



IBM Software Group

WebSphere® Enterprise Service Bus V6.1
WebSphere Process Server V6.1
WebSphere Integration Developer V6.1

XSL transformation mediation primitive



@business on demand.

© 2008 IBM Corporation
Updated June 13, 2008

This presentation provides a detailed look at the XSL transformation primitive, also called the XSLT primitive. The acronym XSLT stands for extensible style sheet language transformations, which is defined in a specification from the World Wide Web Consortium (W3C). The XSL transformations specification defines the syntax and semantics of a language for transforming XML documents into other XML documents.

Goals

- Understand the XSL transformation primitive



XSL transformation

- ▶ Overview of function
- ▶ Use of terminals
- ▶ Definition of properties
- ▶ Maps – creating, editing, defining transforms and testing
- ▶ Migration from previous releases
- ▶ Error handling
- ▶ Example usage



The goal of this presentation is to provide you with a full understanding of the XSL transformation 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 XSL transformation primitive is presented along with information about the primitive's use of terminals and its properties. The primitive uses XML maps to define the contents of the XSL style sheets. Therefore, the presentation looks at the XML maps to see how they are created and edited, the defining of transformations within the maps and how the maps are tested. The primitive has changed between versions 6.0.2 and 6.1, so the migration capabilities are discussed. Finally, error handling characteristics are presented followed by an example usage of an XSL transformation primitive.

Overview of function

- Modifies the service message object (SMO)
 - ▶ Uses extensible style sheet language (XSL) transformations (XSLT)
 - ▶ Message type and message content can be changed
 - ▶ Transformed message is validated
- XPath used to defined the root (starting point) of the transformation
- Maps are used to define the transformations
 - ▶ Created with the XML map editor
 - ▶ XSL style sheets are generated from the XML maps
- XSL style sheets can be specified directly



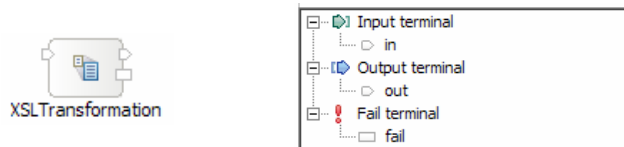
The purpose of the XSL transformation primitive is to modify the service message object (SMO). This is done using extensible style sheet language (XSL) transformations, also called XSLT. The XSL transformation primitive is capable of modifying the content of the SMO and of modifying the message type by restructuring the message body. The transformed message is validated to ensure it conforms to the constraints specified for it. This validation always occurs and is not optionally selected as it is for input messages on this and some other primitives.

There is a root property, which uses an XPath expression to define the starting point within the SMO for the transformation.

Transformations are defined by means of XML maps created using the XML map editor. The XML maps are then used to generate XSL style sheets, which are used by the runtime to perform the transformation.

Terminals

- Terminals:
 - ▶ Input terminal
 - ▶ One output terminal
 - ▶ Fail terminal
- Message type of input and output terminal
 - Same type – for manipulation of values within a message
 - Different type – for changing format of the message body



The XSL transformation primitive has one input terminal, one output terminal and a fail terminal. The output terminal can be for the same message type as the input terminal or for a different message type. When the message type is different, the transformation modifies the structure of the body of the message. Shown here is an XSL transformation primitive with its terminals, and the terminals as seen in the properties view.

Properties

- **Root**
 - ▶ XPath expression defining portion of service message object to transform
 - ▶ One of: /, /body, /context, /headers
 - ▶ Not editable on this panel
- **Mapping file**
 - ▶ Identifies either an XML map or an XSL style sheet
 - ▶ Browse... to select an existing XML map or XSL style sheet using the mapping file selection dialog
 - ▶ Edit... to edit the file specified using the XML map editor or XSL editor
 - ▶ New... to create a new XML map using the new XML mapping dialog
 - ▶ Not editable on this panel
- **Associated XSL**
 - ▶ The XSL style sheet that is generated from the XML map
 - ▶ File is automatically generated and updated when the mapping file is saved
 - ▶ Is not displayed when the mapping file is an XSL style sheet
 - ▶ Not editable on this panel
- **Validate input**
 - ▶ Validates if incoming message is of the expected type and meets specified constraints

XSL transformation mediation primitive © 2008 IBM Corporation

In the upper right is a screen capture of the Details panel from the Properties view for an XSL transformation primitive.

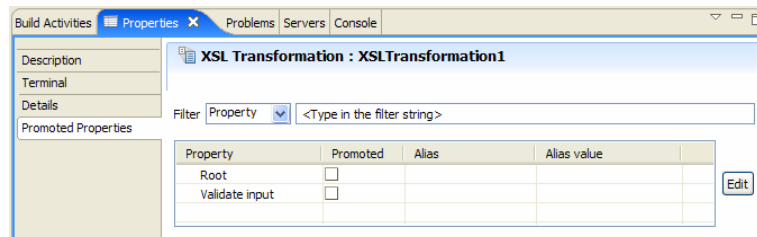
The **Root** property contains an XPath expression defining the major portion of the SMO on which the transformation is performed. Valid values for this property are / (forward slash) meaning the entire SMO, or /body, /context or /header, referring to the corresponding section of the SMO. This property is not editable on this panel. Its value is set by one of the dialogs that is used to define the mapping file.

