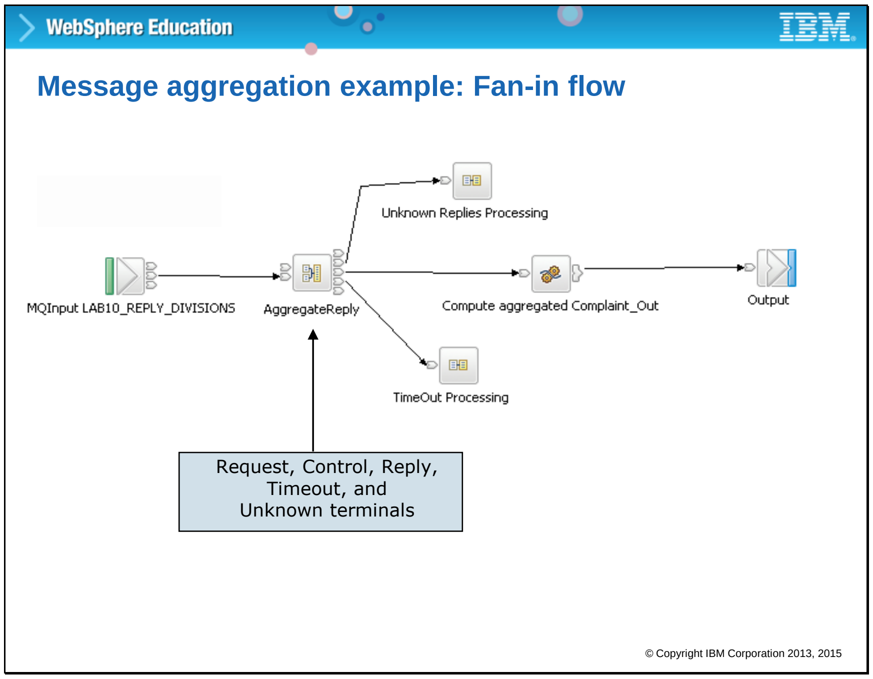
The Compute_Request_CustomerSummary node extracts information from the input message and constructs and routes a new output message based on the value of the DIVISION field in the message body.

In this example, a second Compute node constructs another message that contains state information for MQ.

The flow includes an Aggregate Request node for each output message that the fan-out flow generates. In this example, the Aggregate Request node **Folder name** property identifies the destination for each message. The Aggregate Request node writes a record in MQ for each message that it processes. This record enables the Aggregate Reply node to identify which request each response is associated with.

After all request messages are sent, the control information is propagated from the Aggregate Control node to the Aggregate Reply node in the fan-in flow.

Slide 14


WebSphere Education

Message aggregation example: Fan-in flow

Unknown Replies Processing
MQInput LAB10_REPLY_DIVISIONS
AggregateReply
Compute aggregated Complaint_Out
Output
TimeOut Processing
Request, Control, Reply, Timeout, and Unknown terminals

Message aggregation example: Fan-in flow

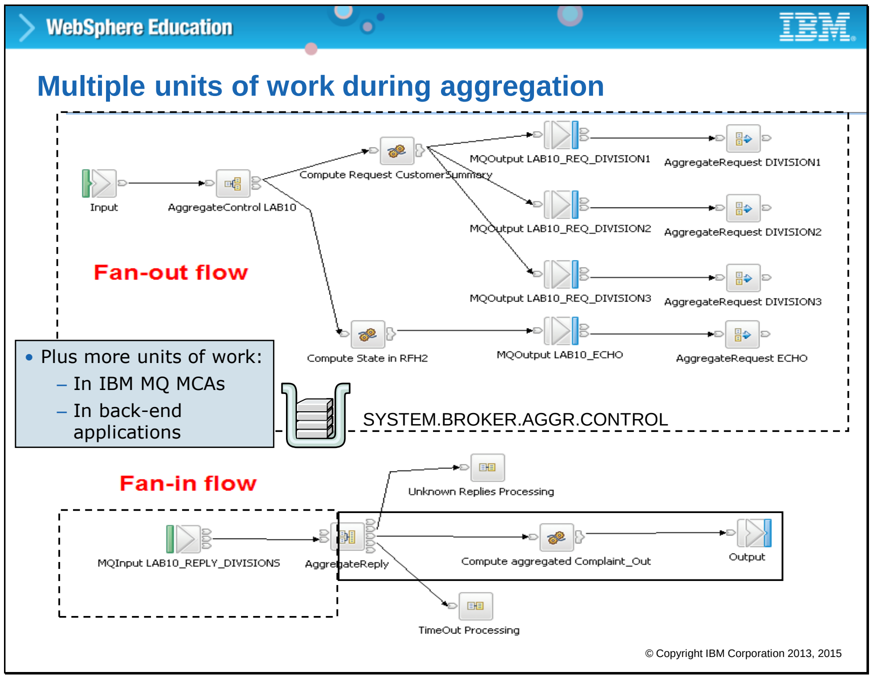
This slide shows the message aggregation fan-in flow for the fan-out flow that was described on the previous slide.

The Aggregate Reply node records any incoming reply from the MQ Input node and stores it until all outstanding replies arrive, or a timeout occurs. Then, the Aggregate Reply node composes a logical message tree with multiple bodies. Each reply is contained in separate body folder, which the request name identifies.

In this example, the Aggregate Reply node Timeout and Unknown terminals are wired to explicitly handle errors.

A Compute node creates a valid output message that can be sent to the destination application.

Slide 15



Multiple units of work during aggregation

Input
AggregateControl LAB10
Compute Request CustomerSummary
MQOutput LAB10_REQ_DIVISION1
AggregateRequest DIVISION1
MQOutput LAB10_REQ_DIVISION2
AggregateRequest DIVISION2
MQOutput LAB10_REQ_DIVISION3
AggregateRequest DIVISION3
Fan-out flow
Compute State in RFH2
MQOutput LAB10_ECHO
AggregateRequest ECHO
SYSTEM.BROKER.AGGR.CONTROL
Fan-in flow
Unknown Replies Processing
MQInput LAB10_REPLY_DIVISIONS
AggregateReply
Compute aggregated Complaint_Out
Output
TimeOut Processing

- Plus more units of work:
  - In IBM MQ MCAs
  - In back-end applications

Multiple units of work during aggregation

An aggregation flow is a complex application that involves multiple units of work.

In a fan-out flow, the first logical unit of work ends when all requests were sent out, and the automatic control message is put onto the SYSTEM.BROKER.AGGR.CONTROL queue.

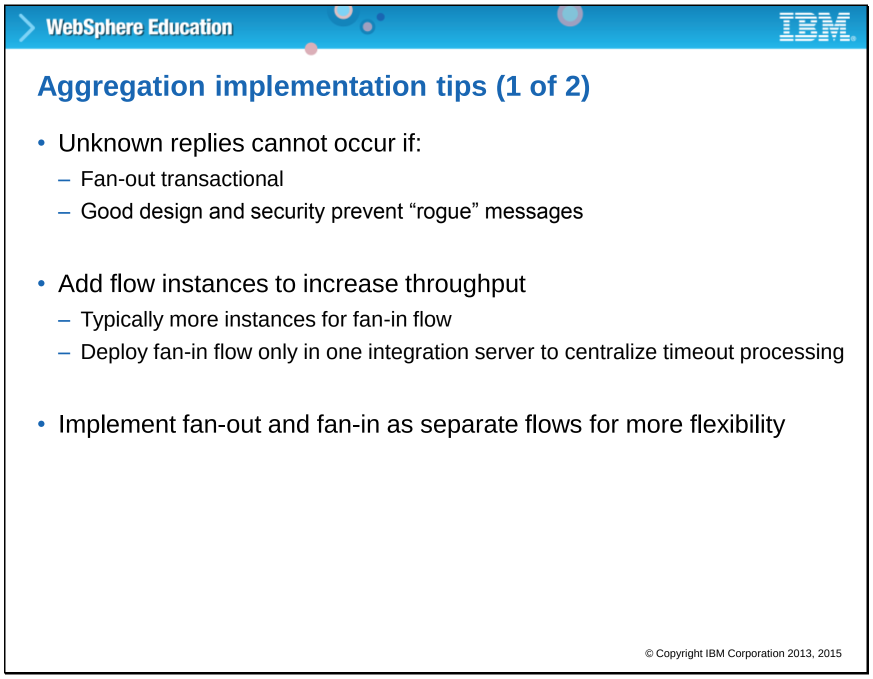
In a fan-in flow, a logical unit of work exists for each incoming reply is while aggregating from the MQ Input node to the Aggregate Reply node.

For the aggregated result message, a new logical unit of work starts in the Aggregate Reply node when all the replies arrive, or the timeout interval expires. The Aggregate Reply has a **TransactionMode** property so that you can control the transaction mode after message aggregation.

More logical units of work exist during MQ transport and in those applications that use the requests and send replies to the fan-in flow.

Logical units of work are important for error handling and recovery. Be sure to consider connecting Catch terminals to capture any errors during message processing.

Slide 16



WebSphere Education

Aggregation implementation tips (1 of 2)

- Unknown replies cannot occur if:
  - Fan-out transactional
  - Good design and security prevent “rogue” messages
- Add flow instances to increase throughput
  - Typically more instances for fan-in flow
  - Deploy fan-in flow only in one integration server to centralize timeout processing
- Implement fan-out and fan-in as separate flows for more flexibility

Aggregation implementation tips (1 of 2)

When a message arrives at the In terminal of an Aggregate Reply node, it is examined to see whether it is an expected reply message. If it is not recognized, it is propagated to the **Unknown** terminal. You might want the integration node to wait for a specific duration before propagating the message because the reply message might arrive before the work that the Aggregate Request node does is transactionally committed.

