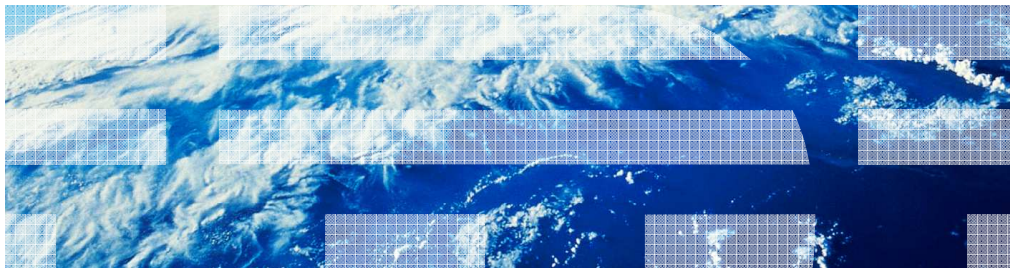


WebSphere Business Process Management

WebSphere Integration Developer
WebSphere Enterprise Service Bus
WebSphere Process Server

Routing primitives utilizing message content information



This presentation introduces the routing primitives that influence the flow of control through a mediation based on message content and configuration information.

Goals and agenda

- Goal
 - Provide an introduction to mediation primitives with these characteristics:
 - Classified in WebSphere Integration developer as routing primitives
 - Influence flow of control in a mediation based on message content and configuration information
- Agenda
 - Introduce the basic functionality of:
 - Message filter
 - Type filter
 - Fan out and fan in
 - Flow order

The goal of this presentation is to introduce a set of primitives that share some common characteristics. These primitives are classified within WebSphere Integration Developer as routing primitives. Within that classification, these are the primitives that influence the flow path taken in a mediation flow. The message content, configuration information and state of the flow can all contribute to how these primitives control the flow.

The presentation provides a description of the basic functionality of each. The primitives presented are the message filter primitive, the type filter primitive, the fan out and fan in primitives, and the flow order primitive.

Message filter primitive



This section provides an overview of the message filter primitive.



Message filter primitive – Overview of function

- Enables control of the paths taken through the flow
- Contains one or more filters – where each filter contains:
 - A simple XPath expression to be evaluated to true or false
 - Name of an output terminal through which to propagate the message
- A filter with a match (evaluates true) fires the output terminal
- Filters are evaluated in the order in which they are defined
- Configurable to allow propagation of the message using distribution mode property
 - By firing the output terminal of only the first matching filter
 - By firing the output terminal of all matching filters
- A default terminal is fired when there are no matching filters
- The service message object (SMO) is not updated

The purpose of the message filter is to enable flow of control logic within a mediation, so that different paths can be taken based on the evaluation of values within the SMO.

The primitive contains one or more filters. Each filter contains a conditional XPath expression that evaluates to true or false, and an output terminal through which to propagate the message. When the expression evaluates to true, it is called a match, and the message is propagated through the terminal of the matching filter.

Filters are defined in a table and are evaluated in the order in which they appear in the table. A configuration option, called distribution mode, is used to specify if the message is propagated through only the first matching filter or through all matching filters. In the case where none of the filters results in a match, there is a default terminal through which the message is propagated.

The SMO is not updated by the message filter primitive.



Resources

- Information center
 - [Message filter mediation primitive](#)
 - [Dynamic routing using message filtering](#)
- IBM Education Assistant
 - [Message filter primitive presentation for V6.2](#)

To further understand the message filter primitive, follow the links on this slide. The information center contains a description of the message filter primitive that includes usage information and defines all of its properties. The next link is to an example that illustrates the usage of a message filter primitive to implement a dynamic routing scenario in a mediation flow. Finally, the IBM Education Assistant for V6.2 contains a complete presentation on the message filter primitive that is still applicable to V7. It provides documentation of the properties, considerations for understanding the distribution mode property, and an example usage of the primitive.

Type filter primitive



The type filter primitive is addressed in this section.



Type filter primitive – Overview of function

- Enables control of the paths taken through the flow based the type of message elements
- Contains one or more filters – where each filter contains:
 - A simple XPath expression identifying an element in the SMO
 - A type to compare with the type of the element
 - Name of an output terminal through which to propagate the message
- A filter with matching types fires the output terminal
 - Filters are evaluated in the order in which they are defined in the table
 - Only the first filter with a matching type is fired
 - An element that is a derived type of the filter type is a match
- A default terminal is fired when there are no matching filters
- The service message object (SMO) is not updated

The purpose of the type filter is to provide logic to control the flow within a mediation, so that different paths can be taken based on the evaluation of element types within the SMO.

