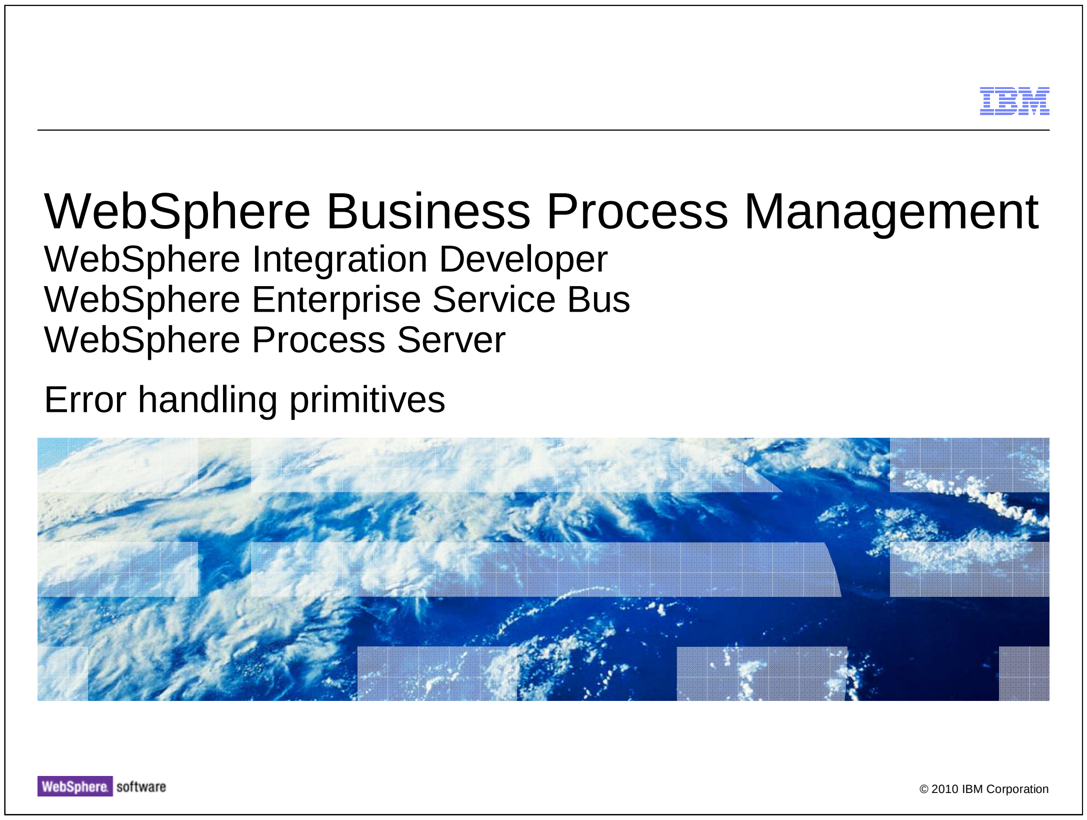
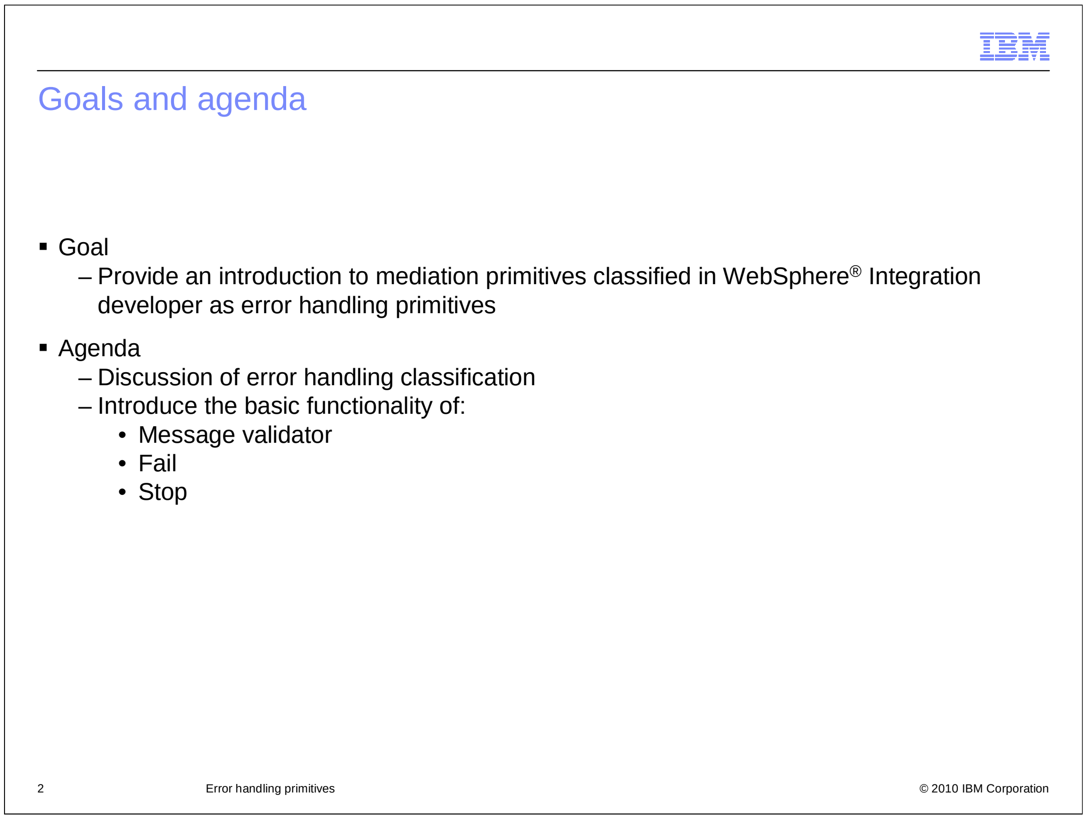


WebSphere Business Process Management

WebSphere Integration Developer
WebSphere Enterprise Service Bus
WebSphere Process Server

Error handling primitives

This presentation introduces the error handling primitives.



