



IBM Software Group

**WebSphere® Enterprise Service Bus V6.0.2**  
**WebSphere Process Server V6.0.2**  
**WebSphere Integration Developer V6.0.2**

***Accessing service message objects***



@business on demand.

© 2007 IBM Corporation  
Updated April 12, 2007

This presentation will examine how Service Message Objects are accessed from within a mediation flow.

## Goals

- Understand the accessing of service message objects (SMO)
  - ▶ Access and manipulation of SMO content
  - ▶ Manipulation of message type
- Approaches used
  - ▶ XPath
  - ▶ XSL transformations (XSL Style sheets)
  - ▶ Java™ code

The goal of this presentation is to provide you with an understanding of how to access Service Message Objects (SMO). Accessing SMOs occurs within mediation flows and involves the reading, writing and updating of the elements of an SMO. Updating an SMO might also include modifying the message type, which occurs when the structure of the payload of the message is changed. Each of the mechanisms for accessing and manipulating SMOs, which includes the use of XPath expressions, XSL Transformations using XSL style sheets and Java code will also be covered.

## Manipulating SMOs

- Three ways to access and manipulate SMOs
  - ▶ XPath 1.0 expressions
    - Primary mechanism for accessing the SMO
    - Used in some form by all of the mediation primitives
    - Identify elements to read, update or process conditional expressions
  - ▶ XSL style sheets
    - Used by the XSLT mediation primitive
    - Normally used to modify message type within a flow
    - Also used to manipulate SMO content without changing message type
  - ▶ Java code
    - Used by the Custom Mediation primitive
    - Generic DataObject APIs
    - SMO APIs
    - Can access and update content and can also modify message type

This slide provides an overview of the three different mechanisms for accessing and manipulating the contents of an SMO. XPath 1.0 is the primary mechanism for accessing SMOs and is employed in one form or another by all of the mediation primitives. XPath expressions can be used to identify elements of the SMO to read or update and can also be conditional expressions to be evaluated. XSL style sheets are used by the XSLT mediation primitive, which is typically used to modify the message type within the flow by changing the structure of the body of the message. They can also be use to update the content of the message without changing the message type. Java code is used by the Custom Mediation primitive to access the SMO. The generic DataObject APIs can be used, making use of XPath expressions to identify properties within the SMO. There are also SMO specific APIs, which provide type safe access to the properties within the SMO. Using Java code, you can update the message content and also have the ability to modify the message type by changing the structure of the message body. In the next series of slides, each of these mechanisms will be examined in detail.

## Section

# *XPath*

This section takes a closer look at the use of XPath for accessing SMOs.

## Manipulating SMOs – XPath

- All mediation primitives use XPath in some form
  - ▶ XPath expression identifying a specific property within the SMO
  - ▶ XPath conditional expression to be evaluated
- Root property
  - ▶ Used to specify what part of the SMO is visible to the primitive
  - ▶ Values selected from a drop down:
    - Typically the values are: **/ /body /context /headers**
    - Some instances of root are more restrictive
    - Other instances of root are less restrictive and allow use of custom XPath
- Other properties using XPath
  - ▶ Specified using the XPath expression builder
  - ▶ Builder accessed using:
    - **Custom XPath...** button
    - **...** Button from a table cell
  - ▶ Select a target expression, optionally add a condition, or override

5

Accessing service message objects

© 2007 IBM Corporation

XPath expressions are used in some form by all of the mediation primitives and are typically simply a qualified path based on the SMO schema that identifies a property within the SMO. The property identified might be a complex DataObject type or a simple element with a primitive type. At other times the XPath expression might be a conditional expression to be evaluated true or false relative to the value of an element in the SMO. There are generally two different ways that an XPath expression is specified within the properties for a mediation primitive. The first is the use of a Root property, which identifies that portion of the SMO that is to be visible to or used by the primitive. In most cases, the Root property consists of four possible values that are selected using a drop down box. The values are / (slash) indicating the entire SMO, /body indicating the body or payload of the SMO, /context referring to the context of the SMO or /headers referring to the SMO headers. However, the use of the Root property is not always exactly like this. One case is more restrictive and only provides two possible choices and another case allows the use of a custom XPath expression that can identify any property within the SMO. Other uses of XPath by mediation primitives make use of the XPath expression builder dialog. The dialog allows you to traverse the SMO schema to select a particular property or element within the SMO and optionally to define a conditional expression involving that element. This dialog is accessed using a Custom XPath... button next to an entry field for an XPath expression or through the ... button in a table cell requiring an XPath expression. The next couple of slides will look more closely at the XPath expression builder dialog.

