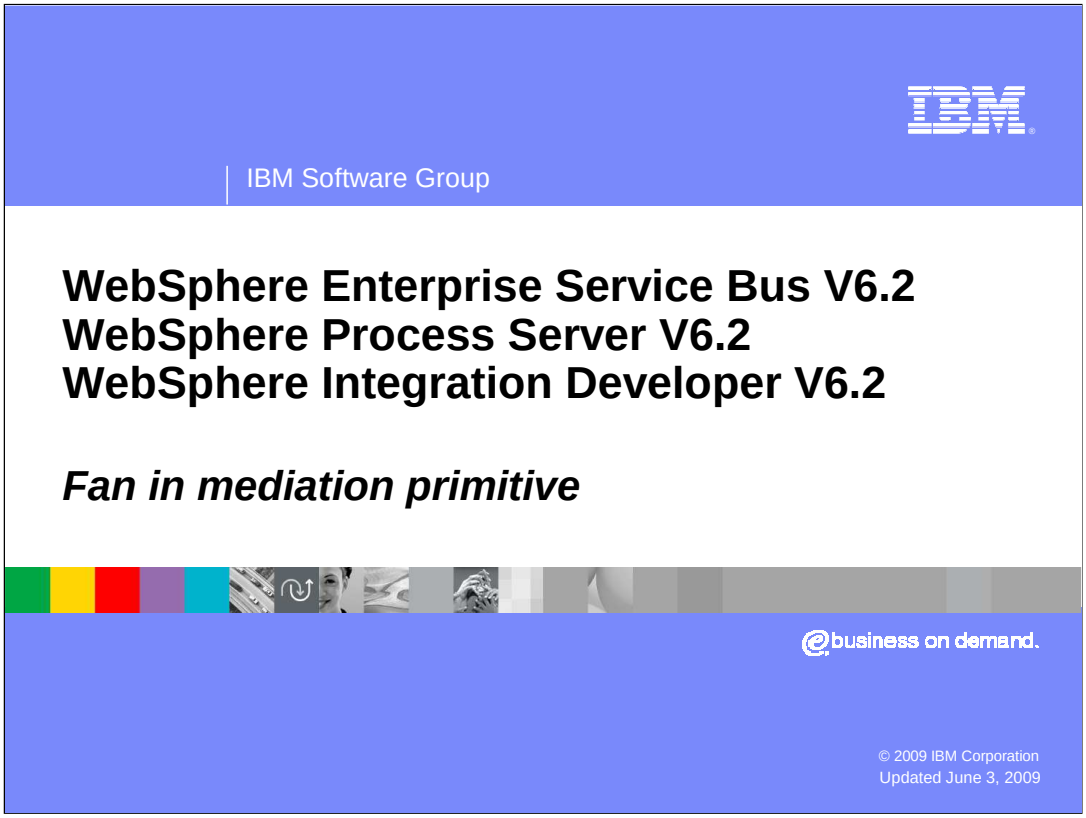


This presentation provides a detailed look at the fan in mediation primitive. You will have a better understanding of this presentation if you have already reviewed the fan out mediation primitive presentation.