Put the fan-out flow under transactional control to ensure that the fan-out flow completes before any response messages get to the Aggregate Reply.

Add flow instances to increase throughput. It is more common to add more instances for fan-in flow that aggregates the messages.

Deploy fan-in flow only in one integration server to centralize timeout processing.

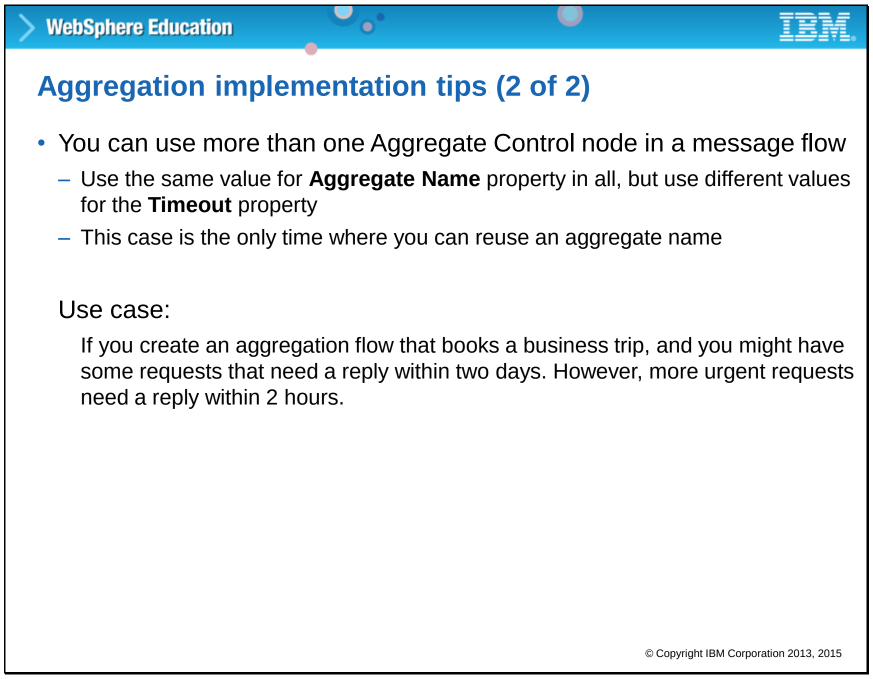
You can either create the fan-out and fan-in flows in the same message flow, or in two different message flows. In either case, the two parts of the aggregation are associated by setting the **Aggregate Name** property.

A single flow is easier to implement for a simple case, but some limitations to this approach exist. You might find that the flexibility offered by two message flows is preferable.

- 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 integration servers to take advantage of multiprocessor systems, or to provide data segregation for security or integrity purposes.

Slide 17



Aggregation implementation tips (2 of 2)

- You can use more than one Aggregate Control node in a message flow
  - Use the same value for **Aggregate Name** property in all, but use different values for the **Timeout** property
  - This case is the only time where you can reuse an aggregate name

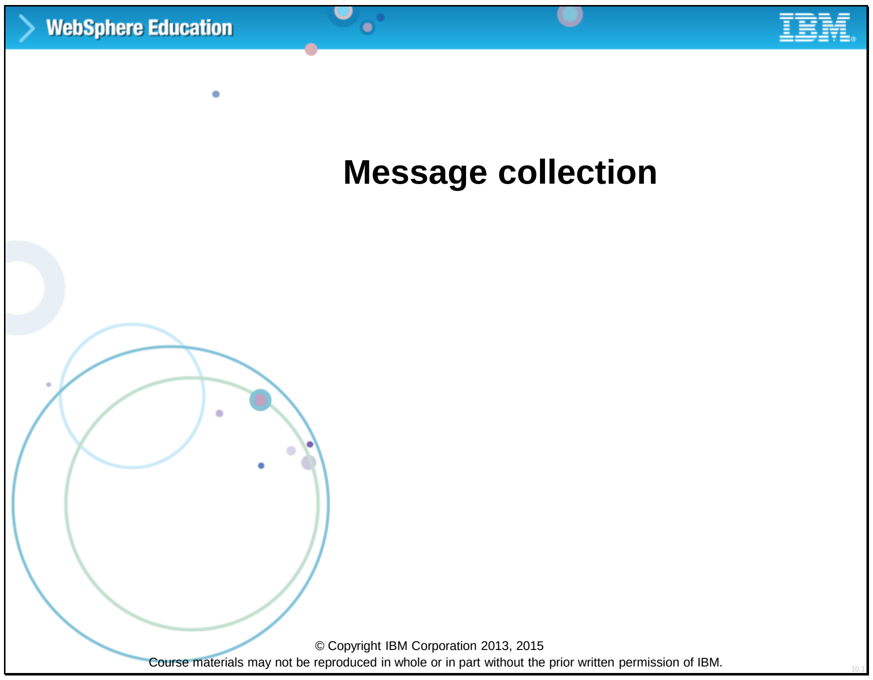
Use case:

If you create an aggregation flow that books a business trip, and you might have some requests that need a reply within two days. However, more urgent requests need a reply within 2 hours.

Aggregation implementation tips (2 of 2)

In some implementations, it might be advantageous to use more than one Aggregate Control node in a message flow to allow for different timeouts. In this implementation, you use the same aggregation name for both Aggregate Control nodes.

Slide 18



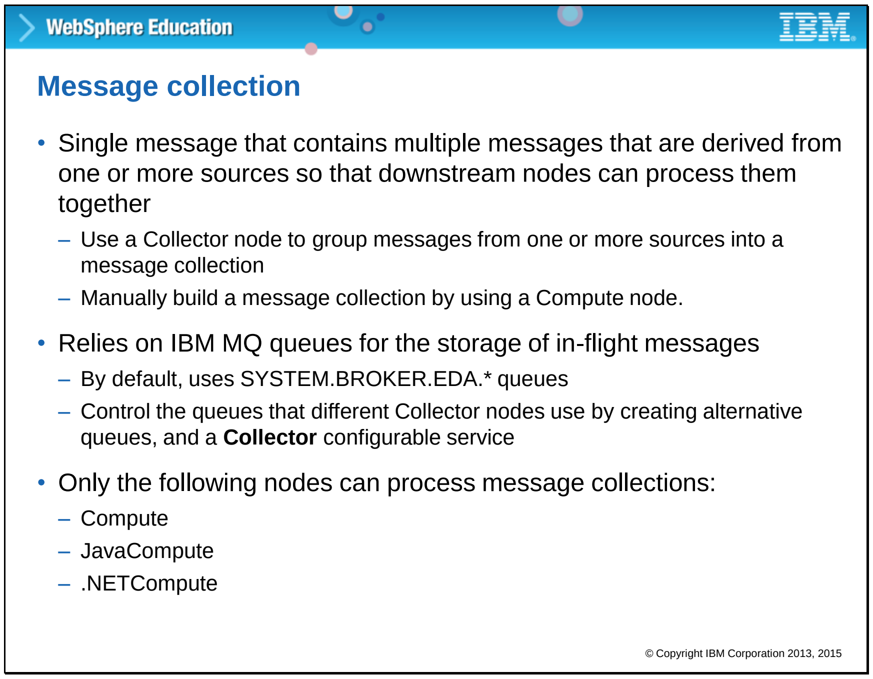
WebSphere Education

Message collection

Topic 2: Message collection

In this topic, you learn how to use a Collector node to group messages from one or more sources into a message collection.

Slide 19


WebSphere Education

Message collection

- Single message that contains multiple messages that are derived from one or more sources so that downstream nodes can process them together
  – Use a Collector node to group messages from one or more sources into a message collection
  – Manually build a message collection by using a Compute node.
- Relies on IBM MQ queues for the storage of in-flight messages
  – By default, uses SYSTEM.BROKER.EDA.* queues
  – Control the queues that different Collector nodes use by creating alternative queues, and a **Collector** configurable service
- Only the following nodes can process message collections:
  – Compute
  – JavaCompute
  – .NETCompute

Message collection

A message collection is a message that contains multiple messages that are derived from one or more sources. For example, you might need to extract, combine, and transform information from three different sources. The messages from these different sources might arrive at the input terminals at different times and in an unknown order.

Message collections can be created in a message flow by using a Collector node or manually by using a Compute node.

In a Collector node, a collection is defined by configuring an event handler for each input terminal.