The **Mapping file** property contains either the file name of an XML map that is used to generate an XSL style sheet or the name of an XSL style sheet. This field displays the file name and is not directly editable. It is set through use of the dialogs accessible using the buttons to the right of the field. The Browse... button opens the mapping file selection dialog that allows you to navigate and select an existing XML map or an existing XSL style sheet. Only style sheets that are not associated with an XML map are displayed and selectable. The Edit... button opens the appropriate editor for the configured mapping file, either the XML map editor or the XSL editor. The New... button opens the new XML mapping dialog, which enables you to define the characteristics of the map, and then places you into the XML map editor to define the transformations.

The **Associated XSL** property contains the name of the XSL style sheet that is generated from the XML map. This field is visible when the mapping file contains an XML map and is not visible when the mapping file contains an XSL style sheet. The XSL style sheet is automatically generated when the XML map is saved. This field is not editable.

The **Validate input** property is a check box used to indicate if incoming messages to the XSL transformation primitive are to be validated before processing. This ensures that the incoming message is of the expected type and that any constraints defined are not violated.

Promoted properties



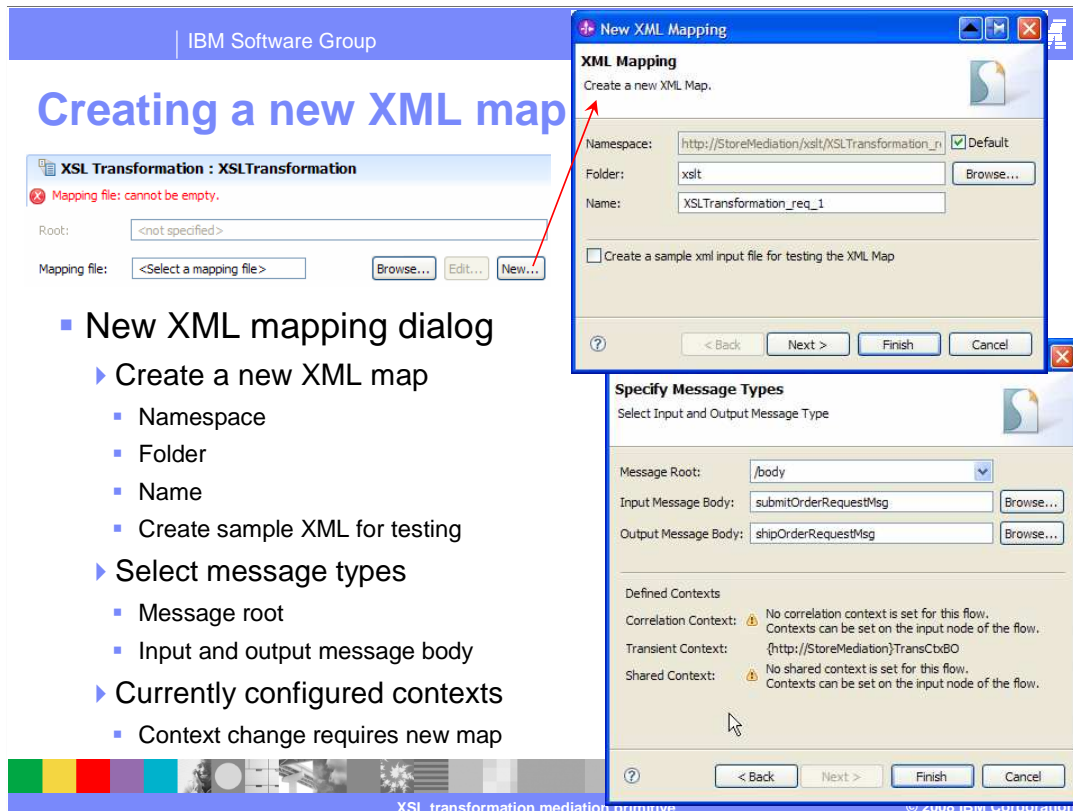
- Promotable
 - ▶ Validate input
 - ▶ Root
- Not promotable
 - ▶ Mapping file
 - ▶ Associated XSL



This slide shows the Promoted Properties panel and lists those properties that are promotable and those that are not.

Promoting the **Validate input** property allows an administrator to turn validation of the SMO off and on. This allows the performance advantage of not doing validation of the input SMO. However, if the need arises to debug a problem, the administrator can turn on validation while the problem is being investigated.

For the remaining properties, there is inconsistency regarding which properties are promotable and which are not. The **Root** property is promotable. However, changing its value at runtime causes a problem, unless you can also change the XSL style sheet that is used so that it is compatible with the new root specification. However, neither the **Mapping file** nor the **Associated XSL** properties are promotable. Therefore, it is best to never promote the root property.