Goals

- Understand the fan in mediation primitive

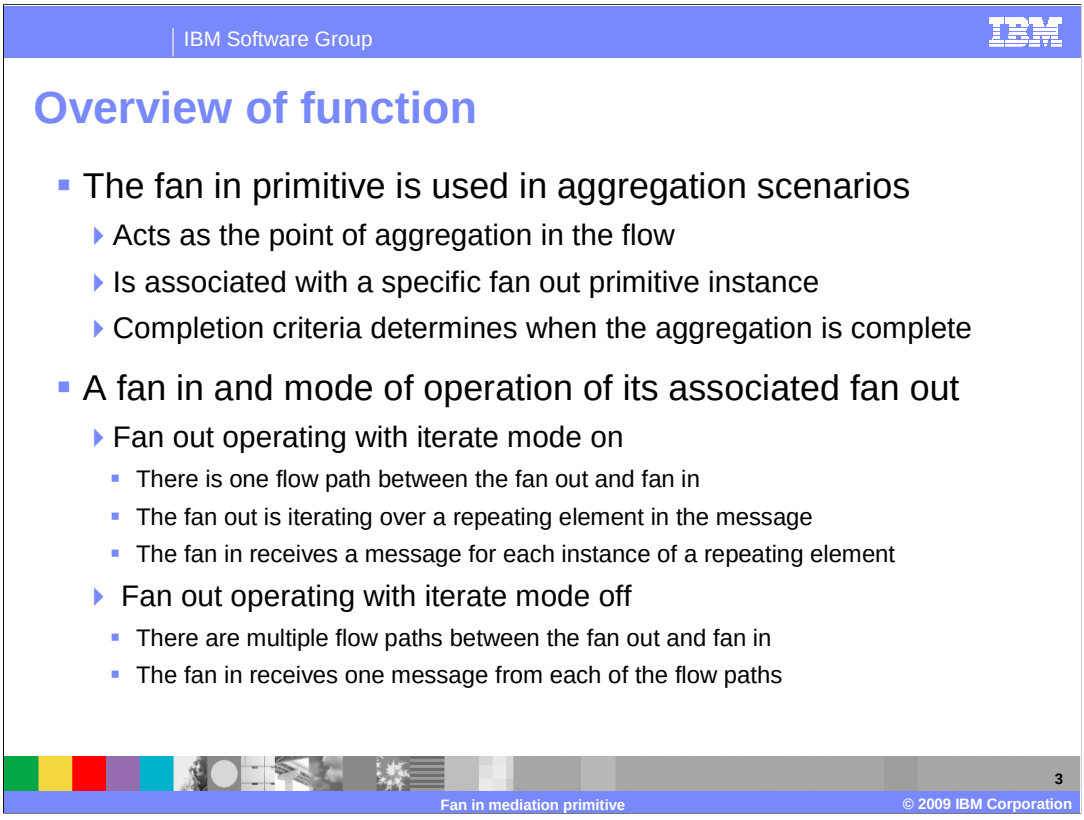
Fan in

- Overview of function
- Use of terminals
- Definition of properties
- Error handling
- Completion criteria and flow characteristics
- Behavior of timeout
- Details of usage scenarios

The goal of this presentation is to provide you with a full understanding of the fan in mediation primitive.

The presentation assumes that you are already familiar with the material presented in the presentations that cover common elements of all mediation primitives, such as properties, terminals, wiring and the use of promoted properties. The general knowledge of mediation primitives they provide is needed to understand the fan in primitive specific material in this presentation.

An overview of the fan in primitive is presented along with information about the primitive's use of terminals, its properties and some error handling considerations. There is a discussion of fan in completion criteria and the effects they have on a mediation flow. The behavior of a timeout for a fan in is explained and a couple of usage scenarios are presented.



The fan in primitive is a key element of mediation flows implementing aggregation scenarios. It provides the point of aggregation in the flow, bringing together multiple flow paths or serving as the end point of an iteration within a flow. Each instance of a fan in primitive is associated with a specific fan out primitive instance in the same flow. The fan in is configured with completion criteria that is used to determine if the flow will proceed from an output terminal of the fan in or return to the fan out to process another iteration or flow path.

Fan out primitives have two different modes of operation which affect the characteristics of the flow between the fan out and fan in primitives. In iterate mode the fan out iterates through a repeating element that is contained in the input message. The output terminal of the fan out is fired once for each element. The fan in receives an input message for each instance of the repeating element.

When iterate mode is off, the output terminal of the fan out is fired once. In this mode, the flow is constructed with multiple flow paths from the out terminal of the fan out which join back together at the fan in. Each flow path wired to the fan out primitive's out terminal is driven sequentially and the fan in receives the result of each flow path as input.