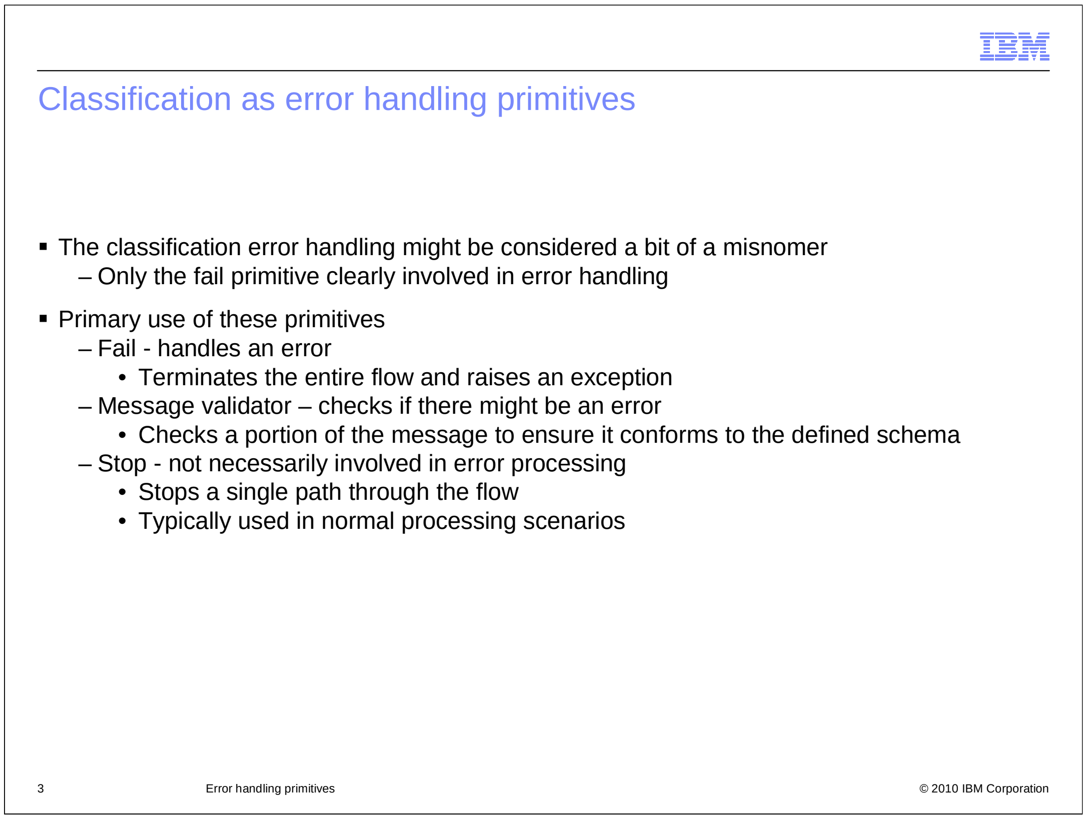
Goals and agenda

- Goal
  - Provide an introduction to mediation primitives classified in WebSphere® Integration developer as error handling primitives
- Agenda
  - Discussion of error handling classification
  - Introduce the basic functionality of:
    - Message validator
    - Fail
    - Stop

2 Error handling primitives © 2010 IBM Corporation

The goal of this presentation is to introduce the primitives that are classified within WebSphere Integration Developer as error handling primitives.

The presentation starts out by discussing the error handling classification of these primitives. It then provides a description of the basic functionality of each. The primitives presented are the message validator primitive, the fail primitive and the stop primitive.


Classification as error handling primitives

- The classification error handling might be considered a bit of a misnomer
  - Only the fail primitive clearly involved in error handling
- Primary use of these primitives
  - Fail - handles an error
    - Terminates the entire flow and raises an exception
  - Message validator – checks if there might be an error
    - Checks a portion of the message to ensure it conforms to the defined schema
  - Stop - not necessarily involved in error processing
    - Stops a single path through the flow
    - Typically used in normal processing scenarios