## Manipulating SMOs – XPath expression builder

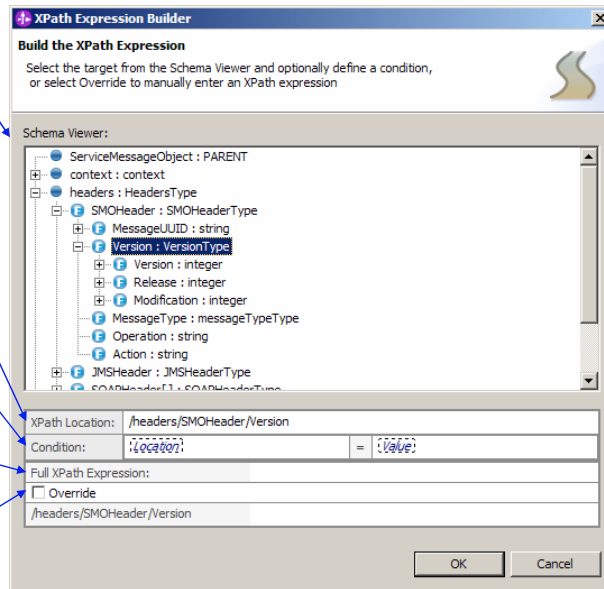
Use the Schema Viewer to navigate SMO to the appropriate XPath location

Selected location will be displayed in the XPath Location field

Optionally, Condition field can be used to define a conditional expression

Full XPath Expression field contains the complete expression

Override lets you manually edit the generated expression



6

Accessing service message objects

© 2007 IBM Corporation

The screen capture on this slide shows the XPath Expression Builder dialog.

The top panel in the dialog is the Schema Viewer, which allows you to traverse the SMO schema to identify the location within the SMO that you are interested in. When you select a specific location, it will be displayed in the XPath Location field and in the Full XPath Expression field. The condition field can be used to create an optional conditional expression. The condition field consists of a location and a value where an is equal to comparison is performed between the value at the location and the value in the condition. The location in the condition is used to further qualify the XPath Location when it identifies a complex type, as the condition can only be performed against a simple element. The conditional expression is added to the Full XPath Expression field. Finally, if the value that has been built in the Full XPath Expression field is not the required expression, the override check box can be selected and the full expression can then be edited as needed.

## Manipulating SMOs – XPath expression builder

Value defined using  
Schema Viewer

XPath Location:	/headers/SMOHeader/Version	
Condition:	{location}	= {value}
Full XPath Expression:		
<input type="checkbox"/> Override		
	/headers/SMOHeader/Version	

Added condition

XPath Location:	/headers/SMOHeader/Version	
Condition:	Release	= 1
Full XPath Expression:		
<input type="checkbox"/> Override		
	/headers/SMOHeader/Version[Release=1]	

Override  
Changed condition to  
> rather than =

XPath Location:		
Condition:	{location}	= {value}
Full XPath Expression:		
<input checked="" type="checkbox"/> Override		
	/headers/SMOHeader/Version[Release>1]	



This slide shows an example of the XPath Expression Builder through the steps of identifying a location, adding a condition and overriding the full expression. The top screen capture shows the result after using the Schema Viewer to select the location /headers/SMOHeader/Version. You can see that this value has been placed into the XPath Location and Full XPath Expression fields. In the center screen capture, a condition has been added, the location has been further qualified by Release and the value has been set to 1. You can see that the Full XPath Expression has been updated to contain this conditional expression. The requirement for this expression is to check for Release greater than 1 rather than equal to 1. The bottom screen capture shows that the Override check box has been selected and the Full XPath Expression has been edited to change the equal operator to a greater than operator.