- New XML mapping dialog
 - ▶ Create a new XML map
 - Namespace
 - Folder
 - Name
 - Create sample XML for testing
 - ▶ Select message types
 - Message root
 - Input and output message body
 - ▶ Currently configured contexts
 - Context change requires new map

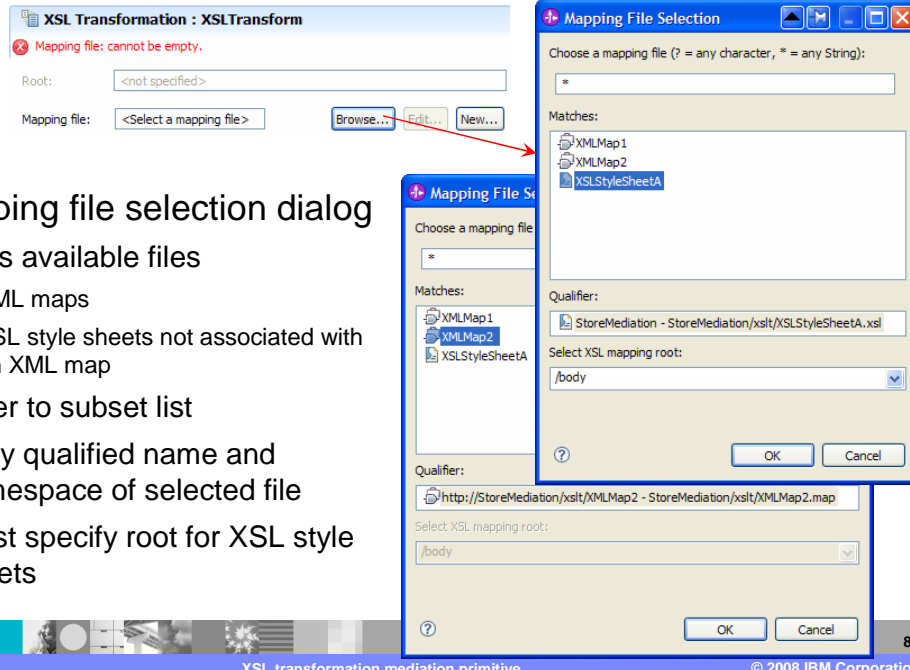
To create a new XML map, the New... button is used to access the new XML mapping dialog, which is shown in the screen captures on this slide. The first panel, seen in the upper right, allows you to specify the namespace, folder and map name, which are all pre-filled with default values. You can change any of these if required to satisfy your naming conventions. There is also a check box that indicates that you want an XML file created to use as input for testing this XML map. The test file created is based on the SMO schema for the portion of the SMO that is input to the transform. More details on using this XML file for testing are provided later in the presentation.

The second panel of the dialog, seen in the lower screen capture, allows you to specify the message root, indicating if the whole SMO, or just the context, headers or body is being transformed. You also specify on this panel the message type of both input and output messages. If the primitive's terminal already has a message type defined, it is pre-filled in the dialog, otherwise the field is left blank. The Browse... button is used to access the change message type dialog, which enables you to define the message type or change an existing message type. Whatever you define here is propagated to the message type definition for the terminal, ensuring that they remain synchronized.

Clicking the Finish button creates an XML map corresponding to the root and message type definitions provided in this dialog. You are placed into the XML map editor so that you can begin to define the transformations between the input and output messages.

The business objects that are configured for the correlation, transient and shared contexts are shown on this panel. It is important to know this if the map is for the entire SMO or for the contexts, as these contribute to the schema use for the map's input and output. After creating the map, changes to the contexts are not refractored into the map. Therefore, you will need to re-create the map if you make changes to the configuration of contexts for the flow.