3 Error handling primitives © 2010 IBM Corporation

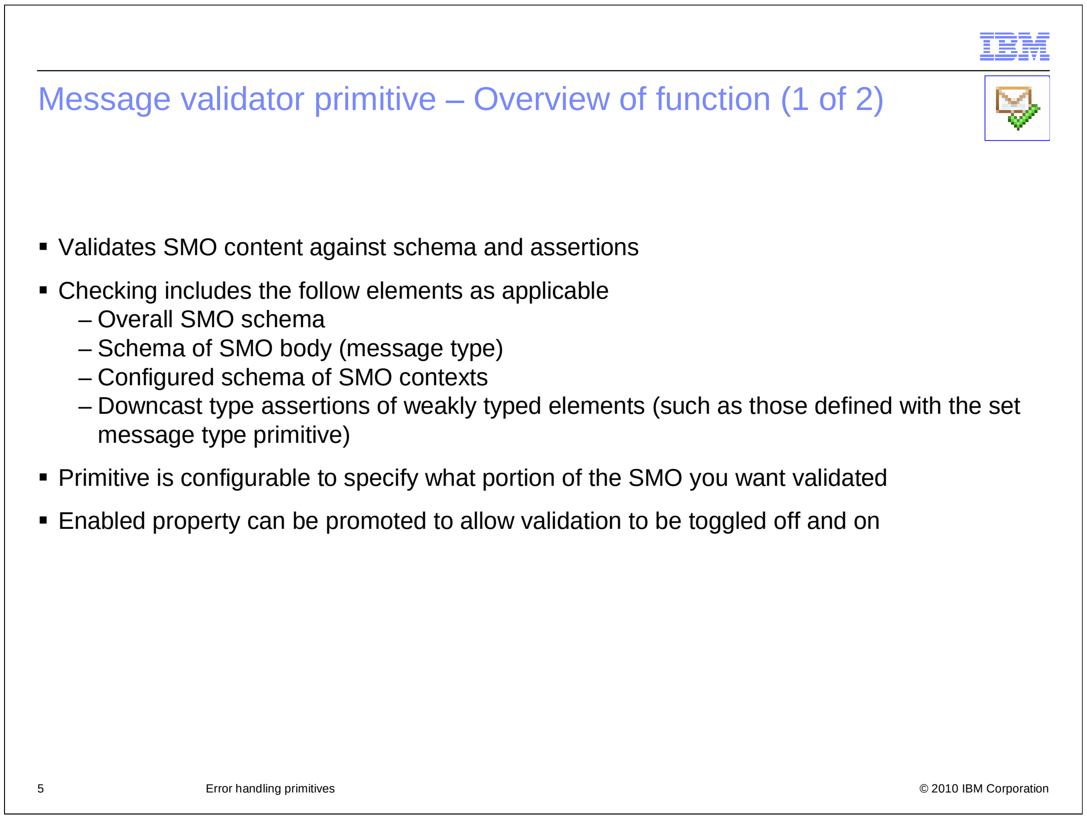
As with all the primitive classification categories in WebSphere Integration Developer, not all primitives fit nicely into their assigned classification. For these three error handling primitives, only the fail primitive is clearly involved in error handling. It is used to handle an error discovered by the flow logic by raising an exception, causing the flow to terminate. The message validator primitive checks to see if there is an error in the SMO content versus the defined schema. It then controls the flow path taken based on the result of the validation. It does not itself raise an error when the message is not valid. The stop primitive is more likely to be used in a normal processing flow than it is in an error situation. It stops a single path of the flow. This is typically done in the case where there is multiple flow path logic in the mediation, or when a mediation determines it should ignore a message.

Section

Message validator primitive

4 Error handling primitives © 2010 IBM Corporation

The first primitive to be discussed is the message validator primitive.



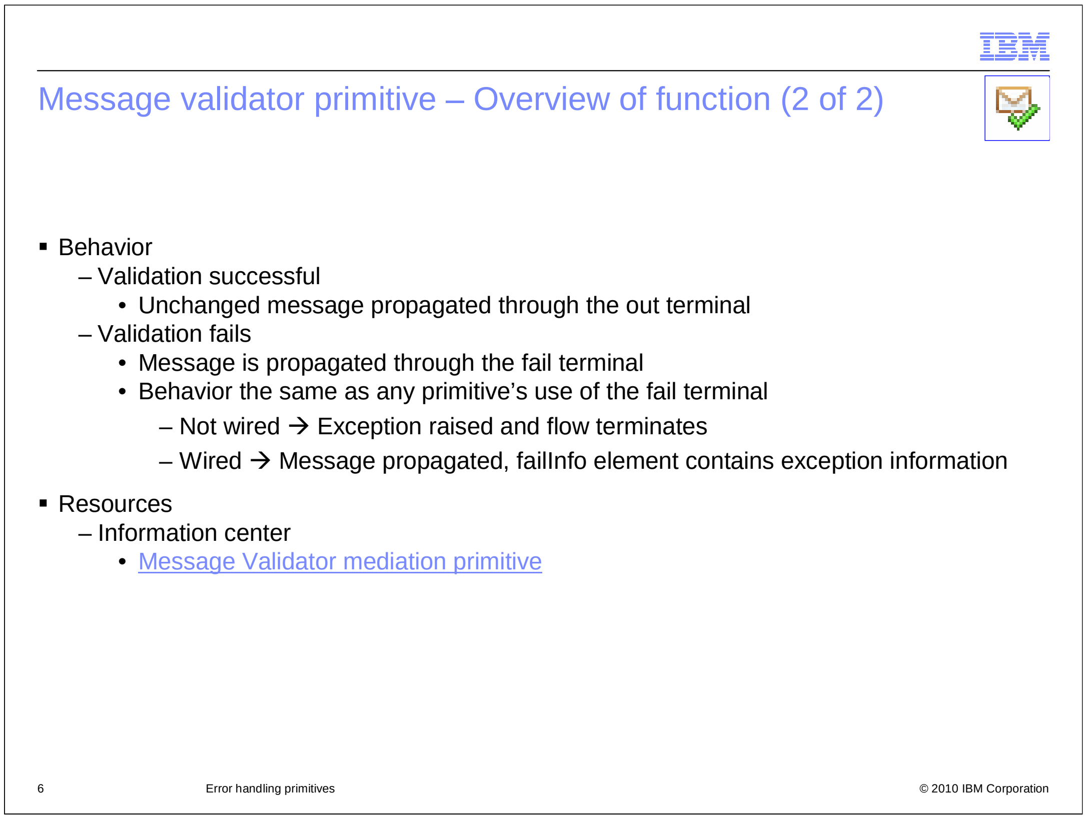
Message validator primitive – Overview of function (1 of 2)