The primitive contains one or more filters contained in a table. Each filter contains a simple XPath expression that identifies an element within the SMO, a specification of type used to compare with the element, and an output terminal through which to propagate the message. When the types match, the message is propagated through the terminal of the matching filter. It is considered a match if the element is of the same type. It is also a match if the element is a derived type of the type specified in the filter.

Filters are defined in a table and are evaluated in the same order that they appear in the table. The message is propagated through only the terminal of the first matching filter. In the case where none of the filters results in a match, there is a default terminal through which the message is propagated.

The SMO is not updated by the type filter primitive.



Resources

- developerWorks articles
 - [What's new in WebSphere Enterprise Service Bus V6.2, Part 2: Service gateway patterns](#)
- Information center
 - [Type filter mediation primitive](#)
- IBM Education Assistant
 - [Type filter primitive presentation for V6.2](#)

The links on this slide provide you with additional resources to help you fully understand the type filter primitive. There is a developerWorks article based on V6.2 describing the service gateway patterns. The type filter primitive was added to WebSphere Enterprise Service Bus in V6.2, specifically to support the static service gateway pattern. The article goes into detail explaining the static service gateway pattern and the part that the type filter plays in the scenario.

The information center contains a description of the type filter primitive that includes usage information and defines all of its properties. Finally, the IBM Education Assistant for V6.2 contains a complete presentation on the message filter primitive that is still applicable to V7 usage of the primitive. It provides documentation of the properties, considerations for some property settings, and an example usage.

Fan out and fan in primitives



Both the fan out and fan in primitives are addressed in this section. They are combined into the same section because the fan in primitive is always paired with a fan out primitive, and they are best understood in combination.

Fan out primitive - Overview of function



- The fan out primitive provides either:
 - The front of an aggregation scenario
 - Message broadcast
- For an aggregation scenario
 - There is a fan in primitive which acts as the point of aggregation
 - A fan in must be associated with a specific fan out instance
- Fan out has two modes of operation
 - Iterate mode
 - Iterates though a repeating element contained within the input message
 - Output terminal fired once for each element instance
 - Output message contains input message plus copy of element instance
 - Once mode
 - Output terminal is fired once
 - Causes the message to be propagated on each of multiple flow paths wired to the terminal
 - Output message is identical to the input message

When considering a fan out primitive, there are two basic ways in which it can be used, either participating in an aggregation scenario or used for enabling message broadcast. When used as part of an aggregation scenario, there is a specific fan in primitive instance in the flow that is associated with the fan out. The fan out is the beginning and the fan in is the end of the flow segment that performs the aggregation. You can think of it as the start and end of a processing loop.

The fan out primitive has two modes of operation, the first being the iterate mode. In this 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. When the output terminal is fired, the SMO contains the original message, plus a copy of the element instance to be processed during this iteration. The copy of the element instance is contained in a designated location in the SMO context. This allows code within the aggregation flow to access the current element without having to index into the repeating element.

When in once mode, the output terminal is fired once. For this mode to be used in an aggregation, the flow must be constructed with multiple flow paths following the fan out. So in actuality, each flow path wired to the output terminal is driven. In this case, the SMO propagated on each path is unchanged from the inbound SMO received by the fan out.



- There are four basic scenarios possible
 - Aggregation with iterate mode
 - Aggregation with once mode
 - Broadcast with iterate mode
 - Broadcast with once mode
- Aggregation scenarios and fan in completion criteria
 - A fan in is configured with completion criteria
 - Completion criteria affects overall flow path
 - Between the fan out and fan in
 - Flow following the fan in
 - Configuration of fan out and fan in completion criteria must be complementary

Consider the fact that a fan out can be used in an aggregation or broadcast scenario, and that it also has two modes of operation, iterate mode or once mode. The result is that there are four overall basic usage scenarios in which a fan out can participate. The first is an aggregation using iterate mode to loop through an array of elements, performing the same processing for each element. When all the elements have been processed, the associated fan in completes and the results of the aggregation are constructed in the SMO by the flow following the fan in. The next is also an aggregation, with the fan out configured in once mode. In this case, there are multiple flow paths between the fan out and fan in, with each flow path running once. When all the flow paths have completed, the fan in completes and the results of the aggregation are constructed in the SMO by the flow following the fan in. The third is a broadcast with iterate mode. This allows each element of an array in the incoming message to have the same processing performed. However, there is no fan in and the results of processing each element are not aggregated together. Finally, there is broadcast with once mode. In this case, the fan out serves as the head of multiple flow paths, each of which is passed the same message, and the results of processing are not aggregated together.

