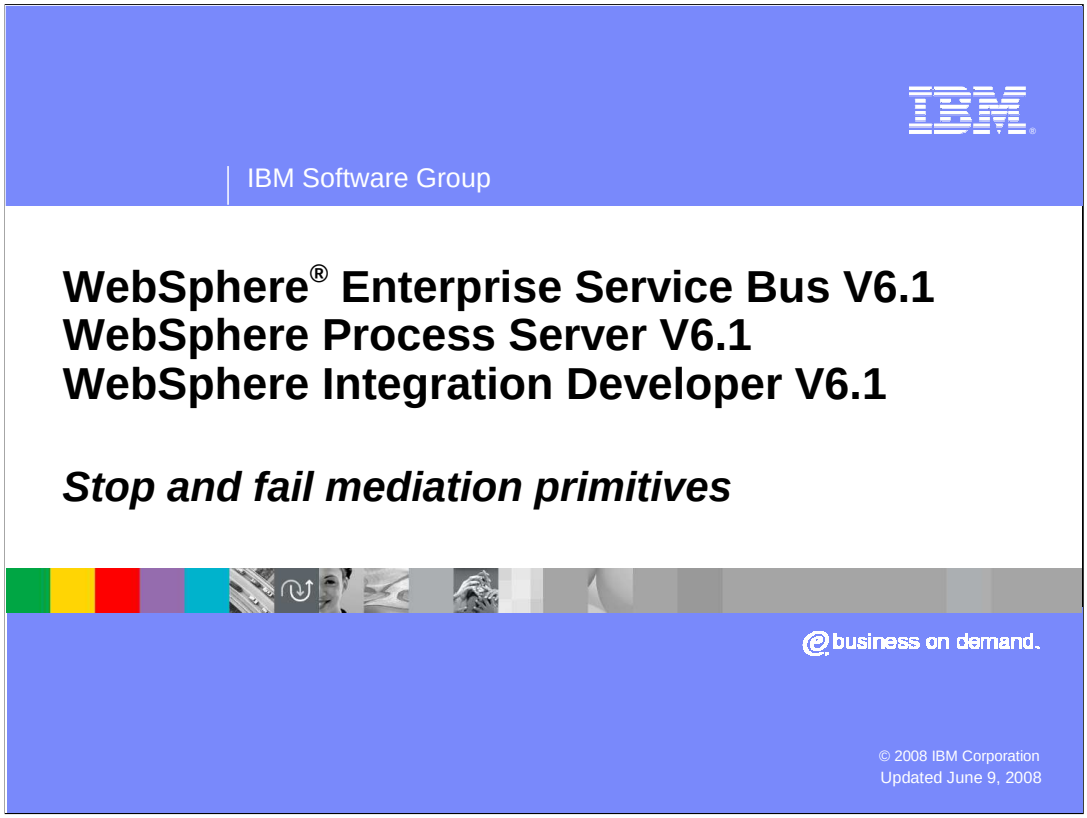


This presentation provides a detailed look at the stop and fail primitives.

Goals

- Understand the stop mediation primitive
  - Overview of function
  - Use of terminals
  - Definition of properties
  - Use of stop versus unwired terminals
  - Example usage

- Understand the fail mediation primitive
  - Overview of function
  - Use of terminals
  - Definition of properties
  - Example usage

The goal of this presentation is to provide you with a full understanding of the stop and fail primitives.

The presentation assumes that you are already familiar with the material presented in the **Mediation primitive common details** presentation and the **Common details – Promoted properties** presentation. These two presentations serve as a base for understanding mediation primitives in general.

The stop primitive is examined first. An overview of its function is provided along with a description of the primitive's use of terminals and properties. The behavior of wiring an output or fail terminal to a stop primitive versus leaving the terminals unwired is compared and contrasted. A usage example of a stop primitive is then provided.

Similarly, an overview of the fail primitive's functionality is provided along with a description of the primitive's use of terminals and properties. Finally, a usage example for a fail primitive is given.

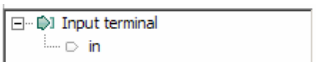
Section

Stop

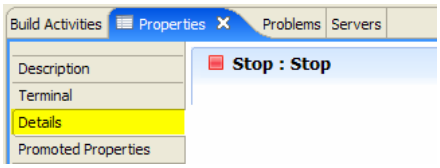
This section examines the stop primitive.