- Validates SMO content against schema and assertions
- Checking includes the follow elements as applicable
  – Overall SMO schema
  – Schema of SMO body (message type)
  – Configured schema of SMO contexts
  – Downcast type assertions of weakly typed elements (such as those defined with the set message type primitive)
- Primitive is configurable to specify what portion of the SMO you want validated
- Enabled property can be promoted to allow validation to be toggled off and on

5 Error handling primitives © 2010 IBM Corporation

The message validator primitive is used to validate the contents of the service message object (SMO) against the schema and assertions defined at that point in the flow. There are several components that make up the schema for which the content of the message must conform. The first is the overall SMO schema defining the headers, context and body of the SMO. The schema for the body is dependent upon the message type defined for the input terminal of the message validator primitive. There are also business objects used to configure the transient, correlation and shared contexts. Additionally, the set message type primitive might have added assertions to weakly typed fields, defining a more specific type. All of these factors are considered when doing the validation.

The primitive is configurable to enable you to provide an XPath expression to that portion of the SMO you want to have validated. This allows you to have a section of the SMO validated without the performance overhead of validating the entire SMO. Also, there is an enabled property, which when promoted allows an administrator to toggle the validation off and on.



Message validator primitive – Overview of function (2 of 2)

- Behavior
  - Validation successful
    - Unchanged message propagated through the out terminal
  - Validation fails
    - Message is propagated through the fail terminal
    - Behavior the same as any primitive's use of the fail terminal
      - Not wired → Exception raised and flow terminates
      - Wired → Message propagated, failInfo element contains exception information
- Resources
  - Information center
    - Message Validator mediation primitive

6 Error handling primitives © 2010 IBM Corporation

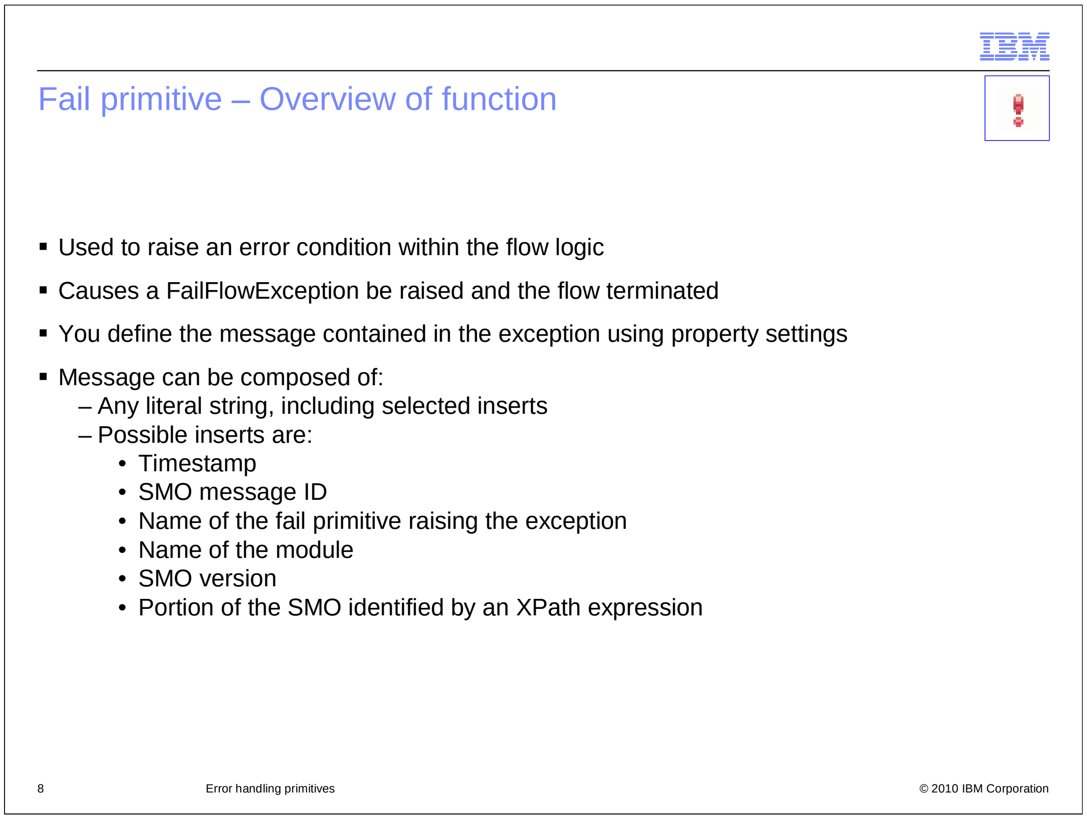
When the validation is successful, the primitive fires the out terminal, propagating the unchanged message. If the validation is not successful, the fail terminal is fired. Here, as with any other primitive, the behavior depends upon whether the fail terminal is wired. When the fail terminal is not wired, an exception is raised and the flow terminates. When the fail terminal is wired, the failInfo element of the SMO is populated with the failure information, and the otherwise unchanged message is propagated to the wire target.

A link is provided here to the message validator mediation primitive documentation in the information center, where you will find more information on its usage and the definition of its properties.

Section

Fail primitive

The fail primitive is addressed in this section.

