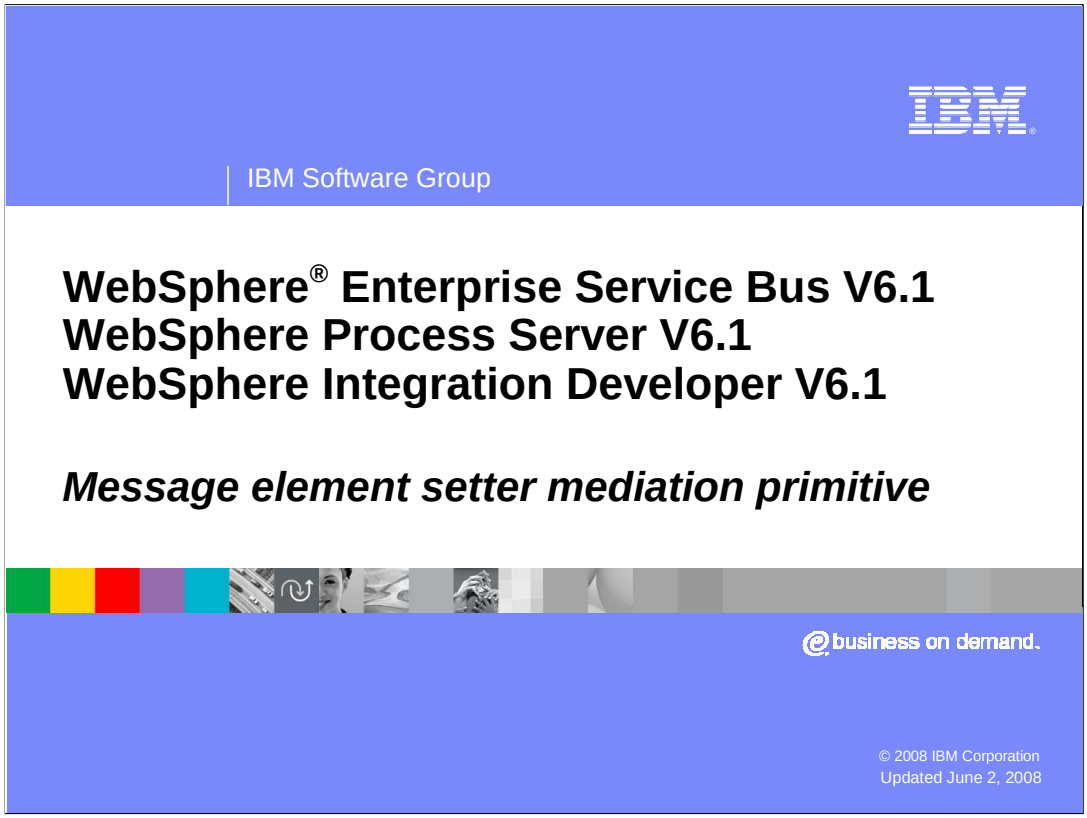


This presentation provides a detailed look at the message element setter mediation primitive.

Goals

- Understand the message element setter primitive


Message element setter

  - Overview of function
  - Use of terminals
  - Definition of properties
  - Processing details
  - Error handling
  - Example usage

The goal of this presentation is to provide you with a full understanding of the message element setter mediation primitive.

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.

An overview of the function provided by the message element setter primitive is presented, along with information about the primitive's use of terminals and its properties. Specific details of the processing behavior are described, followed by the error handling characteristics. Finally, a usage example of the message element setter primitive is provided.

Overview of function

- Updates to the service message object (SMO)
  - Assignment of a constant value
  - Copying from one part of an SMO to another
    - Leaf element
    - Sub-trees, provided source and target types match
  - Appending to an array
  - Deleting elements
    - Setting the element value to “null”
- XPath expressions used to identify elements
  - Target elements
  - Source elements of a copy operation
- Multiple elements can be set within the same primitive
- Easier than coding a custom mediation, XSL transformation or business object map primitives
- More efficient than XSL transformation and business object map primitives (updates are made in place)

The function of the message element setter primitive is to enable an easy and efficient mechanism to make updates to the service message object (SMO). There are four different types of updates that can be made. The first capability is the assignment of a constant value to a leaf element of the SMO. Secondly, a copy capability is provided which allows you to copy from one part of the SMO to another. The copy might be for a leaf element or for a sub-tree, provided that the source and target sub-trees have a matching structure. Similar to the copy operation is the append, which enables you to add an element to the end of an array, providing the array target and source types match. Finally, an element can be deleted. This does not actually delete the element completely from the SMO, but rather sets the value of the element to null.