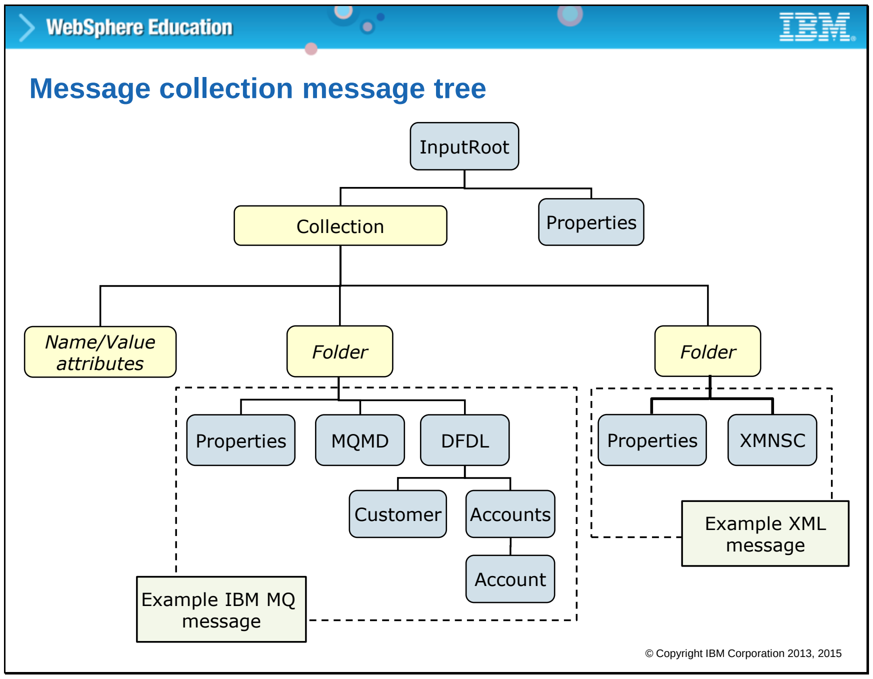
Information about the state of in-flight messages is held on SYSTEM.BROKER.EDA storage queues that MQ controls. If you want to use the capabilities that are provided by the Collector node, you must install MQ on the same computer as your integration node and associate the integration node with the queue manager.

By default, the storage queues used by all Collector nodes are SYSTEM.BROKER.EDA.EVENTS and SYSTEM.BROKER.EDA.COLLECTIONS. You can control the queues that are used by different Collector nodes by creating alternative queues that contain a *QueuePrefix* variable, and by using a Collector configurable service to specify the names of those queues for storing events.

Similar to the Aggregation message tree, the collection message tree contains a folder for each message. In a message flow, use a Compute, JavaCompute, or .NETCompute node to process message collections.

The next slide provides an example of a collection message tree.

Slide 20



Message collection message tree

Message collection message tree

This slide shows an example of the logical message tree that the Collector node creates. The message tree in this example contains two messages, one message that is received from MQ, and one message from a file input source.

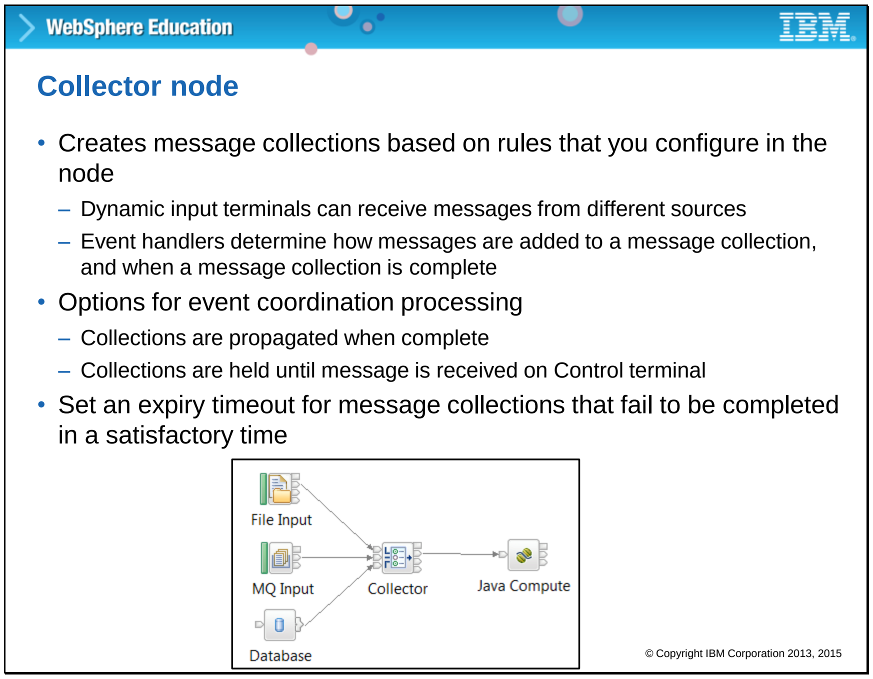
A message collection has a **Properties** header and a single folder element that is named **Collection**.

The Collection folder can have zero or more attributes that are name-value pairs; the name of an attribute must be unique within a message collection. The Collection folder also contains a folder for each message that is added to the message collection.

Each of these folders has a name, but this name does not have to be unique within the message collection. The value for Folder name is derived from the source of the input message.

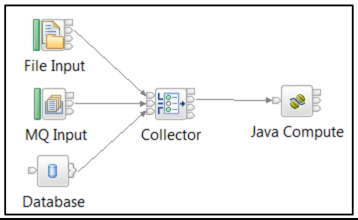
You can use ESQL or XPath expressions to access the content of messages in a message collection by referencing the folder names preceded by InputRoot.Collection.

Slide 21


WebSphere Education

Collector node

- Creates message collections based on rules that you configure in the node
  - Dynamic input terminals can receive messages from different sources
  - Event handlers determine how messages are added to a message collection, and when a message collection is complete
- Options for event coordination processing
  - Collections are propagated when complete
  - Collections are held until message is received on Control terminal
- Set an expiry timeout for message collections that fail to be completed in a satisfactory time



Collector node

The Collector node creates message collections that are based on rules that you define on the node.

The Collector node has one static input terminal, Control, and four static output terminals: Out, Expire, Failure, and Catch. These static terminals are always present on the node.

You can add and configure as many input terminals as required to the Collector node to control how the messages that are received on each input terminal are added to the appropriate message collection. You can also configure properties on the Collector node, which are known as event handlers, to determine how messages are added to a message collection, and when a message collection is complete.

You can use the Control terminal to trigger the output of completed message collections from the Collector node. The Collector node **Event Coordination** property specifies how messages received on the Control terminal for event coordination processing are handled in the Collector node.

- If you accept the default value of **Disabled**, messages to the Control terminal are ignored and collections are propagated when they are complete.

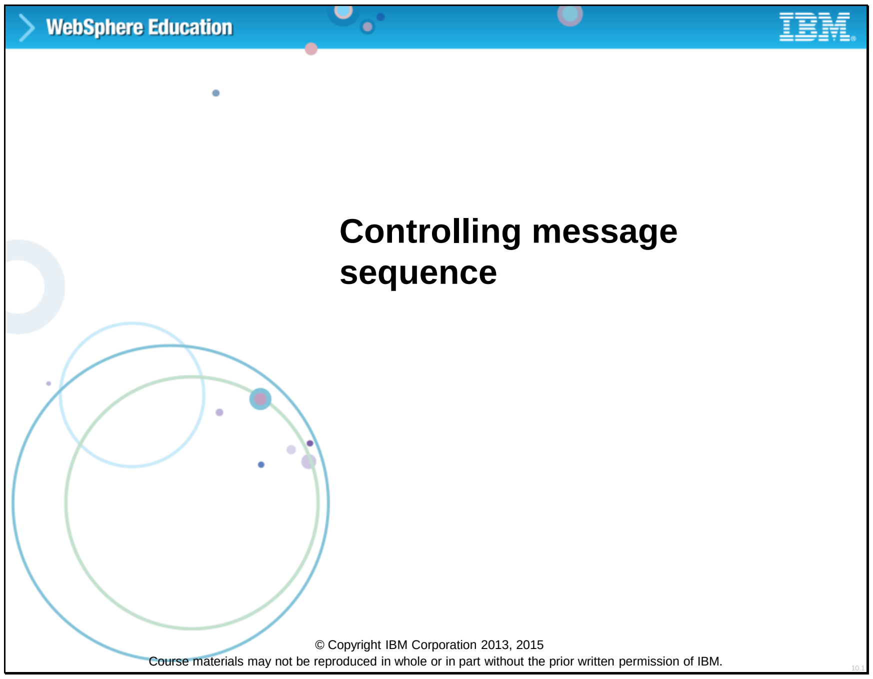
- If you select **All complete collections**, complete message collections are held on the SYSTEM.BROKER.EDA.COLLECTIONS queue. When a message is received on the Control terminal, all complete message collections on the MQ queue are propagated to the Out terminal.

- If you select **First complete collection**, complete message collections are held on the SYSTEM.BROKER.EDA.COLLECTIONS queue. When a message is received on the Control terminal, the first complete message collection on the queue is propagated to the Out terminal.

The collections are propagated in the same order that they become complete. If the MQ queue is empty when the message is received on the Control terminal, the next complete message collection is immediately propagated to the Out terminal. When a message collection is successfully completed, it is ready to be propagated to the Out terminal.

If a value greater than zero is set on the **Collection expiry** property, any incomplete message collections are propagated to the Expire terminal. Set an expiry timeout for message collections that fail to be completed in a satisfactory time.