Overview of function

- A fan in is configured with completion criteria
- Completion criteria affects overall flow path
  - Flow does not continue out of the fan in until completion criteria is met
    - Flow remains between the fan out and fan in, processing messages from the fan out
    - After completion the flow continues following the fan in
- Completion criteria is configured with properties
  - **Count** – set number of messages received at fan in
  - **XPath** – evaluation of XPath expression
  - **Iterate** – waits until it receives all messages from the fan out in iterate mode

A fan in primitive is configured with completion criteria, which is very important for defining the overall flow path that will occur when the flow is run. There are two major divisions in the flow, the part that occurs between the fan out and fan in primitives and the part that proceeds after the fan in primitive. When a message reaches the fan in, if the completion criteria is not satisfied, the flow will return to the fan out to process the next iteration or flow path. If the completion criteria is met, the flow proceeds following the path after the fan in.

There are three mutually exclusive options for specifying the completion criteria. The first is a count of the messages received at the fan in, with completion occurring when a pre-defined number of messages has been received. The next is the specification of an XPath expression that is evaluated and completion occurs when the expression is true. The final option, which is only valid if the fan out is in iterate mode, is for completion to occur after all the repeating elements have been processed.

Contexts used in flows with fan in

- Iterative aggregation scenarios and the FanOutContext
  - The FanOutContext is used by iterative flows between a fan out and fan in
  - It contains the current array element being processed
  - Located in the SMO at context/primitiveContext/FanOutContext
- Aggregation scenarios and the shared context
  - The shared context is defined by a business object (similar to transient and correlation contexts)
  - Flows between a fan out and fan in set values to be aggregated into the shared context
  - After the fan in completes, subsequent primitives use the contents of the shared context to build the aggregated message

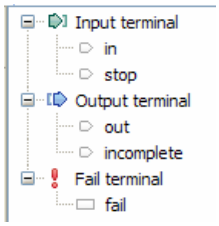
Although not used by the fan in primitive itself, there are two contexts that are important design elements of aggregation flows that use a fan in primitive.

The first is the FanOutContext, which is used in iterative aggregation scenarios. The context is initialized by the fan out primitive and contains a copy of the current array element being processed. This is used by the primitives between the fan out and fan in to address the current array element. It is located in the SMO at context/primitiveContext/FanOutContext.

The next is the shared context, which is used by all aggregation scenarios, both iterative and multi-path flows. You define the contents of the shared context using a business object. In the flow between the fan out and fan in, the aggregated data is build up in the shared context. After the fan in completes, the aggregated data is transformed from the shared context into the message body.

Terminals

- Terminals:
  - Two input terminals
  - Two output terminals
  - Fail terminal



- All terminals must be for the same message type

- Input terminals
  - in
    - Receives the input message
  - stop
    - Explicitly stops both fan out and fan in processing
    - Causes the incomplete terminal to be fired
- Output terminals
  - out
    - Fires when the fan in completion criteria is met
  - incomplete
    - Fires when a message arrives at the stop input terminal
    - Fires when a message arrives after the timeout delay
    - Fires when fan out has no more messages to send and completion criteria is not met

The fan in primitive has two input terminals, two output terminals and a fail terminal.