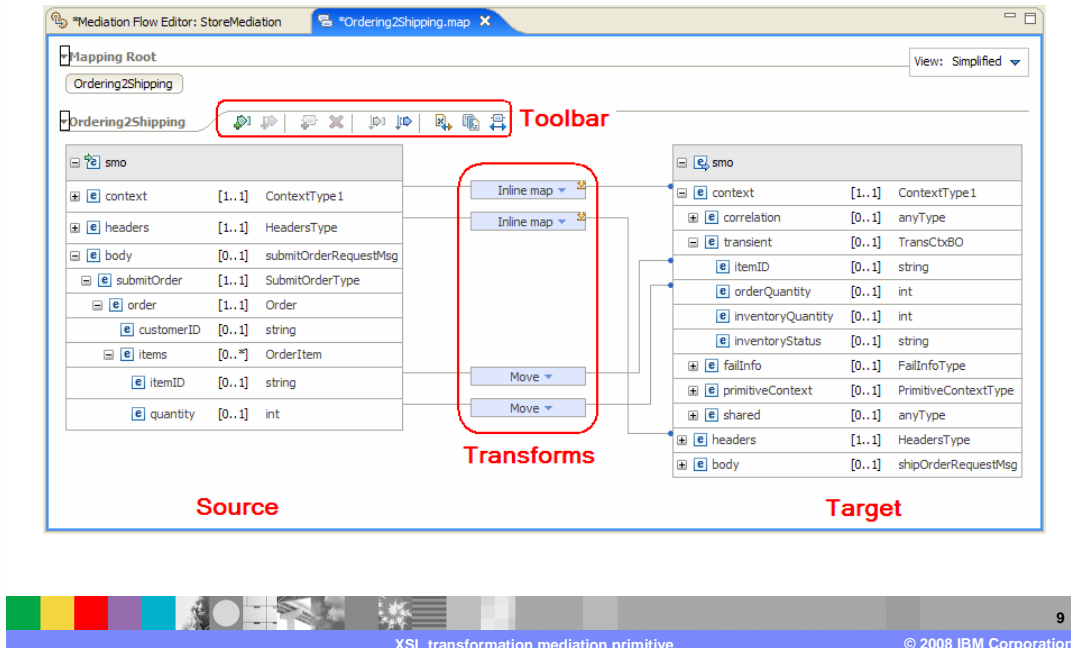
Using an existing mapping file



- Mapping file selection dialog
 - ▶ Lists available files
 - XML maps
 - XSL style sheets not associated with an XML map
 - ▶ Filter to subset list
 - ▶ Fully qualified name and namespace of selected file
 - ▶ Must specify root for XSL style sheets

This slide looks at selecting an existing mapping file, which can be either an XML map or an XSL style sheet. This is done using the mapping file selection dialog that is accessed using the Browse... button. The dialog lists all the maps available in the project and in the libraries used by the project. It also lists the XSL style sheets that are not associated with a map. As with many of the dialogs, a filter string can be used to filter the list, making it easier to find the file you need. When a file is selected in the list, the fully qualified name is shown in the Qualifier field. This can help when there are multiple files within scope that have the same name. When a mapping file is selected, the value for the root property is determined from the map. However, you must specify the root property value when selecting an XSL style sheet, which is done in the dialog using the field labeled Select XSL mapping root. The slide shows the selection of an XSL style sheet in the upper screen capture and the selection of a map in the lower one.

XML map editor



The layout of the XML map editor is shown here. It is very similar to the business object mapping editor, with the source shown on the left and the target on the right. Transforms are created by dragging from a source to a target field and then defining the specifics of the transform in the properties view. There is a toolbar that provides quick access to several functions, such as adding, deleting and sorting transforms, associating XML files for testing and mapping like fields between source and target. There is no capability provided to define the order in which the transforms are performed.

XML map transforms

Transform	Function
move	Copies value between simple types with conversion Copies value between complex elements of the same type
Inline map	Nested map defines transforms between complex types or simple and complex type Not reusable from other maps
submap	Nested map defines transforms between complex types or simple and complex type Reusable from other maps Usable with xsd:anyType (submap defined for actual type expected at runtime)
custom	Applies custom coded XPath or XSLT between source and target Source and target can be either simple or complex types Useable with xsd:anyType
concat	Concatenates multiple source strings and assigns to target string
normalize	Normalizes source string and assigns to target string
substring	Using a delimiter and index, selects a substring from source and assigns to target
assign	Assign a value to a target

10

XSL transformation mediation primitive

© 2008 IBM Corporation

These are the transforms that can be used in an XML map.

The **move** transform is used to copy a value from source to target. When the fields are simple types, conversions are automatically done if required, such as converting an integer source value to a string target. When fields are complex types, they must be of the same type.

The **inline map** transform is a nested map used to map between complex types or between a simple and complex type. The inline map is embedded in the map containing the inline map transform and is therefore not reusable.

The **submap** transform is also a nested map used to map between complex types or between a simple and a complex type. The difference is that the submap is in its own file and is reusable from other maps. Also, the submap can be used to map to or from a field that is defined as an xsd:anyType. The definition of the map must match the actual type contained within the xsd:anyType at runtime.

The **custom** transform allows you to code XPath or XSLT between the source and target, which can be either simple or complex types. Similar to submap, custom can also be used with xsd:anyType fields.

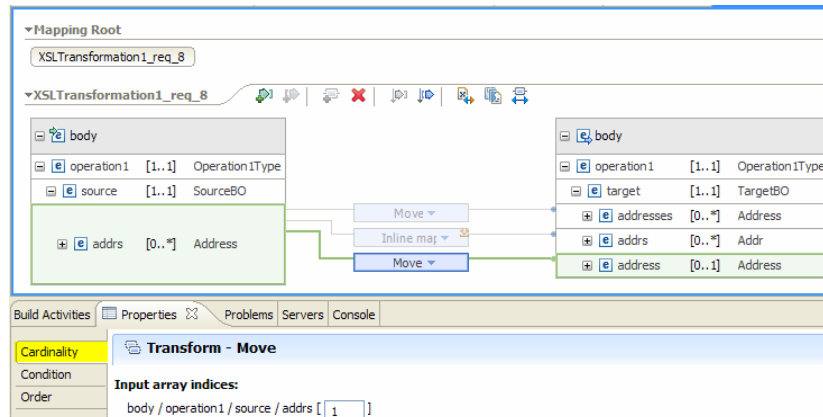
The **concat** transform concatenates multiple source strings and assigns them to a target string.

