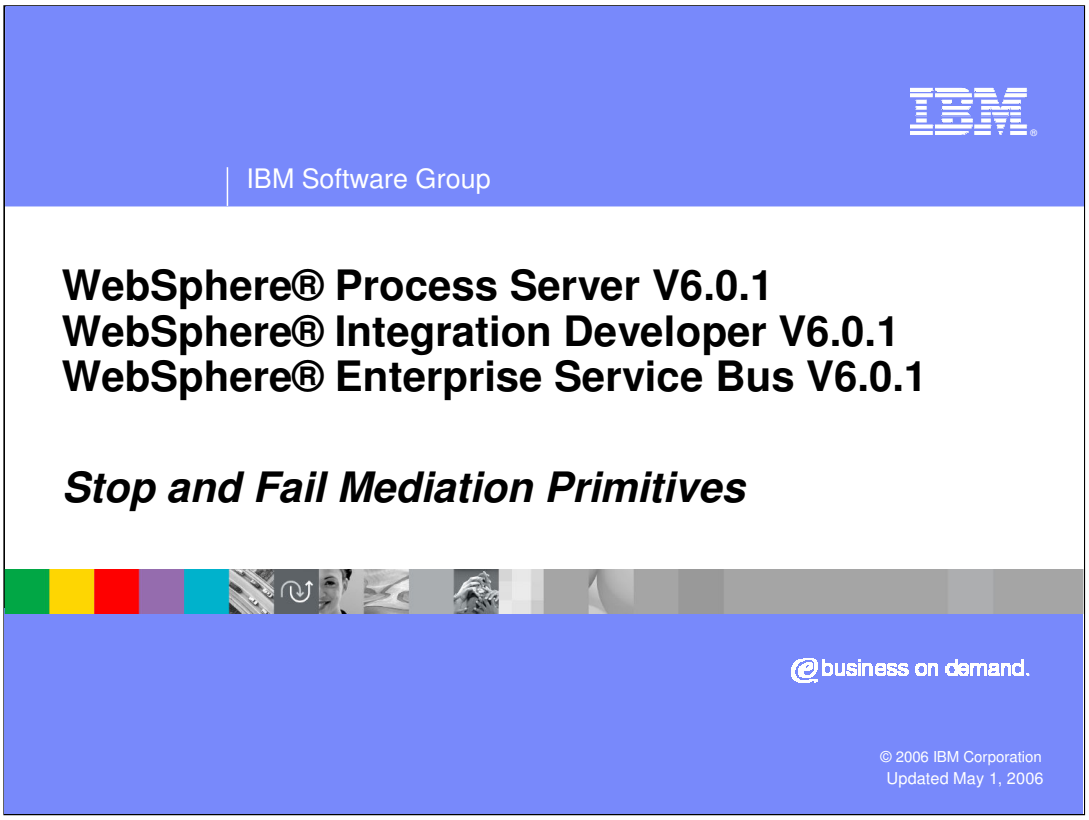


This presentation will provide a detailed look at the Stop mediation primitive and the Fail mediation primitive.

Goals

- Understand the Stop mediation primitive
  - Overview of function
  - Use of terminals
  - Definition of properties
  - Use of Stop vs. 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 mediation primitives. The presentation assumes that you are already familiar with the material presented in the **Mediation Primitive Common Details** presentation, which serves as a base for understanding mediation primitives in general. The Stop mediation primitive is examined first and an overview of its function provided, describing the primitive's use of terminals and its properties. The behavior of the Stop primitive as opposed to leaving terminals unwired is contrasted and an example usage of a Stop mediation primitive is provided. Similarly, an overview of the Fail mediation primitive is provided along with its use of terminals and properties and a usage example.

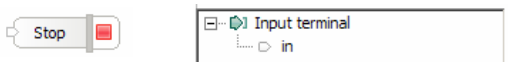
Section

Stop

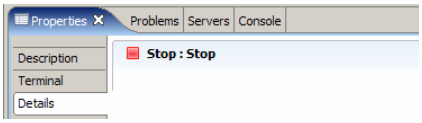
This section examines the Stop mediation primitive.

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, those will continue
- Terminals:
  - Input terminal only
  - No Output or Fail terminals



- This primitive has no properties



The Stop mediation 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 mediation primitive has no properties, with only an input terminal and no output terminals or fail terminal.