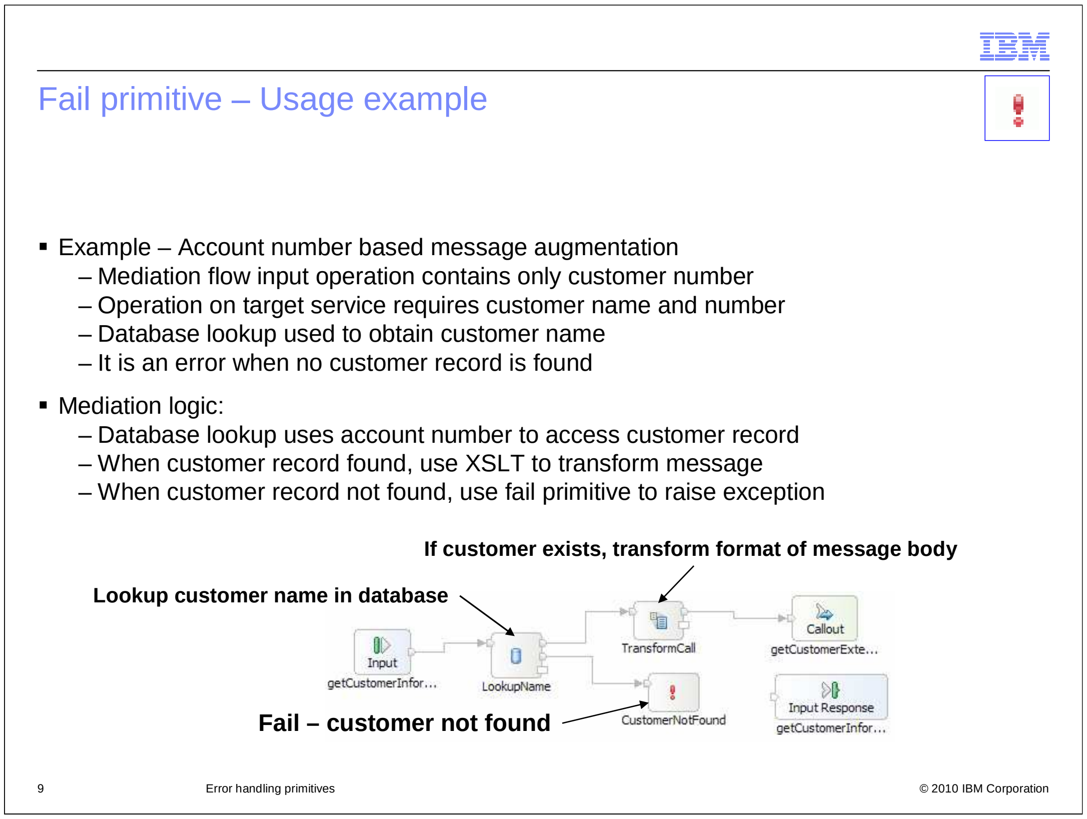


Fail primitive – Overview of function

- Used to raise an error condition within the flow logic
- Causes a FailFlowException be raised and the flow terminated
- You define the message contained in the exception using property settings
- Message can be composed of:
  - Any literal string, including selected inserts
  - Possible inserts are:
    - Timestamp
    - SMO message ID
    - Name of the fail primitive raising the exception
    - Name of the module
    - SMO version
    - Portion of the SMO identified by an XPath expression

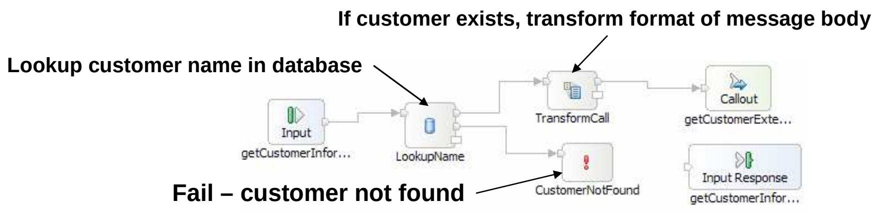
8 Error handling primitives © 2010 IBM Corporation

The fail primitive enables you to raise an error condition within your flow logic. It raises a FailFlowException, which is logged, and the flow is terminated. You can define the contents of the message, which is included in the FailFlowException, by setting the error message and root path properties. This message is constructed from a literal string that you specify, along with optional predefined inserts that you can place anywhere within the message. The inserts available are a timestamp, the message ID of the SMO, the name of the fail mediation primitive, the name of the module and the version of the SMO. The last insert is any portion of the SMO that you have specified using the root path property.



Fail primitive – 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.