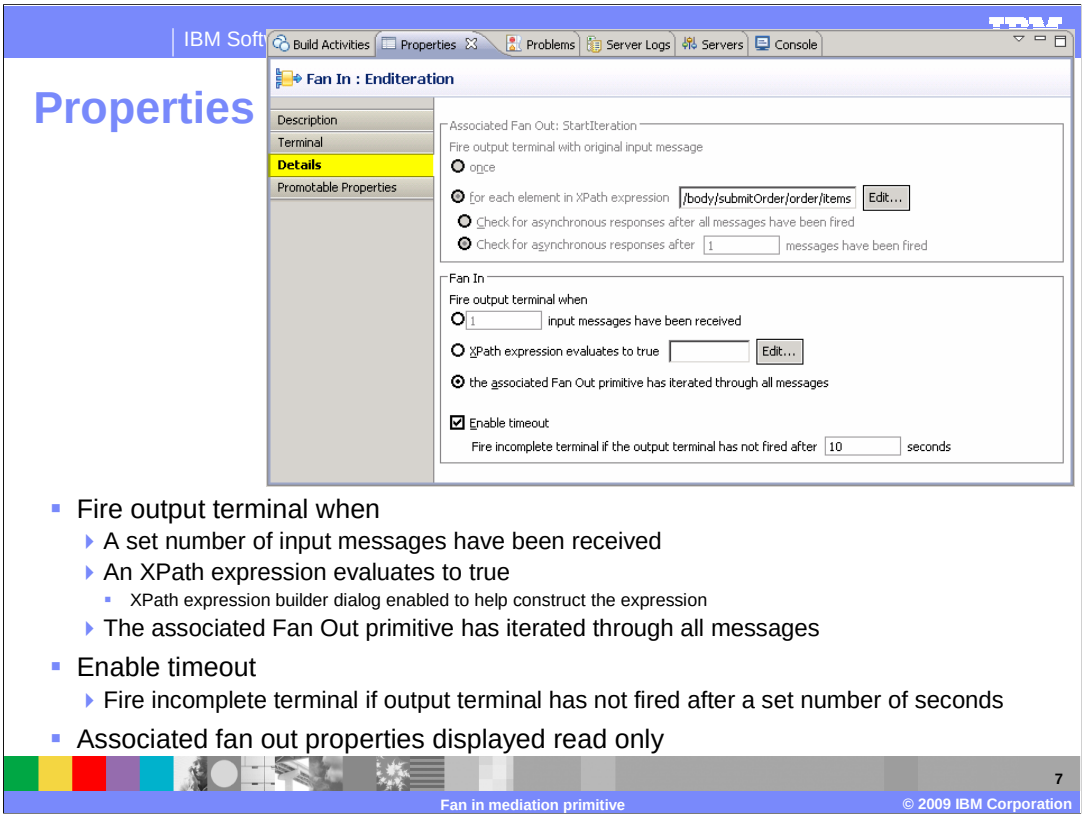
The fan in primitive is unique in that it is the only primitive that has multiple input terminals defined.

The first input terminal is the in terminal. It receives the input message for normal processing situations. For an iterative scenario this terminal normally has one incoming wire over which multiple messages arrive. For a multi-path scenario there are normally multiple messages flowing over multiple wires entering the fan in through this terminal. Depending upon the completion criteria, the flow will either return to the fan out or continue from one of its output terminals.

The next input terminal is the stop terminal which is used for exceptional conditions. When a message arrives at the stop terminal, the processing of the fan out and fan in pair is ended and the message is propagated through the incomplete output terminal.

The first output terminal, named out, is where the message received by the primitive is propagated down the flow. If the completion criteria for the fan in has been met, the message received is propagated through this terminal unchanged. The second output terminal is the incomplete terminal. There are three circumstances which result in this terminal being fired. The first instance is when a message is received through the stop input terminal. The next case is for any message that arrives at the in terminal after the timeout period has expired. The third case for this terminal being fired is when a message arrives at the in terminal, the completion criteria has not been met and the fan out has completed sending all messages.

All the terminals are for the same message type because the fan in primitive does not change anything in the SMO.



This is the properties panel for the fan in primitive. The first property is labeled “Fire output terminal when” and is a mutually exclusive choice between three different completion criteria.

The first choice is the count option which is labeled “n input messages have been received”, where n defines the number of incoming messages.