Stop – Function, terminals and properties

- Terminates the current path of the mediation flow
  - No exception or message of any kind is generated
  - There is no processing of the service message object (SMO)
  - If there are other active paths, they continue
- Terminals:
  - Input terminal only
  - No output or fail terminals



- This primitive has no properties



The stop primitive enables you to terminate the current path of a mediation flow without raising an exception or generating any kind of a message.

The SMO is not processed or updated in any way. If there are multiple active paths in the mediation flow, the stop primitive only stops the path on which it is wired and the other paths continue.

The stop primitive has an input terminal but has no output terminals or fail terminal.

The stop primitive has no properties, and therefore has no promoted properties either.

Stop – Use of stop versus unwired terminals

- Output terminal – unwired versus wired to a stop
  - Behavior is the same
  - Terminates this active path with no exception or message
    - Flow continues if there are other active paths
  - Explicit use of stop is preferable because it shows intention to stop
- Fail terminal – unwired versus wired to a stop
  - Behavior is different
  - Unwired - terminates the flow by throwing an exception
    - Flow terminates even if there are multiple active paths
  - Wired to a stop
    - Suppresses the exception
    - If there are multiple active paths, only this path terminates
    - In most cases it is better not to suppress the exception

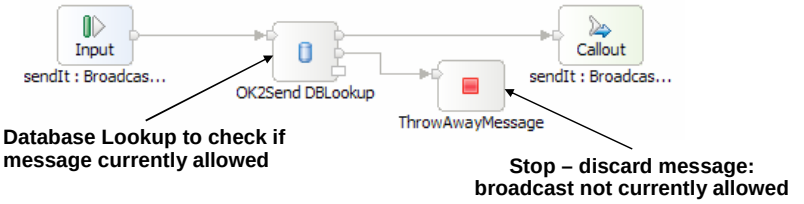
In this slide the behavior of a terminal wired to a stop primitive is compared to the behavior seen when the terminal is left unwired.

The first case to consider is for an output terminal. In this case the behavior is the same if the terminal is left unwired as opposed to being wired to a stop primitive. The active path is terminated with no exception or message. If this was the only active path, the mediation flow ends, but if there are other active paths in the mediation, the flow continues. The explicit use of the stop terminal is preferable because it shows the intention to stop the flow at this point.

The other case to consider is for a fail terminal, in which case the behavior is different. When a fail terminal is left unwired, all active paths of the flow are terminated, an exception is thrown and a log message is written. If the fail terminal is wired to a stop, the exception is suppressed, no log messages are written and other active paths continue. In general, wiring a stop primitive to a fail terminal is not the appropriate logic for the flow.

Stop – Usage example

- Example – Filter broadcast messages
  - Broadcast service dynamically filters messages that can be broadcast
  - Content type identifier contained in the message
  - Database used to maintain current set of legal broadcasts
  - Database lookup used to validate if message can be broadcast
  - Messages currently being filtered out are thrown away
- Mediation logic:
  - Wire a stop to the keyNotFound terminal of the database lookup



This slide provides a usage example for the stop primitive.

The requirement for the mediation is to selectively forward broadcast messages. The messages contain a content identifier that is used to determine if the message should be forwarded or ignored. A database is used to contain the content identifiers for messages that currently should be broadcast. The database is updated to add or delete message identifiers as the broadcast needs vary. The mediation flow is composed of a database lookup mediation primitive, which takes the content identifier from the message and uses it as a key for the lookup. If the lookup is successful, the flow continues through the out terminal that is wired to the callout for the broadcast service. If the lookup is not successful, the keyNotFound output terminal is wired to a stop mediation primitive, which essentially causes the message to be discarded.

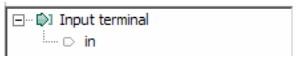
Section

Fail

This section examines the fail primitive.

Fail – Function and terminals

- Terminates mediation flow by raising an exception
  - Allows you to raise an exception at any point in the flow
  - FailFlowException raised containing user defined text
  - There is no processing of the SMO
  - Enables creation of exceptions based on business logic
  - All active paths through the flow are terminated