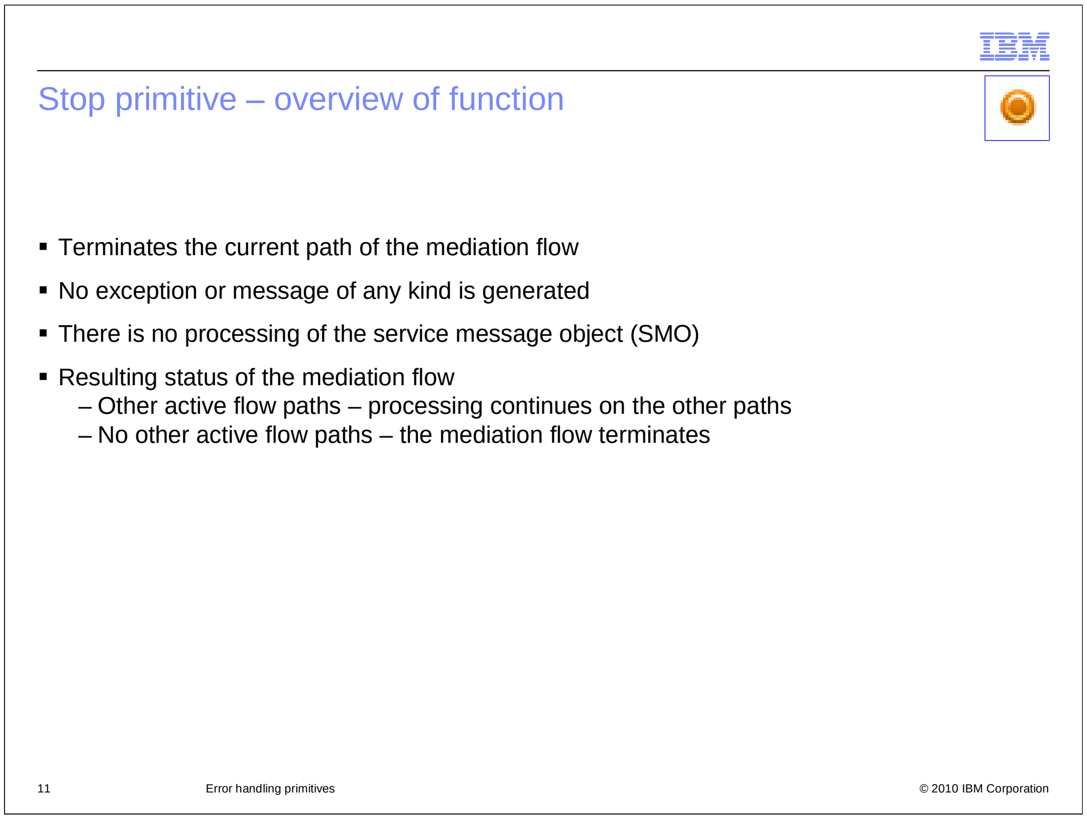
The mediation flow is shown at the bottom of the slide. The input node is wired to a database lookup, which obtains the customer name from the database. If the lookup succeeds, the out terminal is wired to an XSL transformation primitive used to modify the message body. The modified message is then passed 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

Stop primitive

10 Error handling primitives © 2010 IBM Corporation

The stop primitive is discussed in this section.

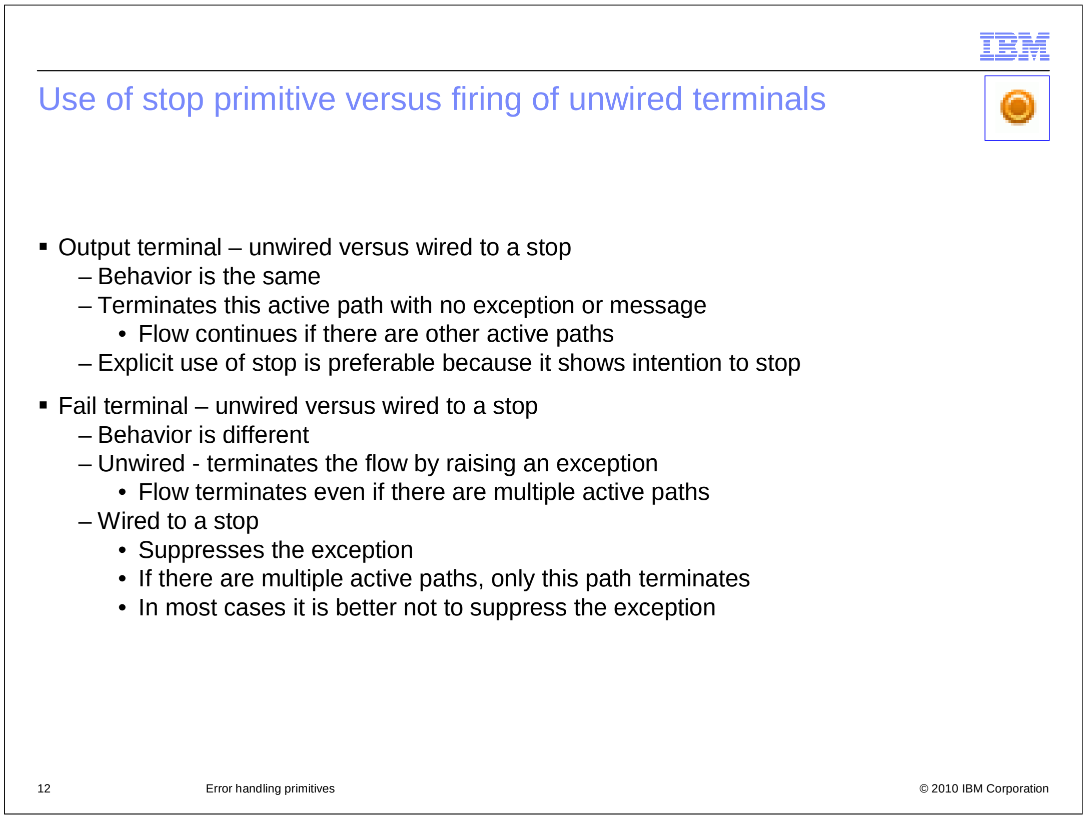


Stop primitive – overview of function

- 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)
- Resulting status of the mediation flow
  - Other active flow paths – processing continues on the other paths
  - No other active flow paths – the mediation flow terminates

11 Error handling primitives © 2010 IBM Corporation

The stop primitive is used to terminate the path on which it is wired, without an exception being raised, or a message being generated. No processing of the SMO occurs. The overall result on the mediation flow depends upon the nature of the flow. If the flow has multiple paths, and other paths are still active, the processing continues on one of the other paths. If there are no other flow paths active, the mediation flow terminates.