Stop – Use of Stop vs. Unwired Terminals

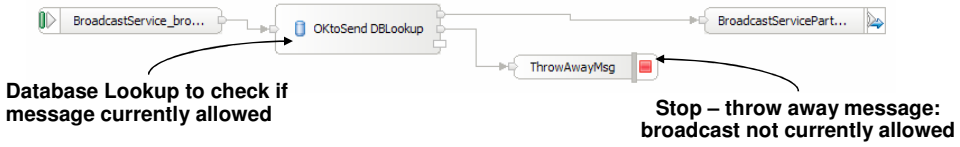
- Output terminal – unwired vs. wired to a Stop
  - Behavior will be the same
  - Terminates this active path with no exception or message
    - Flow will continue if there are other active paths
  - Explicit use of Stop is preferable
    - Explicitly shows the intention to stop
    - Suppresses warning messages in WebSphere Integration Developer
- Fail terminal – unwired vs. wired to a Stop
  - Behavior will be different
  - Unwired - terminates the flow by throwing an exception
    - Flow will terminate even if multiple active paths
  - Wired to a Stop
    - Suppresses the exception
    - If there are multiple active paths, only this path will terminate
    - In most cases it is better not to suppress the exception

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

The first case to consider is for an output terminal, in which case the behavior is the same if the terminal is left unwired as opposed to being wired to a Stop mediation primitive. The active path is terminated with no exception or message and if this is the only active path, the mediation flow ends. If there are other active paths in the mediation, flow then will continue. The explicit use of the Stop terminal is preferable because it shows the intention to stop the flow at this point. It also suppresses warning messages about unwired terminals from WebSphere Integration Developer. 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 message is written and other active paths will continue. In general, wiring a Stop mediation primitive to a fail terminal is not the appropriate logic for the flow.

Stop – Example Usage

- 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 an example usage for the Stop mediation primitive. The requirement is to have a mediation that selectively forwards broadcast messages, which 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 and 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 output terminal which 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.

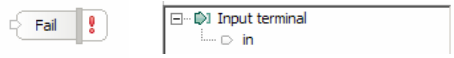
Section

Fail

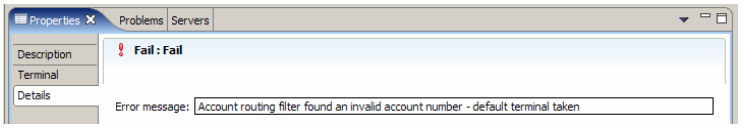
This section examines the Fail mediation primitive.

Fail – Function, Terminals and Properties

- 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



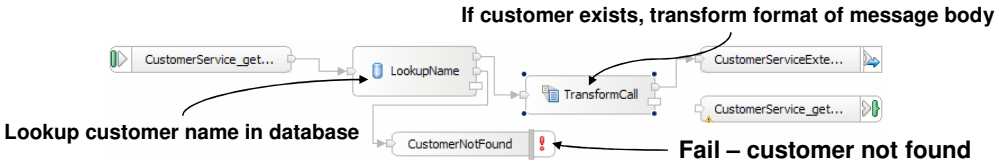
- Properties:
  - Error message
    - Text string that will be placed into the exception
    - String is fixed – variable data cannot be inserted



The Fail mediation primitive enables you to terminate a mediation flow by throwing an exception, allowing you to create exceptions based on business logic errors within the flow. The exception thrown is a FailFlowException, which contains a user defined text string describing the error condition and 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 them all to be stopped. The Fail mediation primitive has an input terminal, but no output terminals or fail terminal. The only property, called **Error message,** contains a static text string which can be used to describe the error condition that has been encountered.

Fail – Example Usage

- Example – Account number based message augmentation
  - Mediation flow input operation contains only customer number
  - Operation on target service callout requires customer name and number
  - Database lookup used to obtain the 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 an example usage of the Fail mediation primitive in a scenario that needs to add additional information to the message. The request contains only a customer number, but the target service provider's interface requires both the customer number and the 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 with the input node wired to the Database Lookup, which obtains the customer name from the database. If the lookup succeeds, the out output terminal is wired to an XSLT primitive used to modify the message body, which is then passed on to the callout to 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 and the Fail primitive throws an exception indicating that the customer record was not found.

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 vs. 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 mediation primitives. First, the Stop mediation 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 and an example usage of a Stop mediation primitive was provided. Similarly, an overview of the Fail mediation primitive was provided, along with its use of terminals, its properties and a usage example.

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	CICS	IMS	MQSeries	Tivoli
IBM(logo)	Cloudscape	Informix	OS/390	WebSphere
e(logo)business	DB2	iSeries	OS/400	xSeries
AIX	DB2 Universal Database	Lotus	pSeries	zSeries

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds.

Other company, product and 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 and/or changes in the product(s) and/or program(s) 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 (e.g., 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.