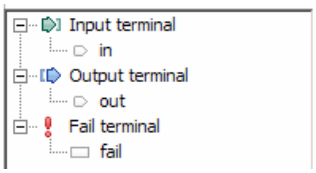
XPath expressions are used to identify the target elements in the SMO that are updated by the primitive. The source elements of copy and append operations are also identified using XPath expressions.

The message element setter primitive allows multiple elements to be updated. It makes use of a table property where each row of the table defines a single update.

The other primitives that can be used to perform the same kind of function are the custom mediation, business object map and the XSL transformation primitives. The message element setter primitive provides an easier mechanism to define the updates than these other primitives. In addition, the updates made by a message element setter are done in place rather than making a totally new copy of the SMO. Therefore, it is much more efficient at runtime than the XSL transformation or business object map primitives.

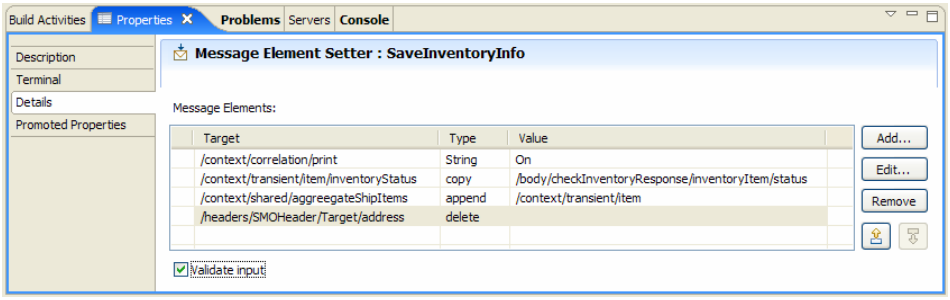
Terminals

- Terminals:
  - Input terminal
  - One Output terminal
  - Fail terminal
- All terminals must be for the same message type



The message element setter primitive has one input terminal, one output terminal and a fail terminal. The output terminal must be for the same message type as the input terminal, since the message element setter primitive does not modify the type of the message body. Shown here is a message element setter primitive with its terminals and the terminals as seen in the properties view.