Use of stop primitive versus firing of unwired terminals

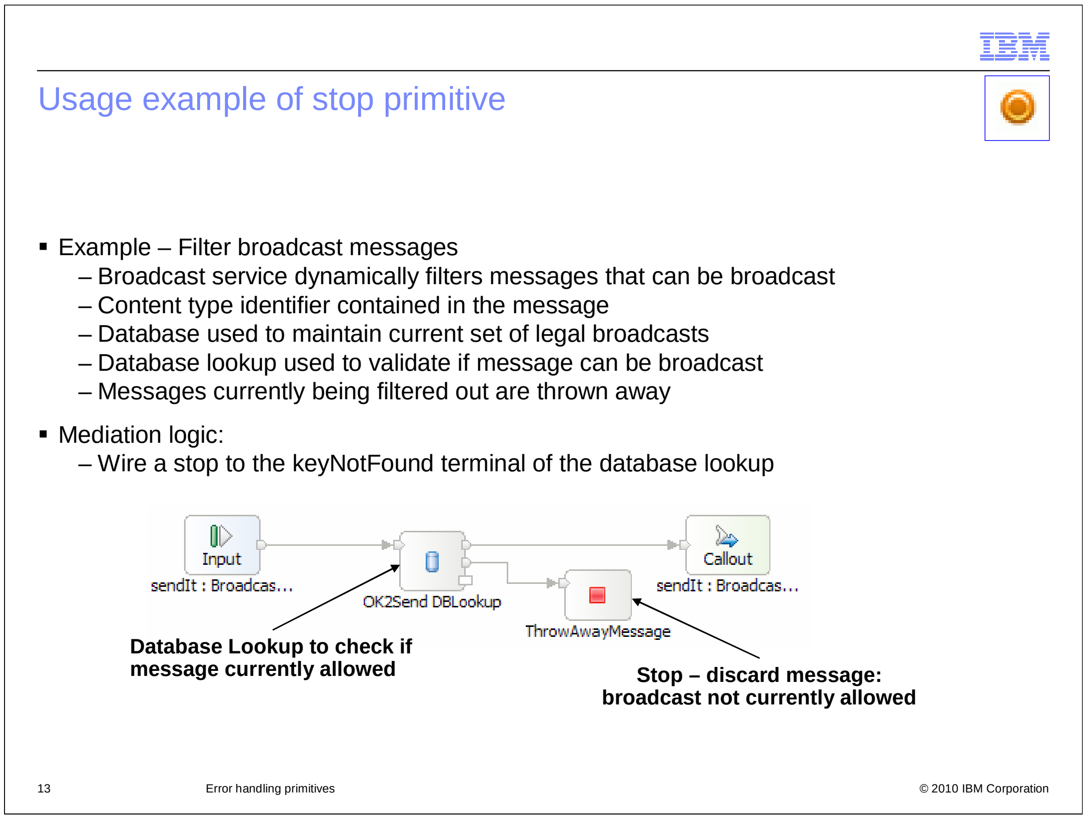
- Output terminal – unwired versus wired to a stop
  - Behavior is the same
  - Terminates this active path with no exception or message
    - Flow continues if there are other active paths
  - Explicit use of stop is preferable because it shows intention to stop
- Fail terminal – unwired versus wired to a stop
  - Behavior is different
  - Unwired - terminates the flow by raising 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

12 Error handling primitives © 2010 IBM Corporation

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.

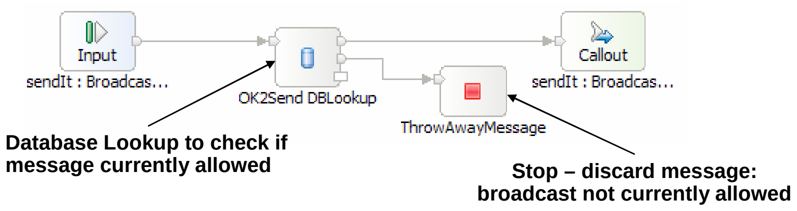
The first case to consider is for an output terminal. In this case the behavior is the same if the terminal is left unwired as opposed to being wired to a stop primitive. The active path is terminated with no exception or message. If this was the only active path, the mediation flow ends, but if there 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 raised 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.