The **normalize** transform does a normalization of the source string and assigns it to the target string.

The **substring** transform uses a delimiter and an index to select a portion of the source string and assign it to the target.

Finally, the **assign** transform allows you to configure a constant value that is assigned to a target field.

Arrays



- Move – can be used between arrays of like type
- Inline map or submap – used between arrays of like or unlike type
- Specific array elements can be indexed using the cardinality property panel

This slide looks at the considerations for arrays that are contained in the source or the target. When source and target arrays are of like type, a move transform can be used to copy the source array to the target. However, if the arrays are of unlike type, an inline map or a submap needs to be used to define the transformation between them. If you want to address a specific element of an array, this is done by specifying a specific index in the cardinality panel of the properties view. The screen capture shows all three of these possibilities. The source contains an array of type Address, which is the source for all three of the transforms. A move transform is used to copy the array to a target array that is also of type address. Then an inline map is used to transform the source array into a target array of type Addr. Finally, a move transform selects a specific element from the source array and copies it to a target field of type Address. The properties view in the screen capture is for this transform, showing the specification of the cardinality for the source array.

Mapping considerations for context or headers

- Context and headers must be explicitly mapped
 - ▶ For maps with root = “/”, “/context” or “/headers”
 - ▶ Runtime errors might occur if not mapped
- Approach
 - ▶ Define unique transforms needed between source and target
 - ▶ Use match mapping icon to create all other transforms

The screenshot shows the IBM XSL transformation mediation primitive interface. On the left is the source SMO (SubmitOrderRequestMsg) and on the right is the target SMO (shipOrderRequestMsg). A toolbar at the top contains a button labeled "Map source to target based on name and types" which is highlighted with a red box. Below the SMOs, two "Move" buttons are visible, indicating that elements like 'itemID' and 'quantity' have been moved from the source body to the target transient context.

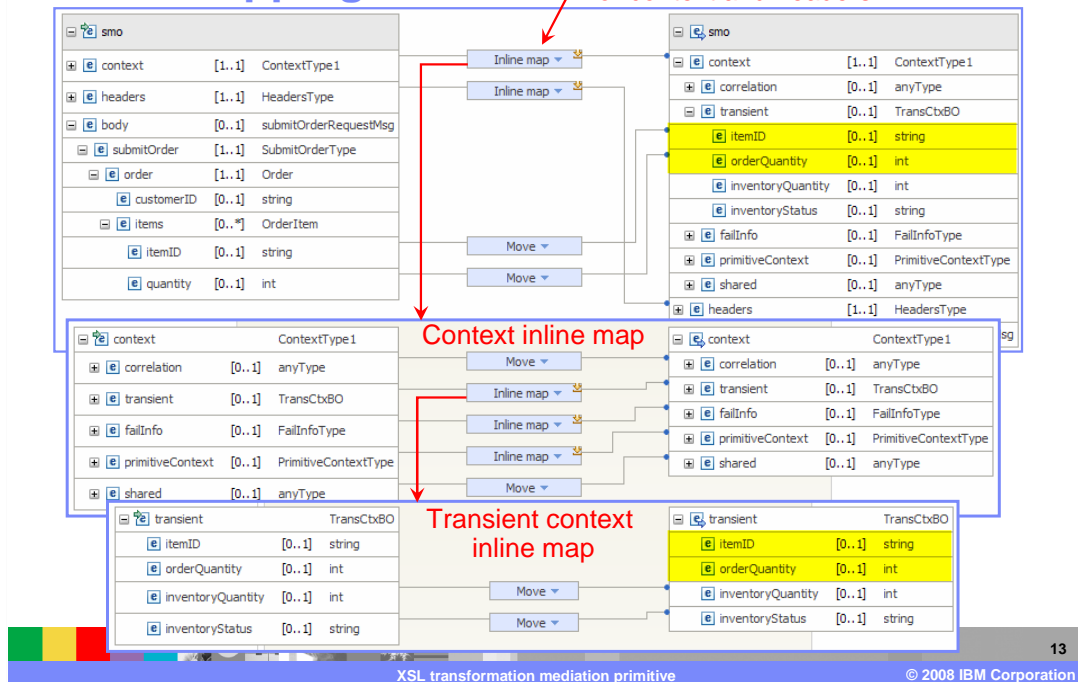
Source SMO Element	Source SMO Type	Target SMO Element	Target SMO Type
context	ContextType1	context	ContextType1
headers	HeadersType	headers	HeadersType
body	submitOrderRequestMsg	body	shipOrderRequestMsg
submitOrder	SubmitOrderType	submitOrder	SubmitOrderType
order	Order	order	Order
customerID	string	customerID	string
items	OrderItem	items	OrderItem
itemID	string	itemID	string
quantity	int	quantity	int
		transient	TransCbBO
		correlation	anyType
		inventoryQuantity	int
		inventoryStatus	string
		fallInfo	FallInfoType
		primitiveContext	PrimitiveContextType
		shared	anyType