Slide 22

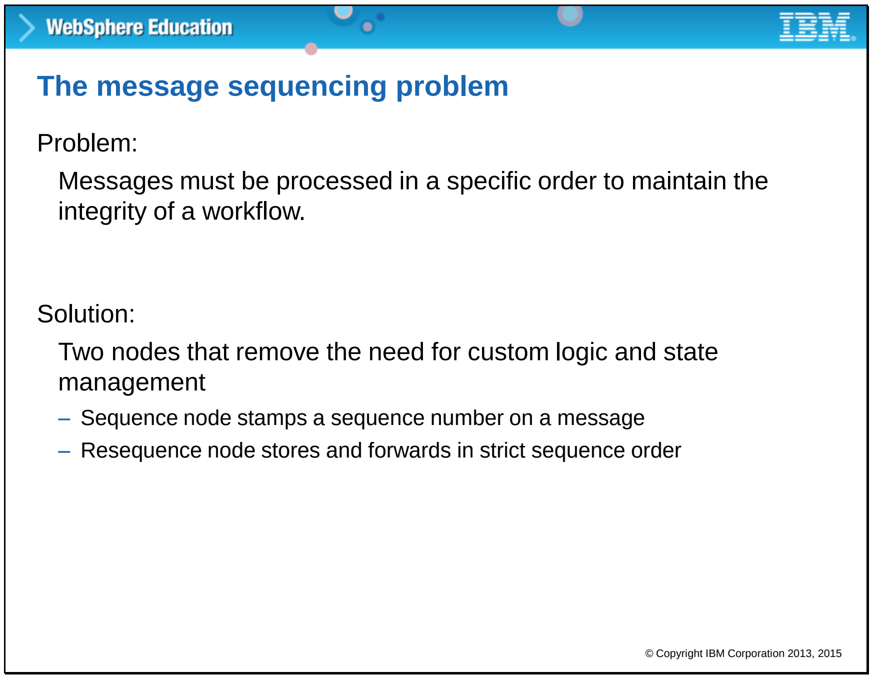

WebSphere Education

Controlling message sequence

Topic 3: Controlling message sequence

In this topic, you learn how to use the Sequence and Resequence message processing nodes to ensure that messages are delivered to the receiving application in a particular order.

Slide 23


WebSphere Education

The message sequencing problem

Problem:

Messages must be processed in a specific order to maintain the integrity of a workflow.

Solution:

Two nodes that remove the need for custom logic and state management

- Sequence node stamps a sequence number on a message
- Resequence node stores and forwards in strict sequence order

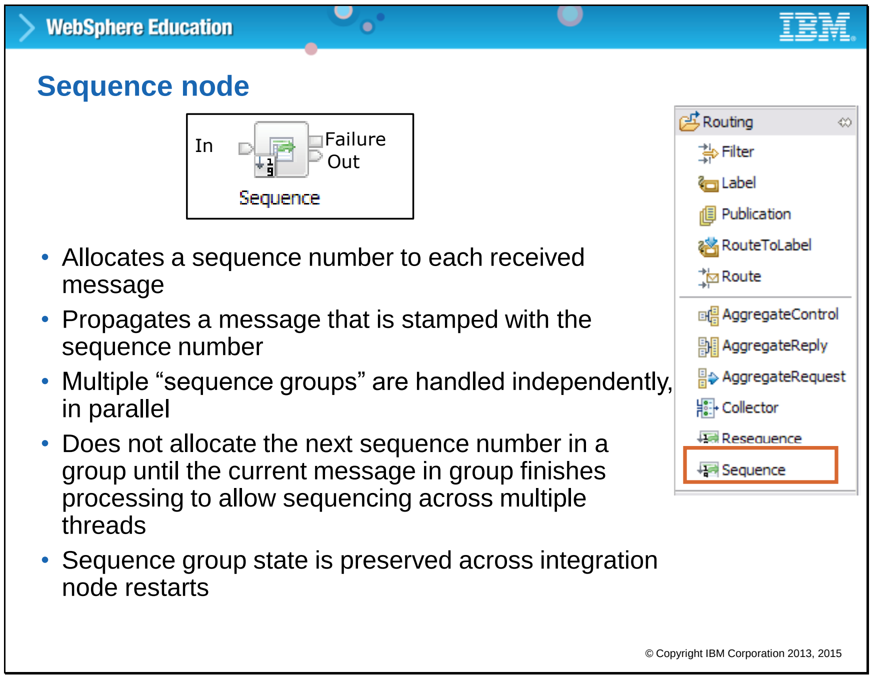
The message sequencing problem

Often, messages must be processed in a specific order. For example, a series of debit and credits against a bank account must be processed in the order they took place.

Integration Bus supports adding sequence numbers to messages, and for reordering messages in the message flow based on their sequence number. Messages can arrive in any order and you can use the Sequence and Resequence nodes to reorder the messages into the required sequence.

Information about the state of in-flight messages is held on storage queues that MQ controls. If you want to use the capabilities provided by the Sequence and Resequence nodes, you must install MQ on the same computer as the integration node.

Slide 24



Sequence node

- Allocates a sequence number to each received message
- Propagates a message that is stamped with the sequence number
- Multiple “sequence groups” are handled independently, in parallel
- Does not allocate the next sequence number in a group until the current message in group finishes processing to allow sequencing across multiple threads
- Sequence group state is preserved across integration node restarts

Sequence node

The Sequence node receives an input message, allocates a monotonically increasing sequence number from a user-defined start point, and stores the number in a user-defined location in the message assembly. It does this process for each message until the sequence ends, which a user-defined condition determines.

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

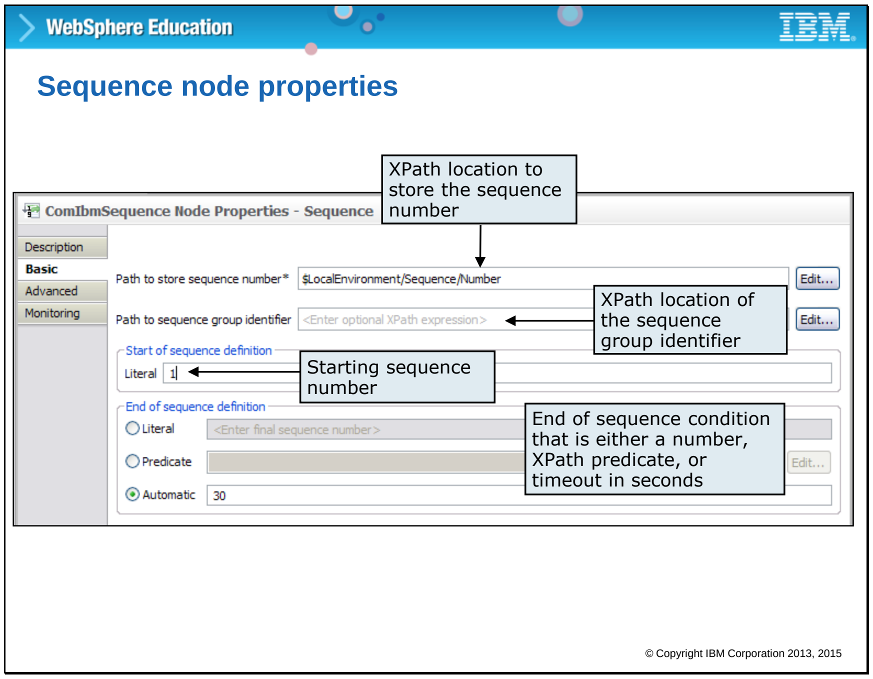
The sequencing applies to messages within the same sequence group only. You can use properties in the Sequence node to organize messages into groups according to a specified condition, for example, grouping all messages with the same value in a customer number field in the message. If no sequence group is specified, a single default group is used for all messages.

The Sequence node can process multiple sequence groups in parallel, but it processes only one request at a time for sequence numbers from the same sequence group.

The Sequence node allocates a sequence number to each message in a sequence group, and the next sequence number in the group is not allocated until the current message in the group finished processing either by being committed or rolled back.

As an option, sequence numbers and state information can be stored as persistent messages so that the sequence survives across integration node and queue manager restarts.

Slide 25


WebSphere Education

Sequence node properties

ComIbmSequence Node Properties - Sequence

Description | Basic | Advanced | Monitoring

Path to store sequence number* $LocalEnvironment/Sequence/Number

Path to sequence group identifier <Enter optional XPath expression>

Start of sequence definition

Literal 1

End of sequence definition

Literal <Enter final sequence number>

Predicate

(●) Automatic 30

XPath location to store the sequence number

XPath location of the sequence group identifier

Starting sequence number

End of sequence condition that is either a number, XPath predicate, or timeout in seconds

Sequence node properties

The slide shows an example of the Sequence node **Basic** properties.

The **Path to store sequence number** property is an XPath expression that specifies the location in which to save the sequence number of the message.

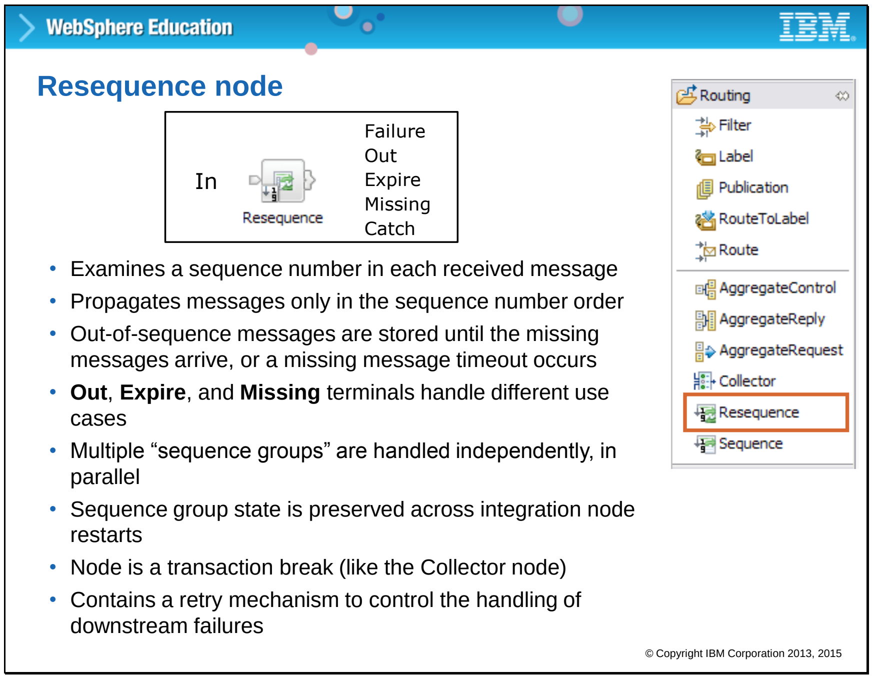
The **Path to sequence group identifier** property is an XPath expression that points to the location of the sequence group identifier. Messages that have the same group identifier are considered part of the same sequence group.