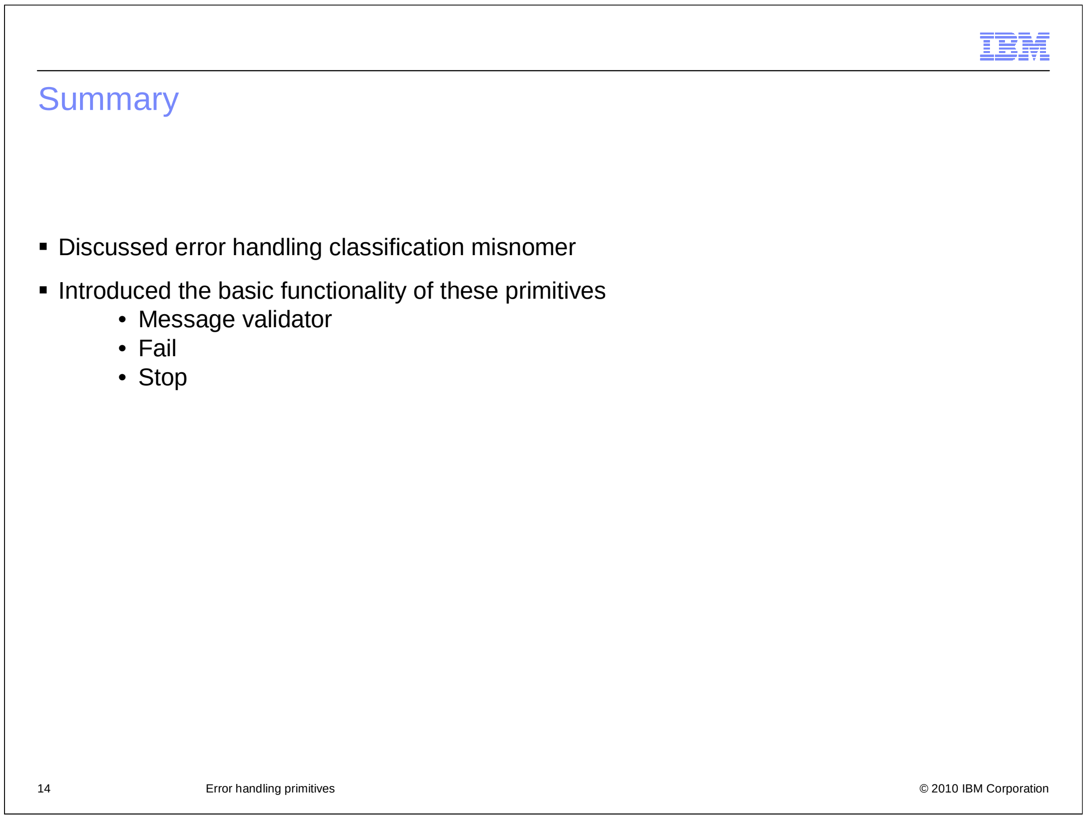
Usage example of stop primitive

- 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


Input
sendIt : Broadcas...
OK2Send DBLookup
Callout
sendIt : Broadcas...
ThrowAwayMessage
Database Lookup to check if message currently allowed
Stop – discard message: broadcast not currently allowed

This slide provides a usage example for the stop primitive.

The requirement being addressed is for a mediation that selectively forwards 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, passing the message through the out terminal to the callout for the broadcast service. If the lookup is not successful, the keyNotFound terminal is wired to a stop mediation primitive, which essentially causes the message to be discarded.

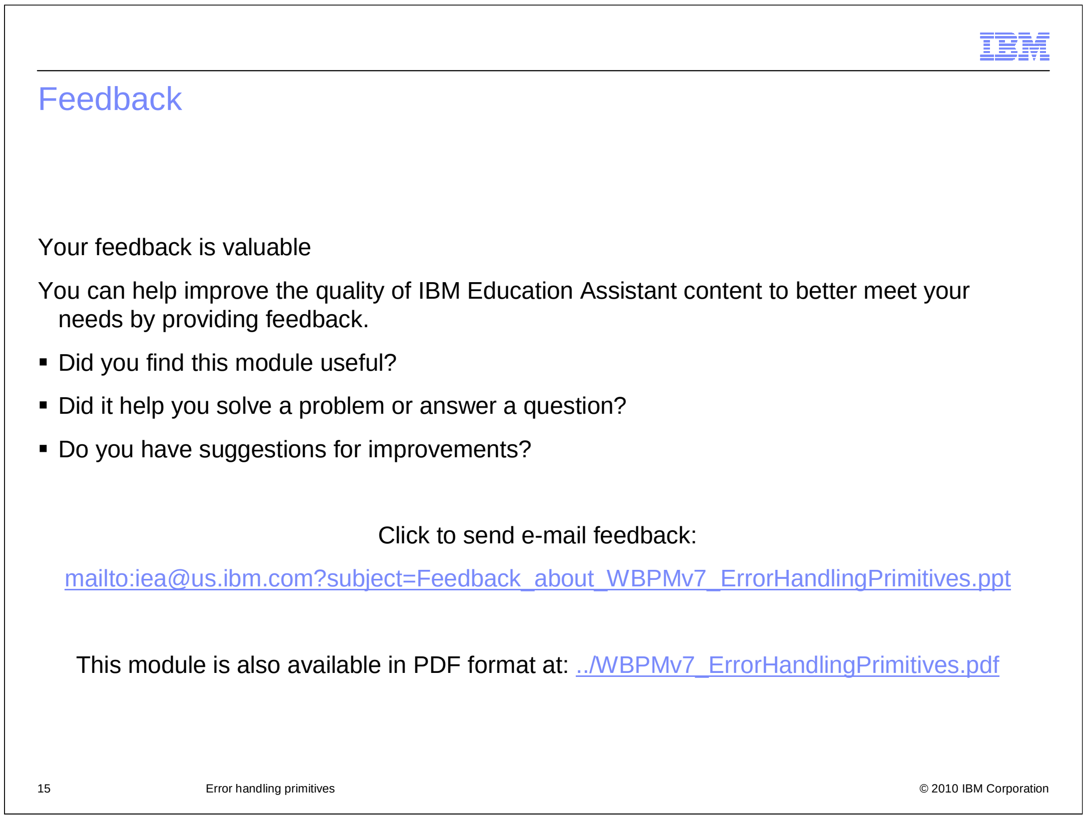


Summary

- Discussed error handling classification misnomer
- Introduced the basic functionality of these primitives
  - Message validator
  - Fail
  - Stop

14 Error handling primitives © 2010 IBM Corporation

In summary, this presentation started out by discussing the appropriateness of the error handling classification of these primitives. It then provided a description of the basic functionality of each, namely the message validator primitive, the fail primitive and the stop 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_ErrorHandlingPrimitives.ppt

This module is also available in PDF format at: ../WBPMv7_ErrorHandlingPrimitives.pdf

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

Trademarks, disclaimer, and copyright information