## Section

# ***XSL transformation (XSL style sheets)***

This section takes a closer look at the use of XSL Transformations, which use XSL style sheets for accessing SMOs.



## Manipulating SMOs – XSL style sheets

- XSLT primitives process XSL style sheets at runtime
- The XSL style sheet used by the XSLT can be:
  - ▶ Generated from an XML map created with the XML mapping editor
  - ▶ Created directly using the XSL editor
  - ▶ Configuration of XSLT primitives has been improved in version 6.0.2
- 
  - See the XSLT mediation primitive presentation for details
- Typical usage is for modification of message type in a flow
  - ▶ When input and callout nodes have different message types
  - ▶ To reply using the input response node in a request flow
  - ▶ To reply with a fault using the input fault node to report a flow error
- Also used to manipulate SMO content without changing message type
  - ▶ Use XSLT functions (string manipulation for example)
  - ▶ Logical processing with XSL choose/otherwise statements

9

Accessing service message objects

© 2007 IBM Corporation

The XSLT mediation primitives use XSL style sheets at runtime to manipulate the SMO. There are two ways to define an XSL style sheet. The first is to use the XML Mapping Editor to define a mapping, which is then used to generate an XSL style sheet. The second approach is to use the XSL Editor to edit the style sheet directly. Note that in version 6.0.2, the behavior of the configuration panels in WebSphere Integration Developer for the XSLT primitive has been improved. They provide a better user experience in the handling of XML maps and XSL style sheets. See the XSLT mediation primitive presentation for details of these enhancements.

Typically the XSL style sheet is used within a mediation flow to modify the message type, and this is required when a mediation flow has an Input node and a Callout node that have terminals for different message types. There are also other situations that require modification of the message type. For instance, if the Input Response node is going to be used in the request flow, the request message must be transformed into a response message. Another instance is if an error was detected in the request flow and the message has to be transformed to a fault message to be returned using the Input Fault node.

XSL style sheets can also be used to manipulate the content of the SMO without changing the message type. This is done by using XSLT functions that provide functionality such as string manipulation or numeric computation. There is also some logical processing that can be defined using the XSL Choose/Otherwise statements.

The XSL style sheets provide a rich set of capabilities for manipulating the content of an SMO.

## Manipulating SMOs – XML mapping editor

- Top panel used to define the mappings
  - ▶ Drag elements from source to target to define a move
  - ▶ Pop-up menus provide additional choices, for example:
    - Applying XSLT functions to target elements
    - Defining a match mapping between source and target sub-trees
- Bottom panel shows the currently defined mappings
- XSLT is quite powerful and therefore so is the XML mapping editor

Example: Define map to transform between input and callout operations

Target	Source	Applied Function/Grouping
smo	smo	
body		
getCustomerExtendedInfo		
customerID	customerID	
portfolioRequested		string

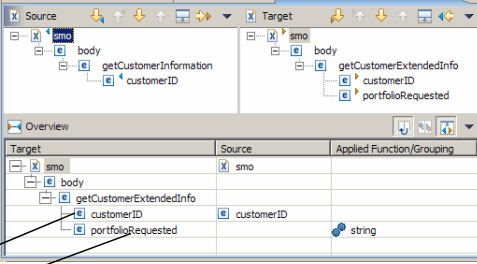
This slide examines the XML Mapping Editor, which is used to define a map, which can then be used to generate an XSL style sheet. The top panel is used to define the mappings, while the bottom panel shows those mappings that have already been defined.

On the top panel, the source XML is shown on the left and the target XML on the right. There are toolbars for use with both the source and the target. Elements of the source can be dragged and dropped onto elements in the target to define a move operation. A pop-up menu for elements in the target can also be used to select an XSLT function that can be configured to update the target element. Another way to define moves between the source and target is to use the Match Mapping menu item. This is a way to automatically define move operations for elements in the source and target that have the same name.