The **Start of sequence definition** property specifies the first sequence number in each group. Valid values are positive or negative integers.

A literal number, an XPath predicate that evaluates to “true” or “false”, or the time to wait for the next message identifies the end of the sequence.

If you do not want to use the default queues, the **Advanced** tab properties control message persistence and specify a configurable service for the storage queues.

Slide 26



Resequence node

- Examines a sequence number in each received message
- Propagates messages only in the 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 handle different use cases
- Multiple “sequence groups” are handled independently, in parallel
- Sequence group state is preserved across integration node restarts
- Node is a transaction break (like the Collector node)
- Contains a retry mechanism to control the handling of downstream failures

Resequence node

The Resequence node receives an input message and checks a sequence number in a user-defined location in the message assembly. If it is the next message in the sequence, it propagates the message to the **Out** terminal.

If an out-of-sequence message arrives, the node stores the message until the missing messages arrive. A missing message timeout can be specified to prevent the node from waiting indefinitely 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 are propagated to the **Expire** terminal.

After a sequence starts, as determined by a user-defined condition, resequence processing continues for each message until the sequence ends, which a user-defined condition also determines.

Similar to the Sequence node, messages can be divided into independent sequence groups. Each group has its own sequence number and is handled independently, enabling the parallel processing of groups.

If an out-of-sequence message arrives, the node stores the message until the missing messages arrive. For example, #1 arrives and is propagated. #3 arrives and is stored. #4 arrives and is stored. #2 arrives and is propagated, followed by #3 and then #4.

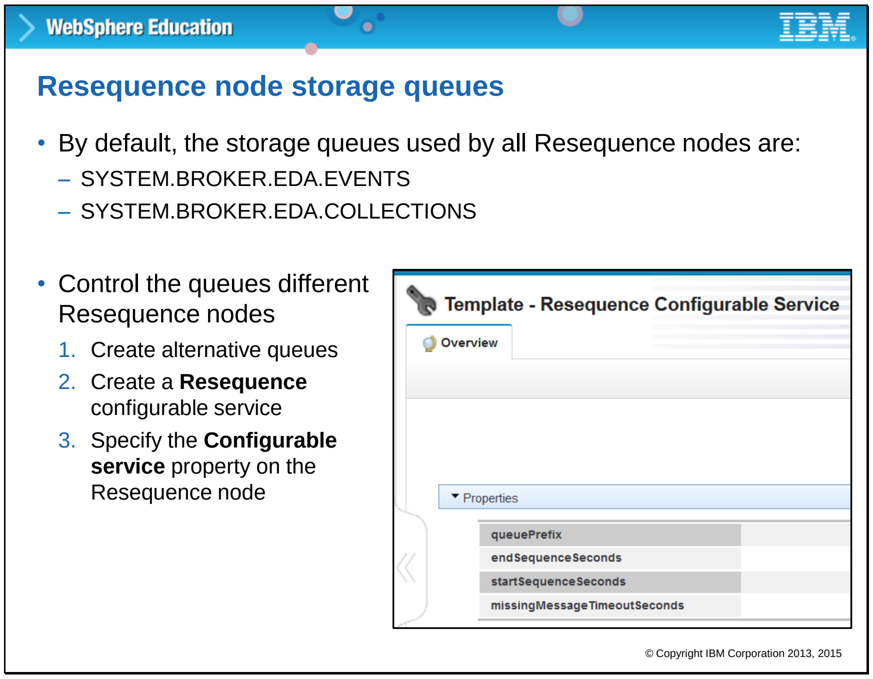
The sequence group state can be persisted so that the sequence survives across queue manager restarts.

The Resequence node uses the same architecture as the Collector node. It causes a transaction break, and all propagated messages are in a new unit of work. The node has a **Catch** terminal to handle any downstream exceptions.

A **Retry mechanism** property specifies how the Resequence node handles a flow failure.

- If the **Failure** option is selected and a message fails to propagate to the Out and Catch terminals, the message is propagated to the Failure terminal. The Resequence node is returned to a state where it is waiting for that failing message, which its sequence number identifies, to arrive again. Any subsequent messages in the sequence that are already received are not sent out until the message that failed is resent to the node.
- If the **Infinite** option is selected, any exceptions that occur in nodes that follow the Resequence node are rolled back to the Catch terminal of the Resequence node. If the Catch terminal is not connected to any other nodes, the messages are redelivered to the original terminal. The messages are never backed out or discarded.

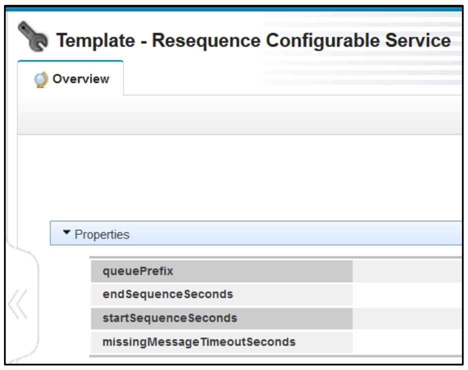
Slide 27



WebSphere Education

Resequence node storage queues

- By default, the storage queues used by all Resequence nodes are:
  - SYSTEM.BROKER.EDA.EVENTS
  - SYSTEM.BROKER.EDA.COLLECTIONS
- Control the queues different Resequence nodes
  1. Create alternative queues
  2. Create a **Resequence** configurable service
  3. Specify the **Configurable service** property on the Resequence node


Template - Resequence Configurable Service
Overview
Properties
queuePrefix
endSequenceSeconds
startSequenceSeconds
missingMessageTimeoutSeconds

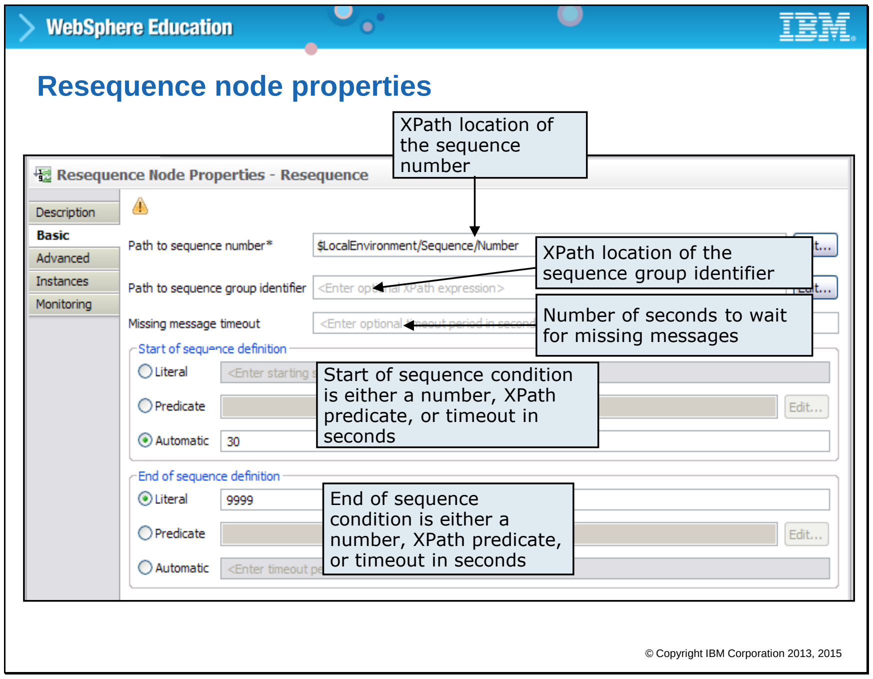
Resequence node storage queues

Information about the state of in-flight messages is held on storage queues that the MQ queue manager controls. By default, the storage queues used by all Resequence nodes are: SYSTEM.BROKER.EDA.EVENTS and SYSTEM.BROKER.EDA.COLLECTIONS.

You can control the queues that are used by different Resequence nodes by creating alternative queues and by using a Resequence configurable service to specify the names of those queues for storing events.

If you do not want to use the default storage queues, the **Advanced** tab properties on the **Resequence** node are used to specify a configurable service for the Resequence queues.

Slide 28


WebSphere Education

Resequence node properties

Resequence Node Properties - Resequence

Path to sequence number*: $LocalEnvironment/Sequence/Number

Path to sequence group identifier: <Enter optional XPath expression>

Missing message timeout: <Enter optional timeout period in secon

Start of sequence definition: Literal / Predicate / [✓ Automatic] 30

End of sequence definition: [✓ Literal] 9999 / Predicate / Automatic

XPath location of the sequence number

XPath location of the sequence group identifier

Number of seconds to wait for missing messages

Start of sequence condition is either a number, XPath predicate, or timeout in seconds

End of sequence condition is either a number, XPath predicate, or timeout in seconds

Resequence node properties

The figure shows the Resequence node **Basic** properties.