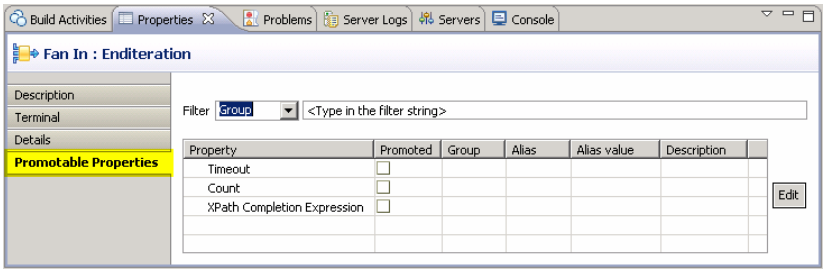
The second choice is the XPath option labeled “XPath expression evaluates to true” and allows you to specify an XPath expression that is evaluated whenever an incoming message is received by the fan in. It has an Edit... button that enables you to access the XPath expression builder dialog for defining the expression.

The third choice is the iterate option which is labeled “the associated Fan Out primitive has iterated through all messages.” This choice is not available if the associated fan out is not configured in iterate mode.

The next property is for specifying timeout, which you select to be either enabled or disabled. When enabled, you must specify the number of seconds that transpires between the first message fired from the fan out primitive and the occurrence of the timeout. The label states “Fire incomplete terminal if the output terminal has not fired after n seconds,” where n is the value you specify. This label is misleading in that it does not correctly reflect the processing that occurs. Once the timeout has occurred, the incomplete terminal is not automatically fired. It is only fired to propagate incoming messages that arrive at the fan in after the timeout period has expired.

You will notice on the properties panel that there is a section for the properties configured on the associated fan out. This section is read only, and is provided to make it easier for you to ensure you have a completion criteria for the fan in that is compatible with the configuration of the fan out.

Promotable properties



- Promotable
  - XPath completion expression
  - Count – number of input messages
  - Timeout – number of seconds
- Not Promotable
  - Fire output terminal when
  - Timeout enabled
- Promoted value only used if associated option is enabled

This slide shows the promotable properties. The properties that are not promotable are those that might require the flow logic to be altered, specifically the choice of completion criteria type and whether timeout is enabled. The properties that are promotable are the XPath expression, the count value and the timeout value. Promoting any of these only makes sense when they will actually be used. For example, if you had a fan in configured with an XPath expression as the completion criteria, then promoting that XPath expression makes sense. However, promoting the count value does not make sense as it is not used.

Error processing

- MediationBusinessException (fail terminal flow)
  - XPath expression with incorrect syntax
    - WebSphere® Integration Developer catches most of these
- XPath references element not found in SMO
  - Expression evaluates to false
  - Not considered an error
- Generally no runtime errors occur during a fan in
  - Configuration errors result in unexpected flow paths
  - Using trace aids in debugging the actual flow taken
    - Trace setting → com.ibm.ws.sibx.*=fine
    - Provides SMO logging before and after each mediation primitive
  - Using the visual debugger is another approach to seeing the actual flow

This slide looks at some of the possible error conditions and considerations for the fan in primitive. It is possible to get a MediationBusinessException if you provide an XPath expression which has an incorrect syntax, with the fail terminal flow taken if it is wired. However, because of the XPath validation checking done by WebSphere Integration Developer, it is unusual for an XPath with incorrect syntax to be configured and encountered at runtime.

It is possible for you to provide an XPath expression that is valid but that references an element in the SMO that does not exist during a particular instance of a flow. For a fan in primitive, this is not considered an error condition and the expression evaluates to false.

Because of the nature of processing done by the fan in, it is very unusual for there to be runtime errors that occur at the fan in primitive itself. It is more likely that an error in configuring the fan in results in unexpected flow paths being taken. If you encounter such a problem, one approach to debugging is to use trace to see what is actually happening in the flow and what paths are being taken. The appropriate trace setting is shown on the slide, and this will result in the SMO being dumped before and after every mediation primitive along with information about the primitive and the specific terminals involved in the flow. Another approach to debugging this is to step through the flow using the visual debugger.