When a map is run, it is only those elements that are explicitly mapped that are created in the target XML file. This must be considered for all maps, but is especially important when defining a map that contains the context or headers sections of the SMO. If these are not mapped, it might result in runtime errors downstream in the flow. So that you do not have to explicitly define the transforms for every field in the context and headers, a function that creates transforms between fields of like name and type is provided. You can see the icon for this in the toolbar of the screen capture.

The correct approach to using this is to first create all the transforms between fields of unlike name and type that satisfy the requirements of your map. In this example, an itemID and quantity field are moved from the body of the source to the transient context of the target. The button labeled Map source to target based on name and types can then be used to automatically create the remaining required transforms to copy the context and headers to the target. The results of this are shown on the next slide.

Match mapping result

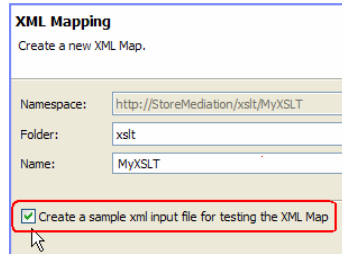
Inline map transforms created for context and headers



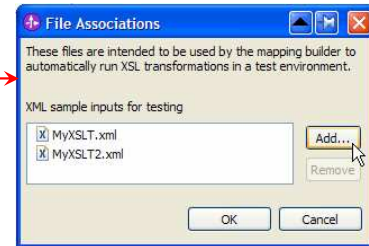
In the upper screen capture is the same XML map as is shown on the previous slide, but after the mapping of like fields has been done. Notice that there are now two inline map transforms, one for the context and the other for the headers. Opening the inline map for the context shows the map in the middle screen capture. Notice that it is composed of inline maps for complex types and move transforms for simple types. Opening the inline map for the transient context results in seeing the map shown in the bottom screen capture. Notice that the `inventoryQuantity` and `inventoryStatus` fields have move transforms, but that there are no transforms for the `itemID` and `orderQuantity` fields. This is because transforms for those target fields were already created before the mapping of like fields took place.

Testing XML maps

- XML files can be associated to maps for testing
 - ▶ Testing of the map occurs when the map is saved
 - ▶ Multiple test files can be associated with a map
- Two ways of associating the input XML files
 - ▶ Generated automatically at map creation time (named <mapname>.xml)



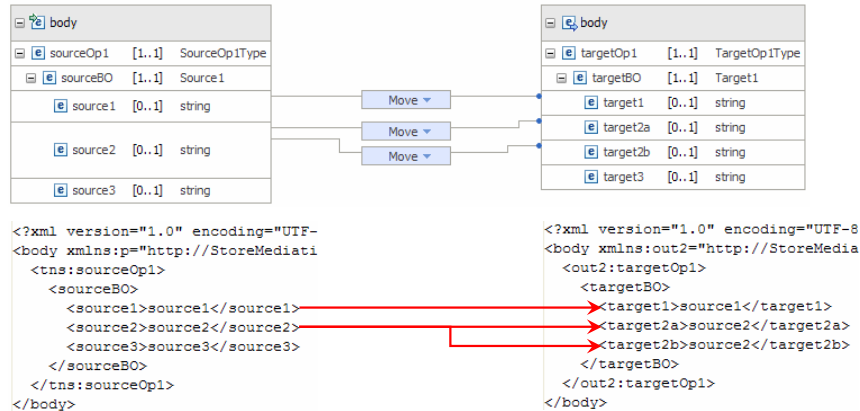
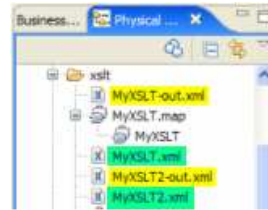
- ▶ Existing XML files associated with existing maps



WebSphere Integration Developer provides an automatic testing capability. It takes an XML file associated with the map, uses it as input to the map whenever the map file is saved, and creates the resulting output XML file. There can be multiple of these input XML test files configured for a map. There are two ways to associate an input XML file with a map. The first is to have it automatically generated when creating the map. This is done by selecting a check box in the new XML map dialog as is shown in the screen capture. The file will have the same name as the map, but with a .xml file extension rather than a .map extension. The other approach is to associate an existing XML file with a map. There is an associate XML files button in the toolbar for a map, which opens the file associations dialog, as is shown in the bottom screen captures.