Similar to the Sequence node, the **Path to store sequence number** property is an XPath expression that specifies the location to which the sequence number of the message is saved.

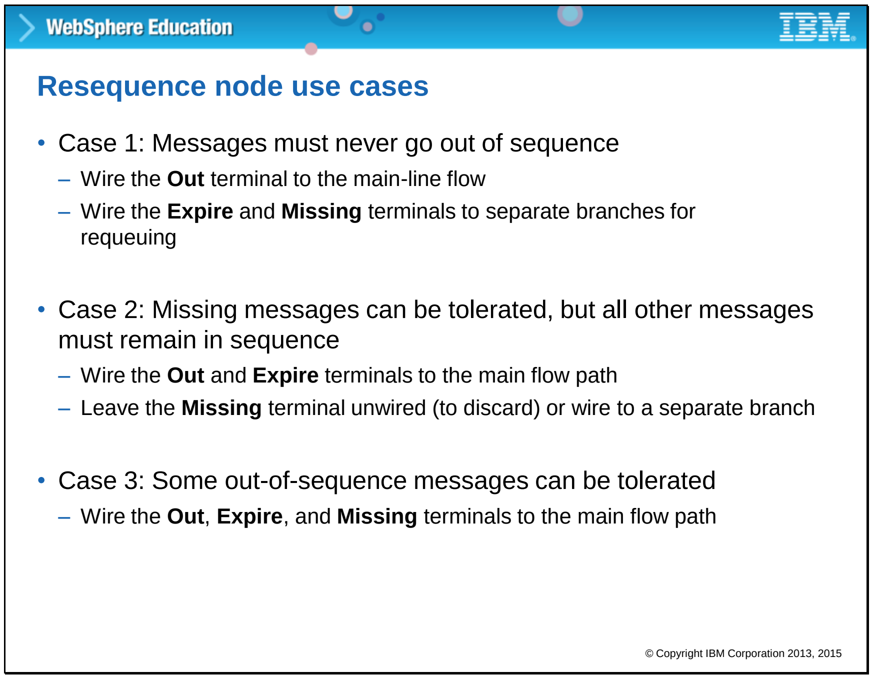
The **Path to sequence group identifier** property is an XPath expression that points to the location of the sequence group identifier.

You can specify the start of sequence by providing a literal number, an XPath predicate that evaluates to true or false, or a time to wait. If a wait time is specified, the node stores all messages until the timer expires. Then, it uses the lowest number as the start of the sequence.

You can also use a literal number, an XPath predicate that evaluates to true or false, or the time to wait for the next message to indicate the end of the message sequence.

If a missing message creates a gap between the previously propagated message and the current message, the sequence numbers of the missing messages are stored in the LocalEnvironment.Sequence.Missing location with one entry for each missing number.

The **Instances** tab allows the Resequence node to request extra threads from the message flow’s thread pool, or to allocate a separate thread pool for its own use.



WebSphere Education

Resequence node use cases

- Case 1: Messages must never go out of sequence
  - Wire the **Out** terminal to the main-line flow
  - Wire the **Expire** and **Missing** terminals to separate branches for requeuing
- Case 2: Missing messages can be tolerated, but all other messages must remain in sequence
  - Wire the **Out** and **Expire** terminals to the main flow path
  - Leave the **Missing** terminal unwired (to discard) or wire to a separate branch
- Case 3: Some out-of-sequence messages can be tolerated
  - Wire the **Out**, **Expire**, and **Missing** terminals to the main flow path

Resequence node use cases

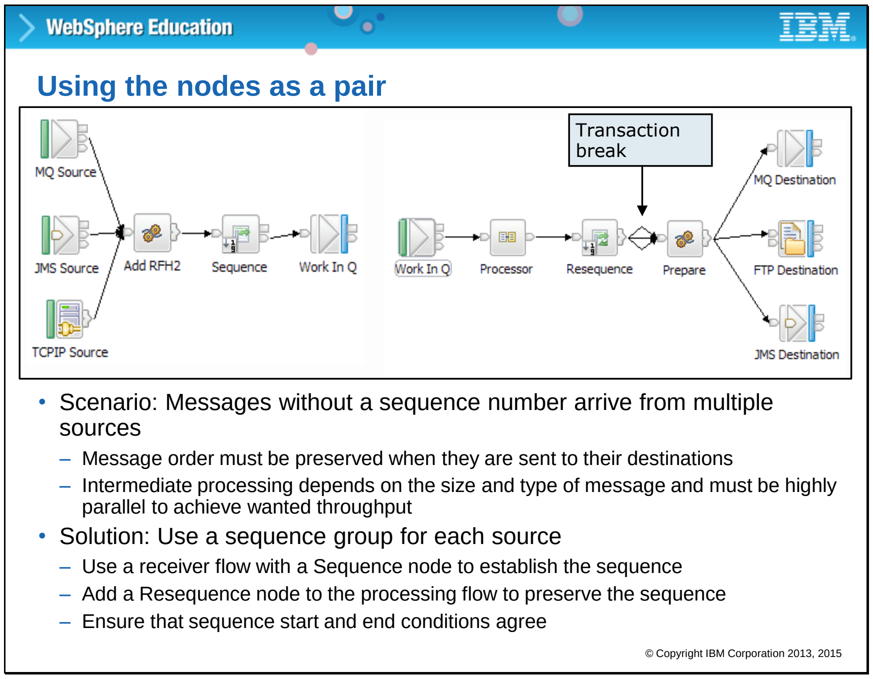
This slide lists some examples of when the Resequence nodes would be used.

Case 1: Messages must never go out of sequence. If a message is 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 flow and the **Expire** and **Missing** terminals to separate branches (most likely for requeuing).

Case 2: Missing messages can be tolerated, but all other messages must remain in sequence. If a message is 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 processing. To configure this behavior, wire the **Out** and **Expire** terminals to the main flow and leave the **Missing** terminal unwired (to discard the message). You can also wire the **Missing** terminal to a separate branch.

Case 3: Some out-of-sequence messages can be tolerated.
The flow must process all of the messages, preferably in sequential order, but some messages can be skipped and processed later to not delay the flow for too long. To configure this behavior, wire the **Out**, **Expire**, and **Missing** terminals into the main flow.

Slide 30



Using the nodes as a pair

- Scenario: Messages without a sequence number arrive from multiple sources
  - Message order must be preserved when they are sent to their destinations
  - Intermediate processing depends on the size and type of message and must be highly parallel to achieve wanted throughput
- Solution: Use a sequence group for each 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 that sequence start and end conditions agree

Using the nodes as a pair

The message flows in the figure shows how the Sequence and Resequence nodes can be used together.

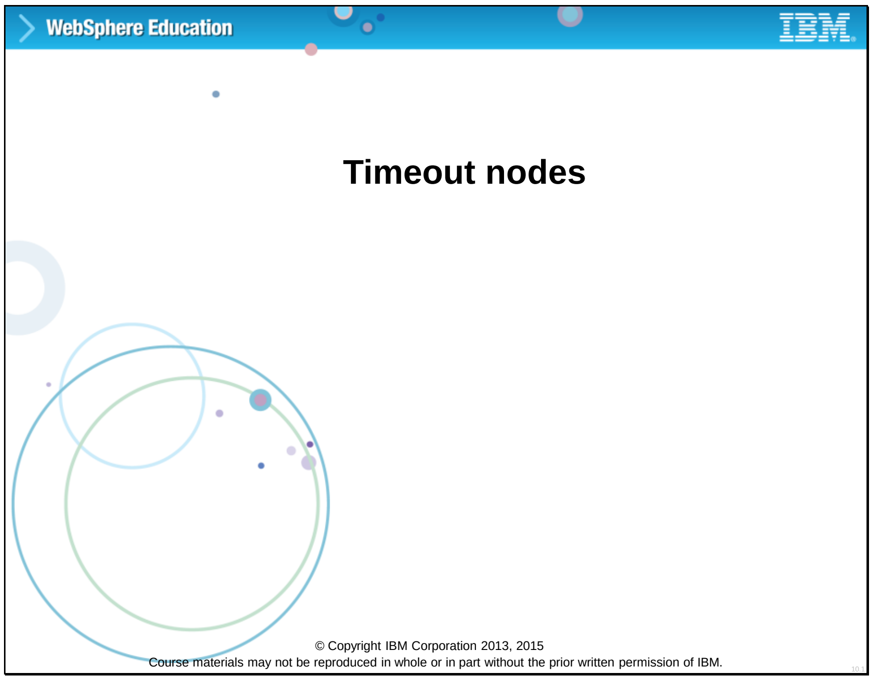
In this example, the Sequence node establishes the sequence that is based on the order of arrival. The Resequence node reestablishes the sequence in case it is out-of-step due to the length of time that was taken during the main processing phase of the flow, which is shown as a subflow.

The Resequence node is a transaction break. It takes three units of work to process a message in this scenario. You can use separate thread pools for the left side and right side of the processing flow to enable the wanted throughput without starving the right side.

In the example, the Resequence node **Out**, **Expire**, and **Missing** terminals are all wired to the next node in the flow for convenience only.

The Sequence and Resequence nodes must detect the start and end of the sequence together. One way to synchronize the nodes is for the Resequence node to use the LocalEnvironment.Sequence.Start and LocalEnvironment.Sequence.End flags that the Sequence node sets.