Properties

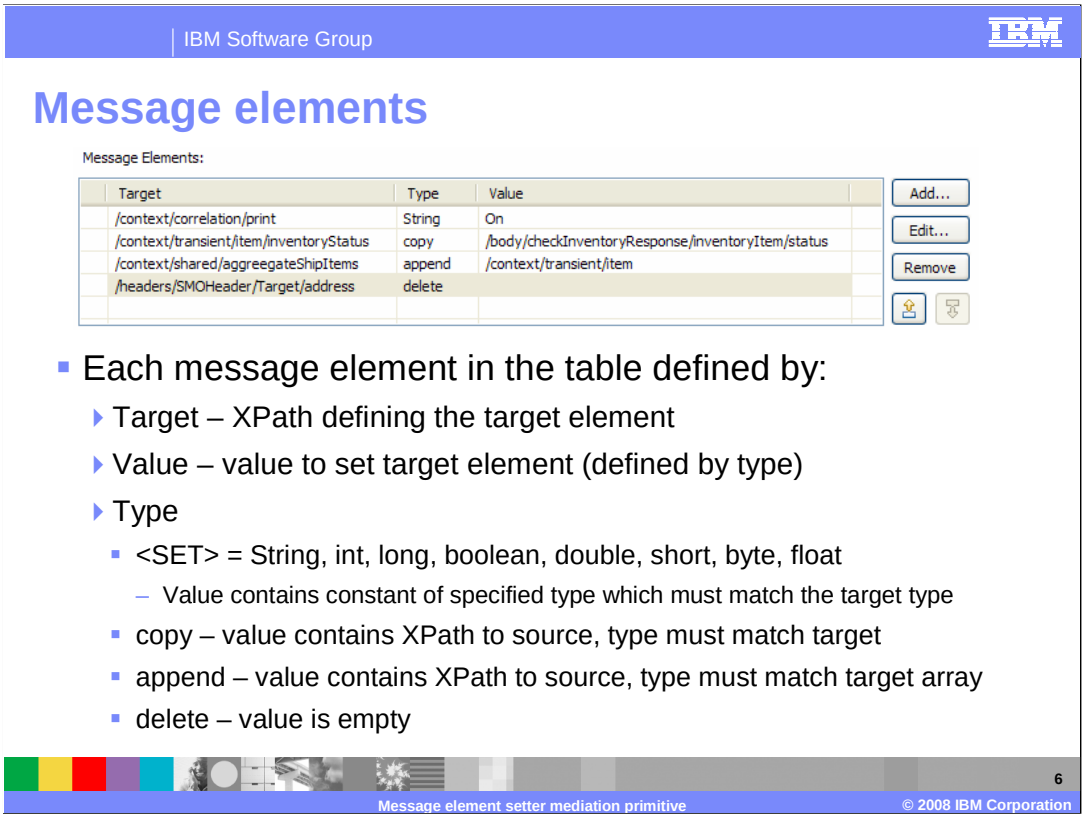


- Message elements
  - Table defining the ordered list of elements to be set
  - Full description on next slide
- Validate input
  - Validate incoming message is of the expected type
  - Ensure it meets constraints defined by its type
    - For example: minOccurs, maxValue

The message element setter primitive has two properties.

The first, the Message Elements property, is a table that provides an ordered list of elements in the SMO that are updated by the primitive. This table is examined in detail on the next slide.

The Validate input property performs a validation of the incoming SMO to ensure it is of the expected type and that it meets the constraints defined by its type. Performing the validation involves runtime processing overhead. Therefore, this should only be selected where there is a possibility that the input SMO might not conform to the specified type.



The Message Elements table based property defines an ordered list of updates made to the SMO, where each row of the table defines an individual update. The table contains three columns used to define the updates.

The Target column contains an XPath expression defining the element in the SMO that is to be updated.

The Value column defines what is to be set in the target element. How the value is interpreted depends upon what operation is defined by the Type column.

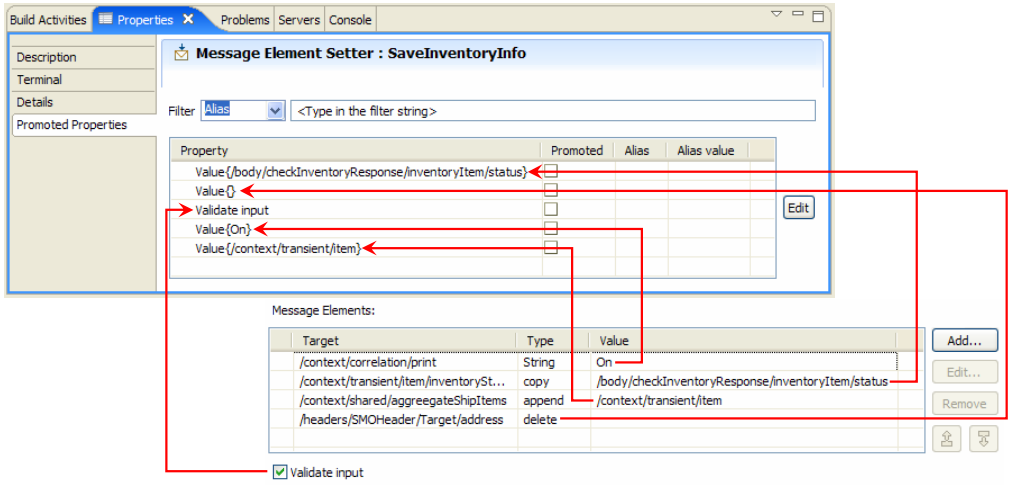
When the setting of a constant value into the target element is required, the Type column defines the type of the constant, such as String or int. It must match the type of the target element as defined in the SMO. Valid values for a constant type are shown on the slide.

When the update is a copy operation, the Type column contains copy and the Value column contains an XPath expression identifying the source element that is to be copied. The source and target types can be simple or complex, but must be of the same type.

When the type column contains append, the Value column contains an XPath expression identifying the source element that is to be appended to the target array. Similar to copy, the source and target types must match.