In the bottom panel note that the source and target are reversed from the top panel. The target XML is on the left, showing each of the elements that will be set. The center column contains the source elements of move operations and the right column contains the applied functions. In this example, there is a source interface with a `getCustomerInformation` operation that contains a `customerID` input. The target interface has a `getCustomerExtendedInfo` operation with `customerID` and `portfolioRequested` inputs. The map that is defined moves the `customerID` from source to target. The `portfolioRequested` is set using the XSLT string function. Opening the string function would show it has been initialized to set the value of `portfolioRequested` to the literal string "default".

## Manipulating SMOs – Resulting XSL style sheet

XML mapping editor  
and the resulting  
XSL style sheet



Target	Source	Applied Function/Grouping
smo	smo	
body		
getCustomerExtendedInfo		
customerID	customerID	
portfolioRequested		string

```

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0"
  xmlns:xalan="http://xml.apache.org/xslt"
  xmlns:tns_1="http://CustomerBackend/Customerservice"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:tns_10="http://PortfolioLibrary/CustomerserviceExtended"
  exclude-result-prefixes="xalan tns_1 xsl">
  <xsl:output method="xml" encoding="UTF-8" indent="yes" xalan:indent-amount="2em" />
  <xsl:strip-space elements="*" />
  <!-- =====>
  <xsl:template match="*" />
  <xsl:call-template name="body" />
  </xsl:template>
  <!-- =====>
  <xsl:template name="body">
    <body>
      <xsl:call-template name="getCustomerExtendedInfo" />
    </body>
  </xsl:template>
  <xsl:template name="getCustomerExtendedInfo">
    <getCustomerExtendedInfo>
      <customerID>
        <xsl:value-of select="/body/getCustomerInformation/customerID/text()" />
      </customerID>
      <portfolioRequested>
        <xsl:value-of select="string(default)" />
      </portfolioRequested>
    </getCustomerExtendedInfo>
  </xsl:template>
  <xsl:template match="*|@*|comment()|processing-instruction()|text()" />
  <xsl:copy>
  <xsl:apply-templates select="*|@*|comment()|processing-instruction()|text()" />
  </xsl:copy>
  </xsl:template>
</xsl:stylesheet>

```

11

Accessing service message objects

© 2007 IBM Corporation

This slide shows the XSL style sheet generated from the map created in the XML Mapping Editor described on the previous slide. In the style sheet, you can see that there is a template / (slash) that contains a template body. The template body contains a template getCustomerExtendedInfo, which is the target of the transformation. There are arrows from the XML Mapping editor to the XSL style sheet to illustrate how the map is reflected in the style sheet. You can see that the customerID in the target is set to the value from the source identified as /body/getCustomerInformation/customerID/text(). In addition, you can also see that portfolioRequested is set to the literal string "default".

## Section

# *Java Code*

This section takes a closer look at the use of Java code for accessing SMOs.

## Manipulating SMOs – Java code

- Java code can be used in Custom Mediation primitives
- Custom Mediation operation:
  - ▶ Is passed the SMO as a DataObject
  - ▶ Returns the SMO as a DataObject
  - ▶ Input/Output message types must match terminals of Custom Mediation
- DataObject API (commonj.sdo.DataObject)
  - ▶ Defined by the Service Data Object (SDO) specification
  - ▶ Provides a dynamic loosely typed interface to access an SMO
  - ▶ Augmented subset of XPath 1.0 can be used with accessor methods
  - ▶ Javadoc and specification available from IBM Developerworks
- ServiceMessageObject API
  - ▶ Provides strongly typed interface for well defined portion of SMO
    - Everything except the contents of the body, transient context and correlation context
  - ▶ Javadoc available from the Information Center