Completion criteria and flow characteristics

- Completion criteria must be complementary to:
  - Configuration of the fan out
  - Design of the flow between fan out and fan in
  - Design of the flow following the fan in
- Firing of the fan in primitive's out terminal
  - Only happens when completion criteria is met
  - Typically occurs when the last message received from the fan out
- When completion criteria met before the fan out is finished
  - Flow proceeds following the fan in
  - When flow following the fan in is finished, fan out sends the next message
  - If using count, the count is reset
- Incomplete terminal fired when:
  - Completion criteria is not met by the message received at the fan in
  - The fan out has no more messages to send

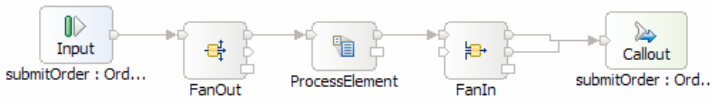
It is very important that the configuration of the fan out, the construction of the flow between the fan out and fan in and the completion criteria of the fan in complement each other. The next few slides discuss the relationship between the completion criteria and the flow characteristics so that you have an understanding of the expected behavior. With this knowledge, you can define your flows so that they behave in a well define way according to your requirements.

The first consideration is that the out terminal of the fan in primitive is fired only when the completion criteria is met. In most typical flow definitions, the completion criteria is designed so that it is met by the last message to arrive at the fan in.

It is possible for the completion criteria to be met before the last message has arrived at the fan in. When this happens, the flow proceeds from the out terminal of the fan in and continues until that flow path ends, such as reaching a callout node or stop primitive in the path. At that point, the fan out again receives control and sends either the next iterative message or initiates the flow on the next path wired to its out terminal. If the fan in primitive is configured with a count completion criteria, it is reset to start a new count.

The fan in incomplete terminal is fired in the case where the fan in receives a message, the completion criteria is not met and the fan out primitive has no more messages to send.

Completion criteria and flow characteristics



- Scenario
  - Fan out configured in iterate mode
  - Incoming SMO has six elements in the array
- Criteria: associated fan out has processed all elements
  - Six iterations between fan out and fan in
  - Message with six elements sent to the callout from the out terminal
- Criteria: count value = 3
  - Three iterations between fan out and fan in
  - Message with three elements sent to the callout from the out terminal
  - Three iterations between fan out and fan in
  - Message with three elements sent to the callout from the out terminal

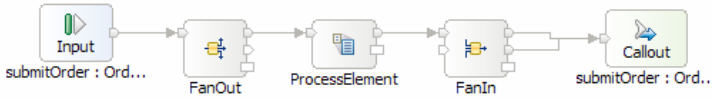
The next two slides examine completion criteria and the resulting flow behavior.

They use the same flow definition and input message but vary only in the completion criteria configured for the fan in. The scenario is a flow with an input node wired to a fan out configured in iterate mode. It is wired to a primitive that performs some element specific processing which then flows to a fan in. The fan in has both its out terminal and incomplete terminal wired to a callout node. Note that this flow definition is not sufficient for a real iteration scenario but has been simplified to help illustrate the flow behavior for various completion criteria. In the scenario, the incoming message has an array containing exactly six elements to be iterated over.

In the first case, the fan in is configured to complete when the associated fan out has processed all of the elements. In this case, there are six iterations that occur between the fan out and fan in, with the sixth message causing the completion criteria to be met. After this, a message with six elements is propagated from the fan in primitive's out terminal to the callout.

The second case considers the processing when the fan in is configured with a completion criteria of count equal to three. In this flow, once the message arrives at the fan out, there are three iterations between the fan out and fan in. On arrival of the third message, the completion criteria is met and a message containing three processed elements is propagated from the out terminal of the fan in to the callout. The flow then returns to the fan out and three more iterations occur between the fan out and fan in. Upon arrival of the third message, the completion criteria is met and a message containing three processed elements is sent to the callout from the out terminal of the fan in.