Finally, the Type column can contain ‘delete’ to signify that the element is to be set to null. In this case the Value column is left blank.

Promoted properties

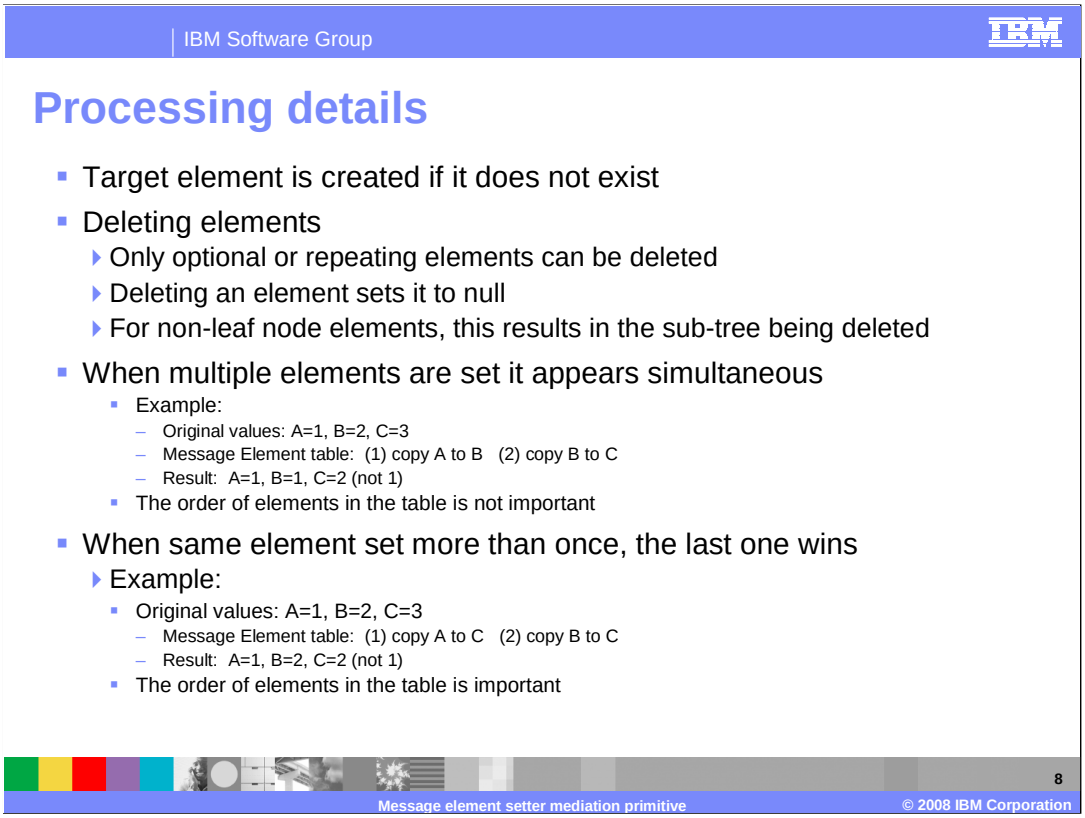


- Promotable
  - Validate input
  - Message elements table (Value column)

This slide examines the Promoted Properties panel for the message element setter primitive. Both of the primitive's properties are promotable.

Promoting the Validate input property allows an administrator to turn validation of the SMO on and off. This enables the performance advantage realized by not doing validation of the input SMO, while at the same time enabling the administrator to turn on validation for problem determination if the need arises to debug a problem.

In the Message Elements table based property, the Value column can be promoted. In the above screen captures, the four types of SMO updates are shown in the Message Elements table, with pointers to how they appear in the Promoted Properties table. Promoting a set constant, copy or append can be useful, depending upon your application scenario. Since the delete does not contain a value, it does not make sense to promote a row for a delete. However, the entry still appears in the Promoted Properties table.



For some cases, it can be important to understand the nuances of behavior exhibited by the message element setter primitive. Several of the processing details for the primitive are provided here.

When the target XPath expression identifies an element in the SMO that does not currently exist in the SMO, it is created.

When deleting elements, there are a few things to be considered. First of all, only optional or repeating elements can be deleted. When an element is deleted, it is not removed from the SMO, rather it is set to null. Also, if the element is not a leaf node element, setting it to null results in the sub-tree for that element being deleted.