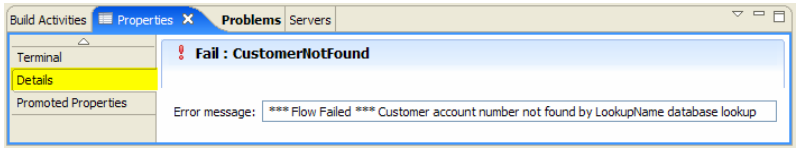


- Terminals:
  - Input terminal only
  - No output or fail terminals

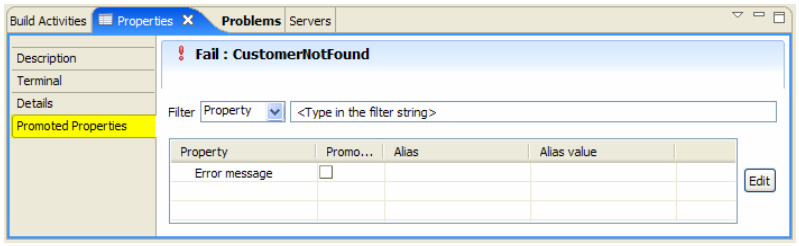
The fail primitive enables you to terminate a mediation flow by raising an exception. This allows you to create exceptions based on business logic errors within the flow. The exception thrown is a FailFlowException, which contains a text string that you provide describing the error condition. The SMO is not processed or updated in any way by the fail primitive. If there are multiple active paths in the mediation flow, the fail primitive causes all of them to be stopped.

The fail primitive has an input terminal, but has no output terminals or fail terminal.

Fail – Properties and promoted properties



- Properties:
  - Error message
    - Text string that is placed into the exception
    - String is fixed – variable data cannot be inserted



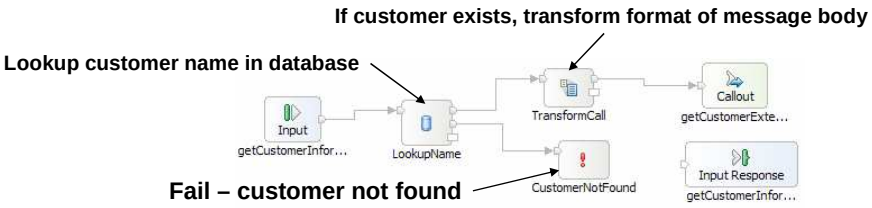
- Promotable
  - Error message

The fail primitive has only one property, called **Error message.** It contains a text string that describes the error condition encountered by the flow. The string is static, so no data from the SMO can be inserted into the message. A FailFlowException is raised containing the text string.

The error message property is promotable as shown in the lower screen capture.

Fail – Usage example

- Example – Account number based message augmentation
  - Mediation flow input operation contains only customer number
  - Operation on target service requires customer name and number
  - Database lookup used to obtain customer name
  - It is an error when no customer record is found
- Mediation logic:
  - Database lookup uses account number to access customer record
  - When customer record found, use XSLT to transform message
  - When customer record not found, use fail primitive to raise exception



This slide shows a usage example of the fail primitive.

In this scenario the mediation needs to add additional information to the message. When the additional information is unavailable it is an error condition. The request message contains only a customer number, but the target service provider's interface requires both the customer number and customer name. A database lookup primitive is used to obtain the customer's name based on the customer number. In the case where the customer number is not found in the database, it is considered an error.

The mediation flow is shown at the bottom of the slide. The input node is wired to a database lookup, which obtains the customer name from the database. If the lookup succeeds, the out terminal is wired to an XSL transformation primitive used to modify the message body. The modified message is then passed on to the callout node for the service provider. In the case where the lookup fails, the flow goes through the keyNotFound output terminal, which is wired to a fail primitive. The fail primitive raises an exception indicating that the customer record was not found.

Section

Summary

The following slide presents a summary of this presentation.

Summary

- Examined the stop mediation primitive
  - Overview of function
  - Use of terminals
  - Definition of properties
  - Use of stop versus unwired terminals
  - Example usage

- Examined the fail mediation primitive
  - Overview of function
  - Use of terminals
  - Definition of properties
  - Example usage

In this presentation, details were provided regarding both the stop and fail primitives. First, the stop primitive was examined, providing an overview of its function, a description of the primitive's use of terminals and its properties. The behavior of the stop primitive as opposed to leaving terminals unwired was contrasted. A usage example of a stop mediation primitive was given. Similarly, an overview of the fail mediation primitive was provided, along with its use of terminals, its properties and a usage example.

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_WBIV61_StopFailPrimitives.ppt

This module is also available in PDF format at: ../WBIV61_StopFailPrimitives.pdf

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

Trademarks, copyrights, and disclaimers

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

IBM WebSphere

A current list of other IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

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

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

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

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

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

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

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