In aggregation scenarios, the fan in associated with the fan out is configured with completion criteria. The completion criteria will affect the overall flow, controlling the flow between the fan out and fan in, and determining when the flow following the fan in should be driven. Because of this, it is 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. This is addressed further in a subsequent slide.

Fan in primitive - Overview of function



- The fan in primitive is used in aggregation scenarios
 - Acts as the point of aggregation in the flow
 - Is associated with a specific fan out primitive instance
 - Completion criteria determines when the aggregation is complete
- A fan in and mode of operation of its associated fan out
 - Fan out operating with iterate mode on
 - There is one flow path between the fan out and fan in
 - The fan out is iterating over a repeating element in the message
 - The fan in receives a message for each instance of a repeating element
 - Fan out operating with iterate mode off
 - There are multiple flow paths between the fan out and fan in
 - The fan in receives one message from each of the flow paths

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 from the fan out primitive's out terminal is driven sequentially, and the fan in receives the result of each flow path as input.




- 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

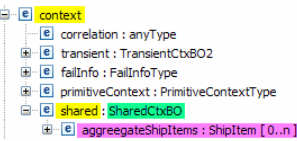
A fan in primitive is configured with completion criteria, which is very important for defining the overall flow path that will occur when the flow is run. There are two major divisions in the flow, the part that occurs between the fan out and fan in primitives and the part that proceeds after the fan 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.



The shared context





- Problem
 - Each time the fan out output terminal is fired a new SMO instance is created
 - Each new SMO instance is a deep copy
 - How are results from each iteration/flow between the fan out and fan in aggregated?
- Solution
 - Shared context in the SMO
 - Single memory area that is not deep copied with each SMO instance
- Shared context usage
 - It is defined by a business object (similar to transient and correlation contexts)
 - In iterative aggregations, the business object typically contains an array
 - Flows between 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

14
Routing primitives utilizing message content information
© 2010 IBM Corporation

This slide examines the shared context used during aggregation scenarios, examining why it is needed, what it provides and how it is used. The first thing to look at is how the fan out handles the SMO when firing its output terminal. The original message arriving at the fan out is saved by the primitive, and a new deep copy is created and passed through the output terminal to the flow. Whatever changes are made to the SMO during the flow are not seen by the other iterations or flow paths. Each receives a new copy of the message as it arrived at the fan out. This poses a problem in an aggregation scenario where the results of processing each iteration or flow are to be aggregated together.

The solution to this is the shared context, which is kept in a shared memory area. Each time the SMO is deep copied, rather than copying the shared context, the SMO contains a reference to the shared memory area.

When building your aggregation flow, you define what the shared context will contain using a business object, similar to how you define the transient or correlation contexts. For an iterative aggregation, the business object typically contains an array. Each iteration or flow between the fan out and fan in needs to update the shared context with the data it is contributing to the aggregated result. Once the fan in completion criteria is met, the flow following the fan in can take the contents of the shared context and use it to build the aggregated message in the SMO body.

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. This slide discusses 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 primitive's 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.

Resources

- developerWorks articles
 - Aggregation functionality in IBM WebSphere Enterprise Service Bus V6.1
 - [Part 1: Introduction to aggregation](#)
 - [Part 2: Service invocation](#)
 - [Part 3: Best practices and patterns for aggregation](#)
- Information center
 - [Fan out mediation primitive](#)
 - [Fan in mediation primitive](#)
 - [Aggregating and broadcasting messages](#)
- IBM Education Assistant
 - [Fan out primitive presentation for V6.2](#)
 - [Fan in primitive presentation for V6.2](#)
 - [Service invoke primitive presentation for V6.2](#)

16 Routing primitives utilizing message content information © 2010 IBM Corporation

The links on this slide provide you with additional resources to help you understand aggregation scenarios and how fan out and fan in primitives play a role in these scenarios.