Although the table is an ordered list of updates, the results of processing updates to multiple elements appears to have occurred simultaneously. For example, suppose the table specifies to copy A to B, and then specifies to copy B to C. The result is that C is set to the original value for B, not to the value for A.

Order is important when the same element is updated more than once. In this case, the last update is the effective one. For example, suppose the table specifies to copy A to C, and then specifies to copy B to C. The result is C contains the original value for B since that was the last update to C.

Error processing

- MediationRuntimeException thrown for:
  - Target or source XPath expression syntax is not valid
  - Value does not match type set in table
    - For example: Type=int, Value=abc
- MediationBusinessException (Fail terminal flow)
  - Source XPath expression specifies non-existent location
  - Validate Input is set and incoming SMO does not pass validation
  - Copy between elements of incompatible types
- Empty Message Element table
  - Setting no elements is not considered an error
  - Mediation primitive is effectively a no-op

The error processing details and considerations are examined in this slide.

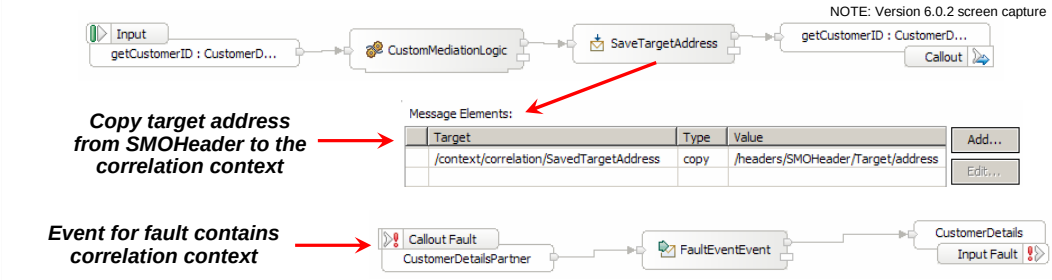
A MediationRuntimeException is thrown when either the source or target XPath expression syntax is not valid. It is also thrown when the value in the table does not match the type. An example of a type mismatch is when the type is integer and the value is an alphabetic string, such as “abc”.

When a MediationBusinessException occurs, the flow passes through the fail terminal for the message element setter if it is wired. This exception occurs if the source XPath expression for a copy or append operation specifies a location that does not exist in the SMO. It also occurs when the Validate input property is set and the SMO does not pass validation. Another reason for the occurrence of a MediationBusinessException is when the source and target elements of a copy or append operation are not of compatible types.

It is not considered an error condition when the message element table property contains no rows. In this case, the message element setter primitive is effectively a no-op. The SMO is not updated and no error is raised.

Usage example

- Raise event containing target address if service returns fault
  - Request flow
    - Has logic that includes setting the target address of the service
    - Message element setter - saves the target address in the correlation context
  - Response flow
    - When a fault is returned an event is emitted
    - Event contains entire SMO (includes fault information and target address)



This slide describes an example usage of the message element setter primitive. The screen capture is taken from the mediation flow editor using WebSphere Integration Developer version 6.0.2. You might notice some differences in the visual appearance from version 6.1, but the flow being described is the same between these versions.

The purpose of the scenario is to be able to raise an event that contains the target address of the service provider if the service returns a fault. In the scenario, the callout to the service provider is a dynamic callout, making use of the target address set into the SMO by logic in the mediation flow. This might be a custom mediation with logic to set the target address or possibly the result of an endpoint lookup primitive. In any case, the target address needs to be preserved across the call in case the service provider returns a fault. This is where the message element setter comes in, copying the target address from the SMO header to an element in the correlation context. Then on the response flow, the callout fault node is wired to an event emitter which places the entire SMO into the event. This produces an event that contains the target address and the fault information.

Summary

- Examined the message element setter primitive

Message element setter

- Overview of function
- Use of terminals
- Definition of properties
- Processing details
- Error handling
- Example usage

In summary, this presentation provides an overview of the function provided by the message element setter primitive, along with information about the primitive's use of terminals and its properties. Details of processing behavior were described followed by the error handling characteristics. Finally, a usage example of the message element setter primitive was 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_WBIV61_MessageElementSetterPrimitive.ppt

This module is also available in PDF format at:
../WBIV61_MessageElementSetterPrimitive.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.