Slide 31



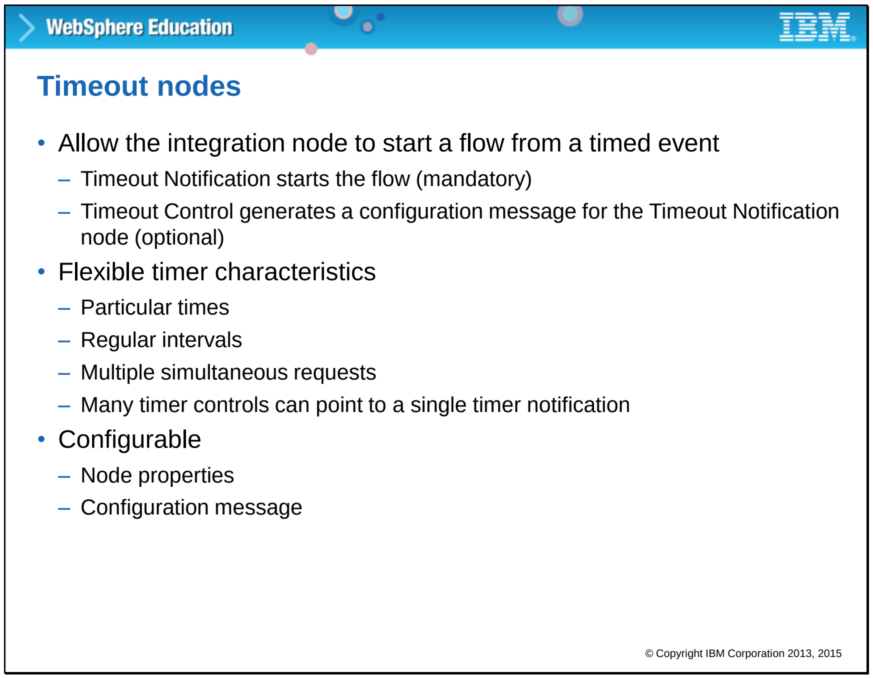
WebSphere Education

Timeout nodes

Topic 4: Timeout nodes

In this topic, you learn how to use the time-sensitive nodes to trigger flows to run at specific times or at fixed intervals.

Slide 33


WebSphere Education

Timeout nodes

- Allow the integration node to start a flow from a timed event
  - Timeout Notification starts the flow (mandatory)
  - Timeout Control generates a configuration message for the Timeout Notification node (optional)
- Flexible timer characteristics
  - Particular times
  - Regular intervals
  - Multiple simultaneous requests
  - Many timer controls can point to a single timer notification
- Configurable
  - Node properties
  - Configuration message

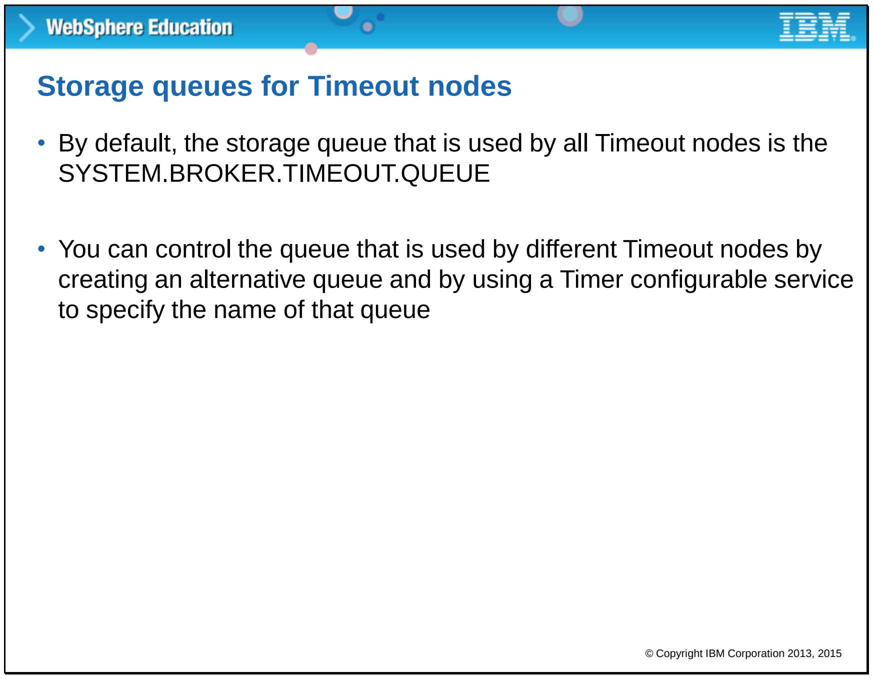
Timeout nodes

You can use the time-sensitive nodes to run message flow applications to run at specific times, or at fixed intervals, and act if transactions are not completed within a defined time.

The time-sensitive nodes are the Timeout Notification and the Timeout Control nodes. The Timeout Notification node starts a flow. The Timeout Control node generates a configuration message for the Timeout Notification node.

Time-sensitive nodes provide a flexible, self-contained mechanism for starting a message flow. No external coordination to the integration node is required beyond what is normally required to deploy a flow. Built-in integration node tracing mechanisms can be used to detect the failure to set the timers.

Slide 33


WebSphere Education

Storage queues for Timeout nodes

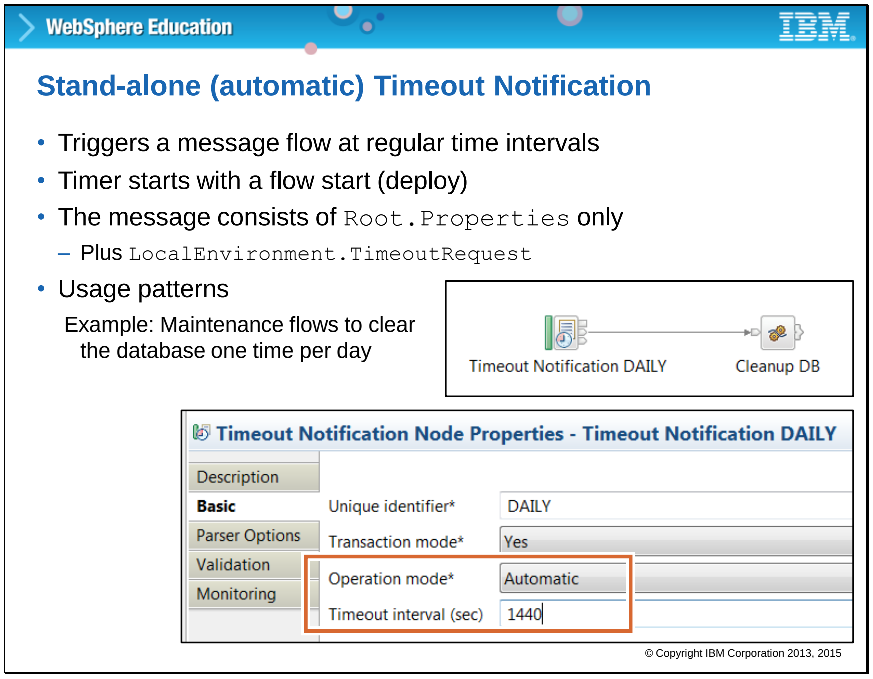
- By default, the storage queue that is used by all Timeout nodes is the SYSTEM.BROKER.TIMEOUT.QUEUE
- You can control the queue that is used by different Timeout nodes by creating an alternative queue and by using a Timer configurable service to specify the name of that queue

Storage queues for Timeout nodes

If you want to use the capabilities provided by the TimeoutControl and TimeoutNotification nodes, information about the state of in-flight messages is held on storage queues that MQ controls, so you must install it on the same computer as your integration node.

By default, the storage queue that is used by all timeout nodes is the SYSTEM.BROKER.TIMEOUT.QUEUE. You can control the queues that are used by different timeout nodes by creating alternative queues that contain a *QueuePrefix* variable, and by using a Timer configurable service to specify the names of those queues for storing events.

Slide 34


WebSphere Education

Stand-alone (automatic) Timeout Notification

- Triggers a message flow at regular time intervals
- Timer starts with a flow start (deploy)
- The message consists of `Root.Properties` only
  - Plus `LocalEnvironment.TimeoutRequest`
- Usage patterns

  Example: Maintenance flows to clear the database one time per day

Timeout Notification Node Properties - Timeout Notification DAILY

Unique identifier*	DAILY
Transaction mode*	Yes
Operation mode*	Automatic
Timeout interval (sec)	1440

Stand-alone (automatic) Timeout Notification

The Timeout Notification node is an input node that can be paired with one or more Timeout Control nodes or used as a stand-alone node.