Testing XML maps

- Output XML files
 - ▶ <inputXMLname>-out.xml
 - ▶ Stored in same directory as the map
- Example map and XML files



15

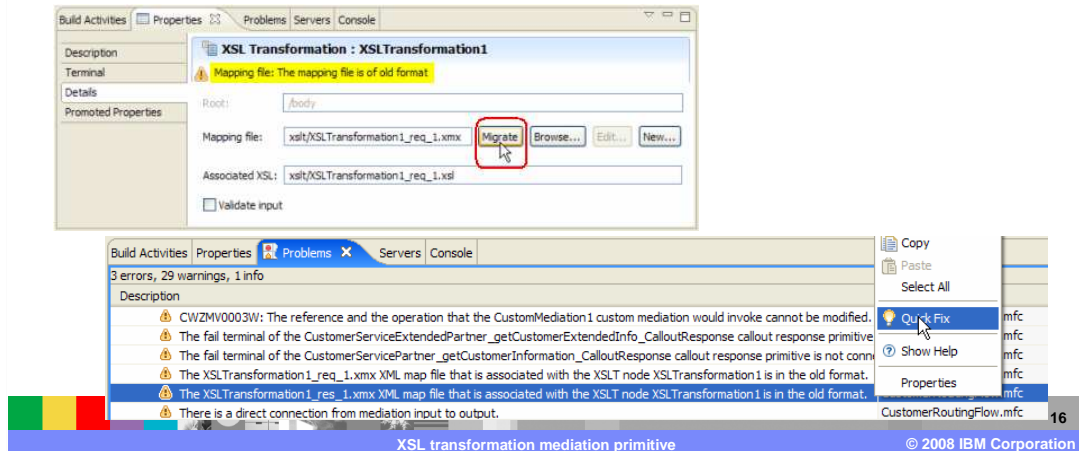
XSL transformation mediation primitive

© 2008 IBM Corporation

When a map file is saved, all of the associated XML files are used as input to the map, producing the output files. The output files are named with the same name as the input files, with dash out (-out) appended. Examples of these can be seen in the screen capture in the upper right. The two input files, MyXSLT.xml and MyXSLT2.xml resulted in the two output files MyXSLT-out.xml and MyXSLT2-out.xml. The screen capture in the middle of the slide shows an XML map, with source1 moved to target1, and source2 moved to target2a and target 2b. In the bottom screen captures you can see the input and output XML files, reflecting the results of the test

Migration from earlier releases

- The XML map editor is new in V6.1
 - ▶ Based on Rational® Application Developer V7 XML map editor
- Considerations for XSL transformation primitives from earlier releases:
 - ▶ No change → XSL style sheets will run as is
 - ▶ Changing → Old format .xmx maps cannot be edited, must be migrated



In version 6.1 a new XML map editor was introduced in WebSphere Integration Developer, replacing the editor that was used in earlier versions. The editor is based on the Rational Application Developer V7 XML map editor. If you have projects from before version 6.1 that contain XSL transformation primitives, you might need to migrate the maps. In the case where you do not need to change the map, the XSL style sheet from the earlier release will work as is. However, if you need to make changes to the map, it must be migrated from the old format to the new. The old format maps can be recognized by the .xmx file extension. Migrating an old format map is quite easy, with two different approaches being provided. The first is shown in the upper screen capture, which shows the properties view of an XSL transformation primitive in a project that was imported from version 6.0.2. Notice that there is a warning that the mapping file is the old format, there is a Migrate button for the mapping file, and the Edit... button is disabled. Click the Migrate button to convert the .xmx map to a version 6.1 .map file. The warning and Migrate button are removed and the Edit... button is enabled. The screen capture at the bottom of the slide is of the Problems view, showing a warning message about the XSL transformation primitive being configured with an old format map. Using the pop-up menu and selecting Quick Fix opens a series of dialogs that allows you to migrate this .xmx map. You can also select others in the project to migrate at the same time.

Error processing

- **MediationRuntimeException** thrown for:
 - ▶ XSL style sheet not found or not specified
- **MediationBusinessException** (fail terminal flow)
 - ▶ Validate input specified and input message fails validation testing
 - ▶ Output message fails validation testing
 - The message resulting from the transformation is always validated
- **XSLTMediationException** (fail terminal flow)
 - ▶ Errors during XSL style sheet processing



The error processing details and considerations are examined in this slide, describing some of the error situations you might encounter.

A **MediationRuntimeException** is thrown for problems accessing the XSL style sheet, such as when the style sheet cannot be found, has not been specified or for some other reason cannot be loaded by the mediation runtime.

A **MediationBusinessException** occurs if validate input has been specified and the input message fails the validation processing. Output messages are always validated and the **MediationBusinessException** occurs when the output message fails validation testing. For these and other **MediationBusinessException** cases, if the fail terminal is wired, the flow from the fail terminal is followed rather than the exception being raised.

There is a new exception being used in version 6.1, the **XSLTMediationException**. It occurs if there are errors encountered during the processing of the XSL style sheet, such as a parsing error processing the input XML. Similar to the **MediationBusinessException**, this exception results in the fail terminal flow if the fail terminal is wired.