There is a series of developerWorks articles on aggregation based on V6.1, the release in which aggregation was introduced. These articles are still applicable to V7 as the basic functionality of aggregation has not changed since that release. One possible exception to this might be service invocation which was enhanced in V6.2 to enable asynchronous parallel aggregation scenarios using service invoke primitives.

The information center contains a descriptions of both the fan out and fan in primitives that includes usage information and defines all of their properties. The next link, aggregating and broadcasting messages, is to a page that itself is a set of links to related topics. These include usage of the shared context, combining results from multiple services, broadcasting messages, performing chained aggregation and an iterative aggregation example.

Finally, the IBM Education Assistant for V6.2 contains a complete presentation on the primitives, all of which address usage and define the properties of the primitive. In addition, the fan out presentation addresses parallel processing of service calls, and has four examples that address the four basic scenarios in which a fan out is used. The fan in presentation provides more detail on fan in completion criteria and how it affects flow behavior, understanding timeout processing, and examples of scenarios utilizing a fan in. A link to the service invoke primitive is provided because it goes into detail on invocation style and how it relates to an aggregation with asynchronous parallel processing.

Flow order primitive



This section looks at the characteristics of the flow order primitive.

Flow order primitive – Overview of function



- Controls the order in which branches of the flow are fired
- It has one input terminal and two or more output terminals
- A copy of the unmodified input message is fired on each of the output terminals
- The terminals are fired in the order defined on the primitive
- There are no properties
- Other behavior of the flow is identical to wiring multiple branches from an output terminal of a primitive
 - Waits for a branch to complete before firing the next terminal
 - If branch contains asynchronous service invoke the next terminal is fired after service is invoked
 - An unhandled exception on any branch terminates further processing

The flow order primitive is used to control the order in which multiple branches on a flow are fired. It has one input terminal and two or more output terminals, one for each branch on which the message is to be propagated. The message is not modified by the flow order primitive, with a new copy of the SMO propagated on each branch. The order in which the terminals are fired is the order in which they appear on the primitive, from top to bottom. This primitive is unique in that it has no properties.

To put this primitive into perspective, the behavior of the flow with a flow order primitive is identical to the behavior when there are multiple wires coming off of an out terminal. The only difference is that the order in which the message is propagated onto the branches is prescriptive rather than being undefined. Therefore, these behaviors still exist. The mediation flow waits for one branch to complete before the message is propagated onto the next branch. The exception to this is when a branch contains an asynchronous service invoke primitive, in which case the next branch receives control after the service is invoked. Finally, if there is an unhandled exception on any branch, the processing terminates.

Resources



- Information center
 - [Flow Order mediation primitive](#)

The information center contains a description of the flow order primitive.

Summary

- Introduced the basic functionality of these primitives:
 - Message filter
 - Type filter
 - Fan out and fan in
 - Flow order

This presentation provided a description of the basic functionality of the routing primitives that influence flow of control in a mediation based on message content and configuration information . The primitives presented were the message filter primitive, the type filter primitive, the fan out and fan in primitives, and the flow order primitive.



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 WBPMv7 RoutingContentPrimitives.ppt](mailto:iea@us.ibm.com?subject=Feedback%20about%20WBPMv7%20RoutingContentPrimitives.ppt)

This module is also available in PDF format at: [../WBPMv7_RoutingContentPrimitives.pdf](..\\WBPMv7_RoutingContentPrimitives.pdf)

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



Trademarks, disclaimer, and copyright information

IBM, the IBM logo, ibm.com, developerWorks, and WebSphere are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of other IBM trademarks is available on the Web at "[Copyright and trademark information](http://www.ibm.com/legal/copytrade.shtml)" at <http://www.ibm.com/legal/copytrade.shtml>

THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY. WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. IN ADDITION, THIS INFORMATION IS BASED ON IBM'S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM WITHOUT NOTICE. IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION. NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, NOR SHALL HAVE THE EFFECT OF, CREATING ANY WARRANTIES OR REPRESENTATIONS FROM IBM (OR ITS SUPPLIERS OR LICENSORS), OR ALTERING THE TERMS AND CONDITIONS OF ANY AGREEMENT OR LICENSE GOVERNING THE USE OF IBM PRODUCTS OR SOFTWARE.

© Copyright International Business Machines Corporation 2010. All rights reserved.