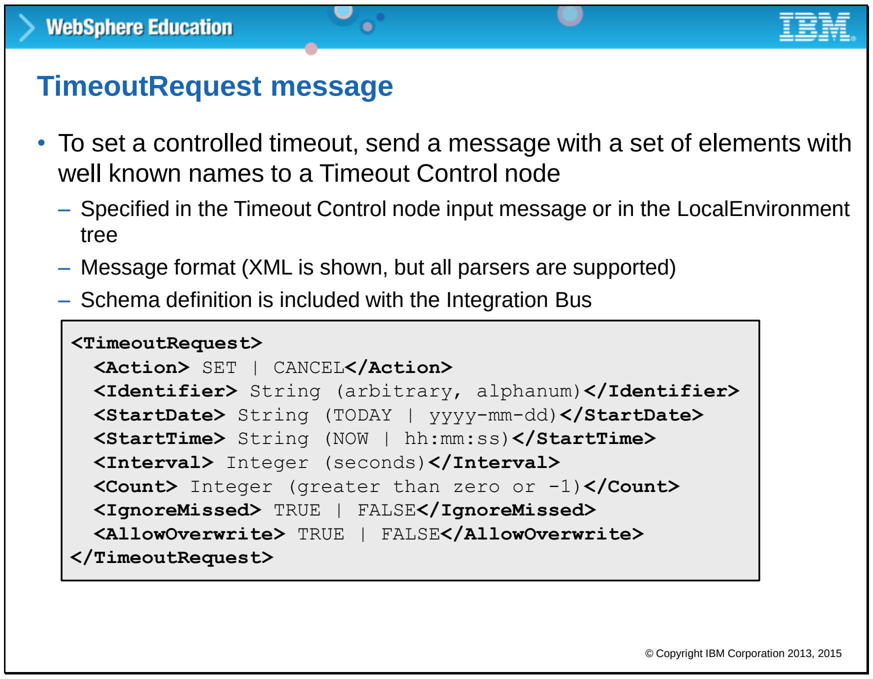
As a stand-alone node, generated messages are propagated to the next node in the message flow at time intervals that are specified in the configuration of this node. So, as an example, you can have a message flow that runs every 30 seconds, or every hour, or every 17 years.

Little flexibility exists when only a Timeout Notification node is used in a message flow. The node begins timing as soon as the message flow is deployed. The first timer event occurs as soon as the flow is deployed. You cannot stop the timer (short of stopping the message flow itself), nor is there any way to dynamically change the time interval. Also, if a Timeout Notification node specifies an event to occur every hour, the event happens one time an hour after the deployment occurred. So, if the flow was deployed at 6:17 AM, then events occur at 6:17 AM, 7:17 AM, 8:17 AM, and every hour thereafter.

This stand-alone scenario would most likely occur if you needed an integration node flow to run for a maintenance purpose, such as processing or clearing rows that accumulated in a database table. Generally, it is not important exactly when these maintenance procedures occur, if they occur on a periodic basis.

The Timeout nodes create or use a special Timeout Request structure, which is described next.

Slide 35


WebSphere Education

TimeoutRequest message

- To set a controlled timeout, send a message with a set of elements with well known names to a Timeout Control node
  - Specified in the Timeout Control node input message or in the LocalEnvironment tree
  - Message format (XML is shown, but all parsers are supported)
  - Schema definition is included with the Integration Bus

```
<TimeoutRequest>
  <Action> SET | CANCEL</Action>
  <Identifier> String (arbitrary, alphanum)</Identifier>
  <StartDate> String (TODAY | yyyy-mm-dd)</StartDate>
  <StartTime> String (NOW | hh:mm:ss)</StartTime>
  <Interval> Integer (seconds)</Interval>
  <Count> Integer (greater than zero or -1)</Count>
  <IgnoreMissed> TRUE | FALSE</IgnoreMissed>
  <AllowOverwrite> TRUE | FALSE</AllowOverwrite>
</TimeoutRequest>
```

TimeoutRequest message

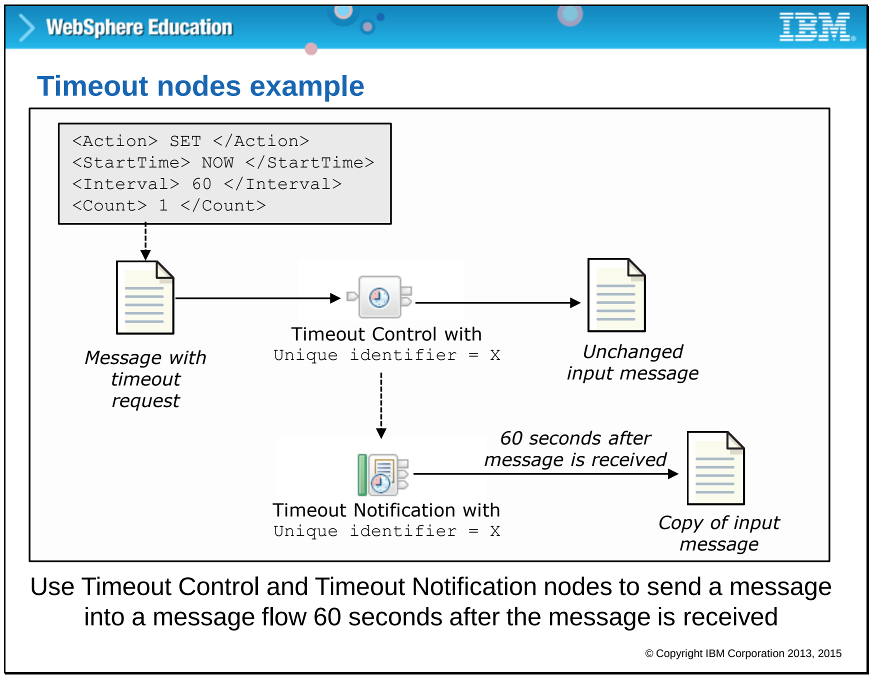
You can set a controlled timeout by sending a message with a set of elements with known names to a TimeoutControl node. These elements control the properties of the timeout to be created or deleted.

This slide shows an example of a TimeoutRequest message.

Set the Request Location on the TimeoutControl node to InputRoot.XML.TimeoutRequest to read these properties. If you want to obtain properties from a different part of your message, specify the appropriate correlation name for the parent element for the properties.

If your requirements are more complex than simple, fixed interval timer events, you must associate one or more TimeoutControl nodes with the TimeoutNotification node.

Slide 36


WebSphere Education

Timeout nodes example

```
<Action> SET </Action>
<StartTime> NOW </StartTime>
<Interval> 60 </Interval>
<Count> 1 </Count>
```

Message with timeout request
Timeout Control with Unique identifier = X
Unchanged input message
60 seconds after message is received
Timeout Notification with Unique identifier = X
Copy of input message

Use Timeout Control and Timeout Notification nodes to send a message into a message flow 60 seconds after the message is received

Timeout nodes example

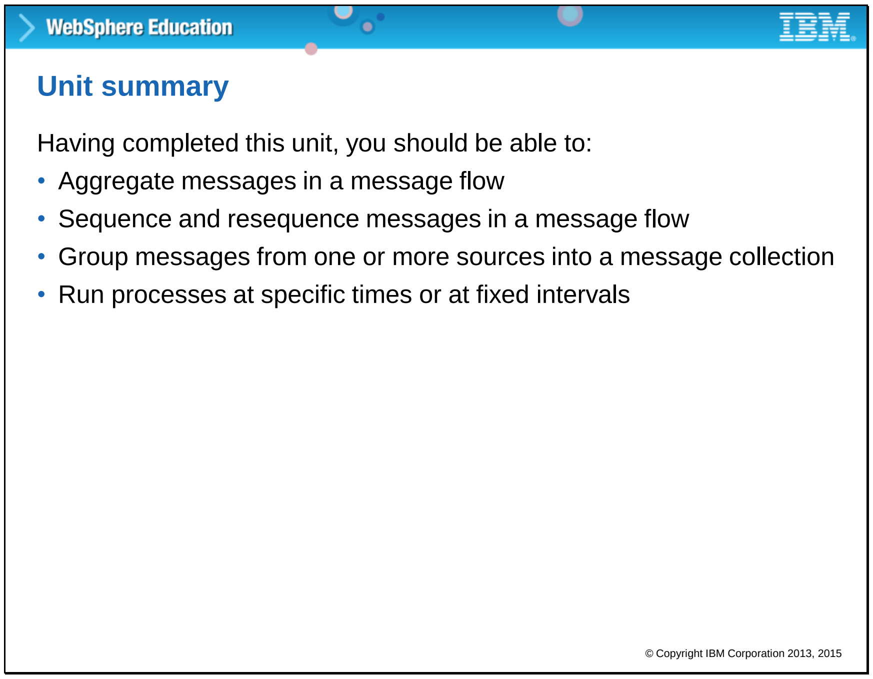
The example shows the path of a message that contains a timeout request through a Timeout Control node. A Timeout Notification node with an identifier that matches the Timeout Control node then processes the timeout request. The example also shows the message that the Timeout Notification node produces after processing the timeout request.

The message comes into the Timeout Control node with values set in the TimeoutRequest section of the message.

The Timeout Control node validates the TimeoutRequest; default values are assumed for properties that are not explicitly defined. The original message is then sent on to the next node in the message flow.

A property of the Timeout Notification node is its Unique Identifier, which is a 1 – 12-character label. If the TimeoutRequest is valid, the Timeout Notification node with the same **Unique identifier** as the Timeout Control node propagates a copy of the message to the message flow 60 seconds after the message was received. The **Unique Identifier** must be globally unique within the integration node instance in which this node is deployed.

Slide 37


WebSphere Education

Unit summary

Having completed this unit, you should be able to:
- Aggregate messages in a message flow
- Sequence and resequence messages in a message flow
- Group messages from one or more sources into a message collection
- Run processes at specific times or at fixed intervals

Unit summary

Event-driven message processing nodes control the flow of messages through message flows by using aggregation, message collections, message sequences, and timeout flows. In this unit, you learned how to aggregate and control the sequence of messages in a message flow. You also learned how to use time-sensitive nodes to control when processes run.

Having completed this unit, you should be able to:
- Aggregate messages in a message flow
- Sequence and resequence messages in a message flow
- Group messages from one or more sources into a message collection
- Run processes at specific times or at fixed intervals