Completion criteria and flow characteristics



- Scenario
  - Fan out configured in iterate mode
  - Incoming SMO has six elements in the array
- Criteria: XPath expression = FanOutContext/iteration = 3
  - NOTE: index in FanOutContext/iteration is 0 based
  - Four iterations between fan out and fan in
  - Message with four elements sent to the callout from the out terminal
  - Two iterations between fan out and fan in
  - Message with two elements sent to the callout from the incomplete terminal

On this slide the overall flow and the incoming message are the same as on the previous slide. The completion criteria for the fan in is set to an XPath expression which checks to see if the iteration field in the FanOutContext is equal to the value three. Before explaining the flow, note that the index value in the FanOutContext iteration field is zero based. Therefore, the first iteration has a value of zero, the second iteration has a value of one, and so on.

In this situation, after the incoming message arrives at the fan out, there are four iterations between the fan out and fan in. When the fourth message arrives, the FanOutContext iteration field contains the value three, so the completion criteria is met. The message is propagated from the fan in primitives out terminal to the callout. The control now returns to the fan out and there are two iterations between the fan out and fan in. When the second message arrives at the fan in, the completion criteria is not met, but the fan out has no more messages to send. Therefore, the incomplete terminal of the fan in is fired and a message with two elements is sent to the callout.

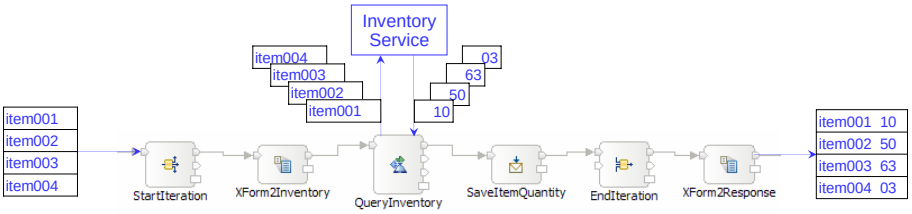
Understanding timeout processing

- A timeout is not considered an error condition
  - Does not raise an exception
  - Does not cause the flow to terminate
- Processing
  - After a timeout occurs the configured completion criteria is ignored
  - Any messages arriving at the fan in after the timeout delay is “late”
  - The message is propagated through the incomplete terminal
  - The associated fan out continues to send messages after the timeout

Now that you understand the relationship between completion criteria and the flow behavior, you have a basis for understanding the flow behavior associated with timeout processing.

The first thing to note is that a timeout for a fan in primitive does not behave in a way that you normally associate with a timeout. For a fan in, a timeout is not considered an error condition, it does not cause an exception to be raised nor does it terminate the flow. The first thing that happens once a timeout occurs is that the configured completion criteria is no longer considered. Any message that arrives at the fan in after the time out has occurred is considered “late” and the message is propagated through the incomplete terminal. The associated fan out processing is not terminated and will continue to send additional messages which will each be propagated through the fan in primitives incomplete terminal when they arrive.

Aggregation with iterate mode on



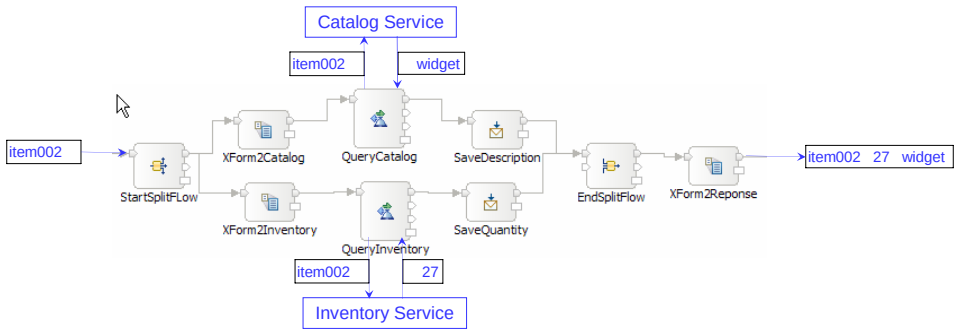
- Determine status of inventory for list of items
  - Request contains array of item IDs
  - Response contains array of item IDs and quantity in stock
  - Flow iterates for each item between fan out and fan in
  - Inventory service called for each individual item
  - Shared context used to save individual results during iteration
  - Contents of shared context use to build the final response message

The next slides look at scenarios for using a fan in. On this slide, an aggregation scenario with iterate mode on is examined.