Custom Mediation primitives contain Java code, which defines the logic for the primitive. The Java operation called by the custom mediation takes the SMO typed as a DataObject as input and returns the SMO typed as a DataObject. The body portion of the SMO must conform to the message type defined for the input and output terminals of the Custom Mediation primitive. The Java code can make use of the commonj.sdo.DataObject APIs to access the SMO, which are defined by the Service Data Object specification, also known as SDO. These APIs provide a loosely typed interface for accessing the SMO and have accessor methods that make use of an augmented subset of XPath 1.0 to identify properties within the DataObject. The full javadoc for these APIs as well as the SDO specification are available from IBM Developerworks. There is also a set of SMO APIs, which provide a strongly typed interface for accessing the well defined portion of the SMO. In other words, these APIs understand the schema for the SMO except for the body, the transient context and the correlation context, which are unique to each individual flow. There is javadoc describing these APIs in the Information Center.

## Manipulating SMOs – Java code examples

- Compare DataObject API usage to SMO API usage
  - ▶ Code to access the MessageUUID field contained in the SMOHeader
    - Using DataObject with full path
 

```
// Get MessageUUID using DataObject and XPath
String uuid = input1.getString("headers/SMOHeader/MessageUUID");
```
    - Using DataObject and traversing down through each property
 

```
// Get Message UUID using DataObject and traverse properties
DataObject header = input1.getDataObject("headers");
DataObject SMOHeader = header.getDataObject("SMOHeader");
String uuid = SMOHeader.getString("MessageUUID");
```
    - Using the SMO strongly typed APIs
 

```
// Get Message UUID using SMO strongly typed APIs
ServiceMessageObject smo = (ServiceMessageObject)input1;
HeadersType headers = smo.getHeaders();
SMOHeaderType SMOHeader = headers.getSMOHeader();
String uuid = SMOHeader.getMessageUUID();
```
- Discovery of coding errors
  - ▶ Loosely typed DataObject errors such as misspelling a property name are not discovered until runtime
  - ▶ Strongly typed SMO errors are caught at compile time

14

Accessing service message objects

© 2007 IBM Corporation

This slide compares the use of the DataObject APIs to the SMO APIs by using three examples do the same thing. They each access the MessageUUID field that is in the SMOHeader.

The first code example uses the loosely typed DataObject APIs and a fully qualified XPath expression to identify the field. The XPath expression headers/SMOHeader/MessageUUID is used to obtain the MessageUUID string from the SMO DataObject.

The second code example also uses the loosely typed DataObject APIs, but in this case traverses down through each property individually. First, the headers DataObject is obtained from the SMO DataObject. Then the SMOHeader DataObject is obtained from headers DataObject. Finally, the MessageUUID string is obtained from SMOHeader DataObject.

In the third code example, the strongly typed SMO APIs are used. In this case, the first thing that must be done is to cast the input from DataObject to the ServiceMessageObject type. Using the ServiceMessageObject, the getHeaders operation can be used to obtain the headers. From the headers object the getSMOHeader operation can be used to obtain the SMOHeader. Finally, the getMessageUUID operation is used to get the MessageUUID string from the SMOHeader.

Contrasting these two approaches with respect to coding errors, the loosely typed DataObject APIs generally have coding errors surface at runtime, whereas the strongly typed SMO APIs tend to have coding errors caught at compile time. Therefore, the SMO APIs might be preferred over the DataObject APIs for ease in development and debugging.

## Section

# *Summary*

The following slide presents a summary of this presentation.

## Summary

- Examined service message objects in terms of:
  - ▶ Access and manipulation of SMO content
  - ▶ Manipulation of message type
- Approaches used
  - ▶ XPath
  - ▶ XSL transformations (XSL style sheets)
  - ▶ Java code

This presentation presented details about accessing Service Message Objects, including modifying their content and changing their message type. The use of XPath expressions, XSL Transformations using XSL style sheets and the use of Java code in Custom Mediations was also examined.



## 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](#)



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

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

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 NON-INFRINGEMENT. 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.

© Copyright International Business Machines Corporation 2007. All rights reserved.

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.