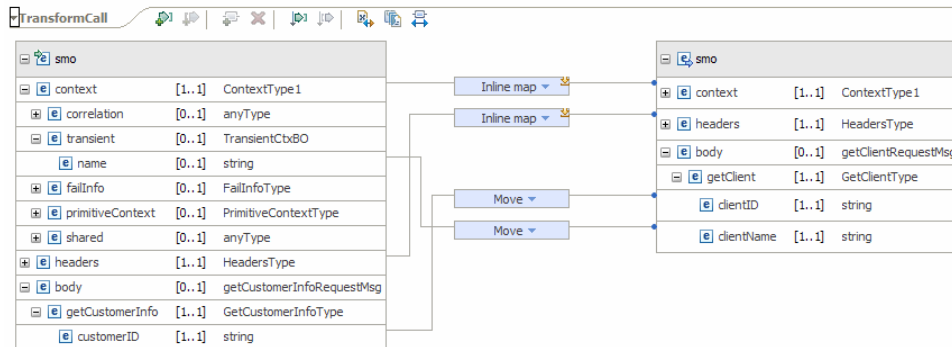
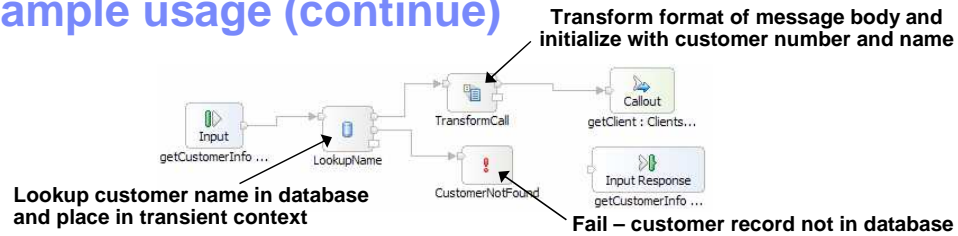
Example usage

- Example – Setup call for target service
 - ▶ Mediation flow input operation contains only customer number
 - ▶ Operation on target service callout requires customer name and number
- Mediation logic:
 - ▶ Database lookup
 - Using customer account number looks up customer name from database
 - Customer name placed into the transient context
 - Failure if no record found for customer
 - ▶ XSL transformation
 - Maps source message format to target message format
 - Copies the account number from source message body to target message body
 - Copies the name from the source transient context to the target message body



The next couple of slides provide an example usage of the XSL transformation primitive. In this scenario, a call to a service provider requires more information than the call coming in from the service requestor. Specifically, the request contains only a customer number, and the service provider has an interface requiring both a customer name and customer number. This is done using an XSL transformation primitive in conjunction with a database lookup primitive. The mediation flow logic starts with the database lookup, which uses the customer number to look up a customer name from a database. The customer name is placed into the transient context of the SMO by the database lookup. Since the service provider always needs both the customer name and number it is considered an error condition if the lookup does not find the customer record. Following the database lookup, the XSL transformation primitive maps the source message to the target message, changing the message type to match the operation being called on the service provider. The account number is moved from the source SMO body to the target SMO body and the name is moved from the source SMO transient context to the body of the target SMO.

Example usage (continue)



19

XSL transformation mediation primitive

© 2008 IBM Corporation

The top portion of this slide shows the mediation flow logic for the example scenario. The input node is wired to a database lookup, which looks up the customer name using the customer number as the key. The keyNotFound terminal is wired to a fail primitive, which raises an exception. If the lookup is successful, the flow goes to an XSL transformation primitive, which modifies the SMO to the format required by the service provider. The bottom portion of the slide shows the XML mapping editor. You can see the move transform for the customerID in the source body to the clientID in the target body. The other move transform shown here is for the name field in the source transient context to the clientName in the target body. Also notice the inline maps that move the context and headers from the source to the target.

Summary

- Examined the XSL transformation primitive



XSL transformation

- ▶ Overview of function
- ▶ Use of terminals
- ▶ Definition of properties
- ▶ Maps – creating, editing, defining transforms and testing
- ▶ Migration from previous releases
- ▶ Error handling
- ▶ Example usage

In summary, this presentation provided an overview of the XSL transformation primitive along with information about the primitive's use of terminals and its properties. The presentation looked at the XML maps to see how they are created and edited, the defining of transformations within the maps and how the maps are tested. The migration capabilities from version 6.0.2 to 6.1 were discussed. Finally, error handling characteristics were presented followed by an example usage of an XSL transformation 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_WBIV61_XSLTransformationPrimitive.ppt

This module is also available in PDF format at: [../WBIV61_XSLTransformationPrimitive.pdf](..WBIV61_XSLTransformationPrimitive.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 Rational WebSphere

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

Rational is a trademark of International Business Machines Corporation and Rational Software Corporation 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 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.

© Copyright International Business Machines Corporation 2008. 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.