In this scenario, a request is made to find out the inventory status of a list of items. The input contains a list of item IDs and the response is the list of item IDs along with the current quantity of each item that is in stock. There is an inventory service which can be queried to determine the in stock quantity, but this service can only be called for a single item at a time, not for a list of items.

To implement this scenario, there is a fan out and an associated fan in. Looking at the flow above, starting on the left, you can see a list of item IDs being passed into the StartIteration fan out primitive. It iterates through the array, passing the SMO with each element to the XForm2Inventory XSL transformation primitive. This primitive does two things. It sets up the message body so that the call to the inventory service can be made, and it saves the item ID in the shared context. The QueryInventory service invoke primitive calls the inventory service, obtaining the in stock quantity for that one item. The next primitive is the SaveItemQuantity message element setter which takes the item quantity returned and saves it in the shared context. The EndIteration fan in primitive is next, which causes the flow to return to the StartIteration fan out unless all items have already been processed. When that is the case, the flow continues to the XForm2Response XSL transformation which takes the values from the shared context and builds the response message with the list of item IDs and quantities.

Aggregation with iterate mode off



- Determine description and quantity in stock for an item
  - Request contains single item ID
  - Response contains single item ID with quantity and description
  - Two flow paths between fan out and fan in, each calling a service
  - Shared context used to save individual results of each flow path
  - Contents of shared context use to build the final response message

This scenario is for aggregation with iterate mode off. In this flow, a single item ID is received and the response contains the item ID, an in stock quantity and item description. The in stock quantity is obtained from the inventory service and the description is obtained from a catalog service.

Looking at the flow, you can see the item ID entering the StartSplitFlow fan out primitive, which is configured in once mode. When the out terminal is fired, the flow passes to the XForm2Catalog XSL transformation which saves the item ID in the shared context and sets up the message body for the call to the catalog service. The QueryCatalog service invoke primitive makes the call and receives the item description in response. The SaveDescription message element setter saves the description in the shared context. The flow continues to the EndSplitFlow fan in, which is configured to complete after it receives two messages. Since this is the first message, the flow returns to the StartSplitFlow fan out primitive, and continues to the XForm2Inventory XSL transformation. This primitive sets up the body to call the inventory service. The QueryInventory service invoke primitive calls the inventory service and receives the in stock quantity in response. The SaveQuantity message element setter saves the quantity in the shared context and the flow proceeds again to the EndSplitFlow fan in, which is now complete because this is the second message received. The flow proceeds to the XForm2Response XSL transformation, which builds the response message body from the item ID, quantity and description that is saved in the shared context.

Summary

- Examined the fan in mediation primitive

Fan in

  - Overview of function
  - Use of terminals
  - Definition of properties
  - Error handling
  - Completion criteria and flow characteristics
  - Behavior of timeout
  - Details of usage scenarios

In this presentation the fan in mediation primitive was examined. It started with an overview of the fan in primitive, along with information about the primitive's use of terminals, its properties and error handling considerations. There was a discussion of the fan in completion criteria and the effects that they have on a mediation flow. The behavior of a fan in timeout was explained and a couple of usage scenarios were provided.

Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send e-mail feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_WBPMv62_FanInPrimitive.ppt

This module is also available in PDF format at: ../WBPMv62_FanInPrimitive.pdf

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

Trademarks, copyrights, and disclaimers

IBM, the IBM logo, ibm.com, and the following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

WebSphere

If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of other IBM trademarks is available on the Web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml

Other company, product, or service names may be trademarks or service marks of others.

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.

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.