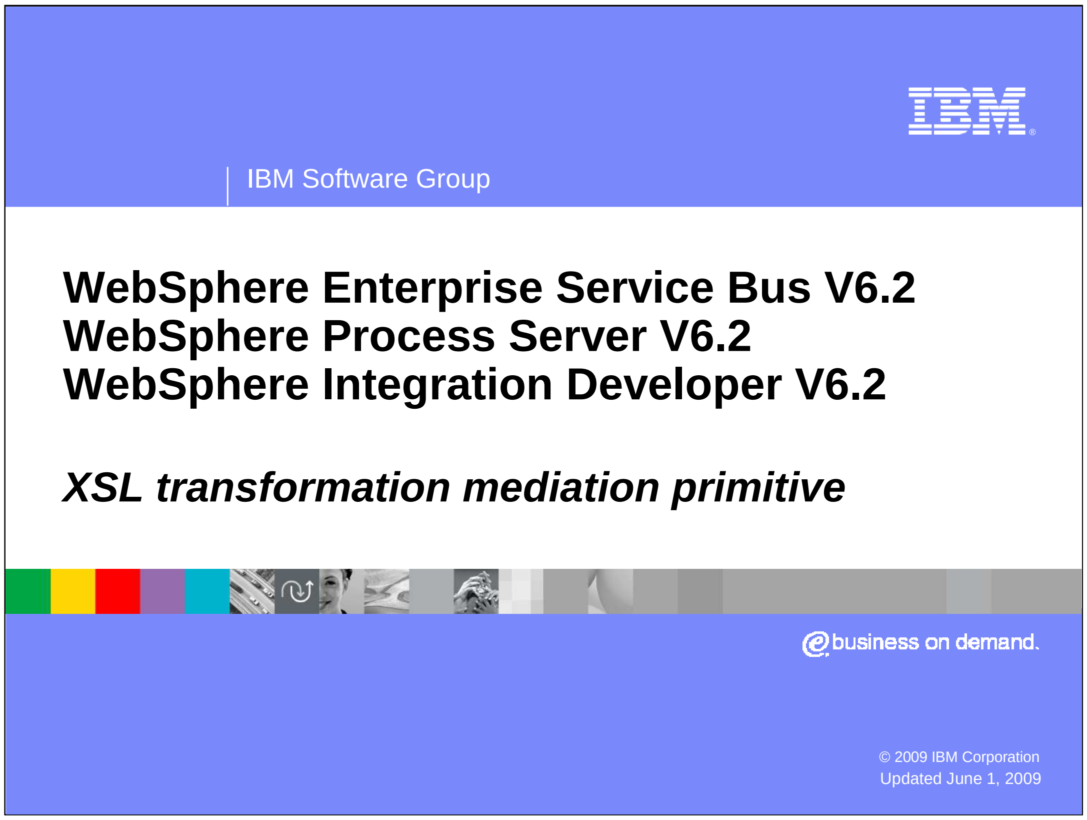


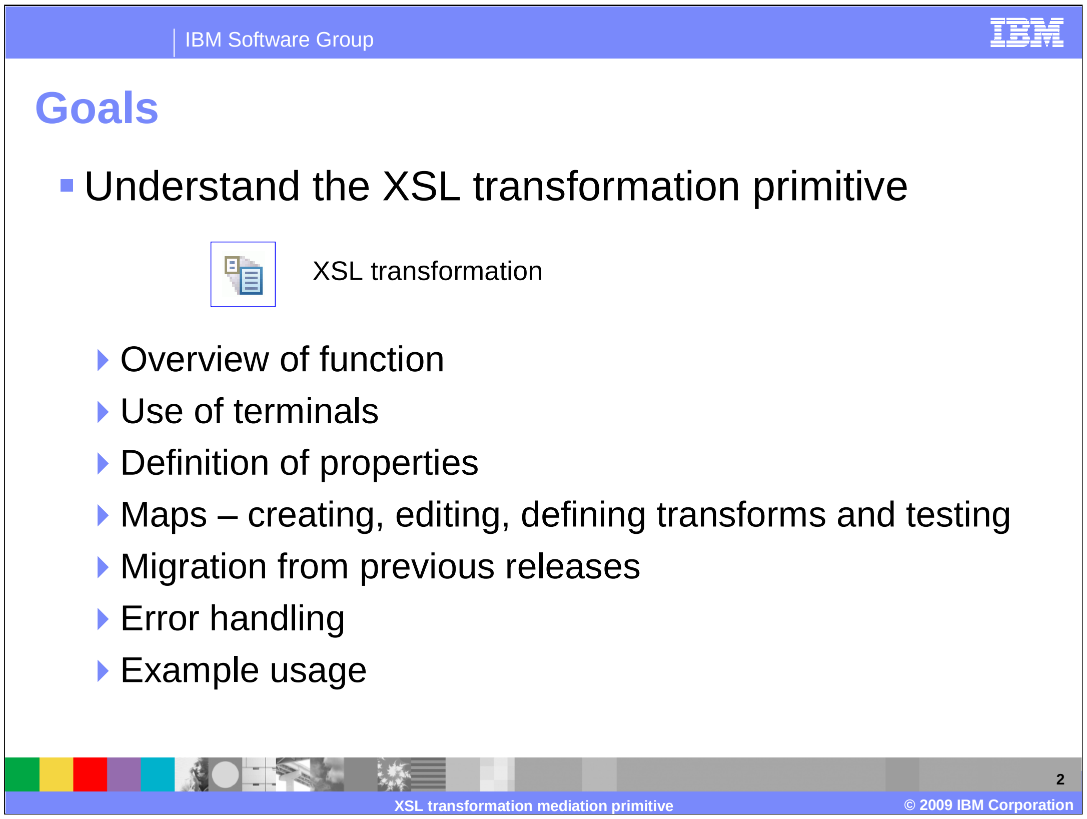
IBM Software Group

WebSphere Enterprise Service Bus V6.2 WebSphere Process Server V6.2 WebSphere Integration Developer V6.2

XSL transformation mediation primitive

business on demand.

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

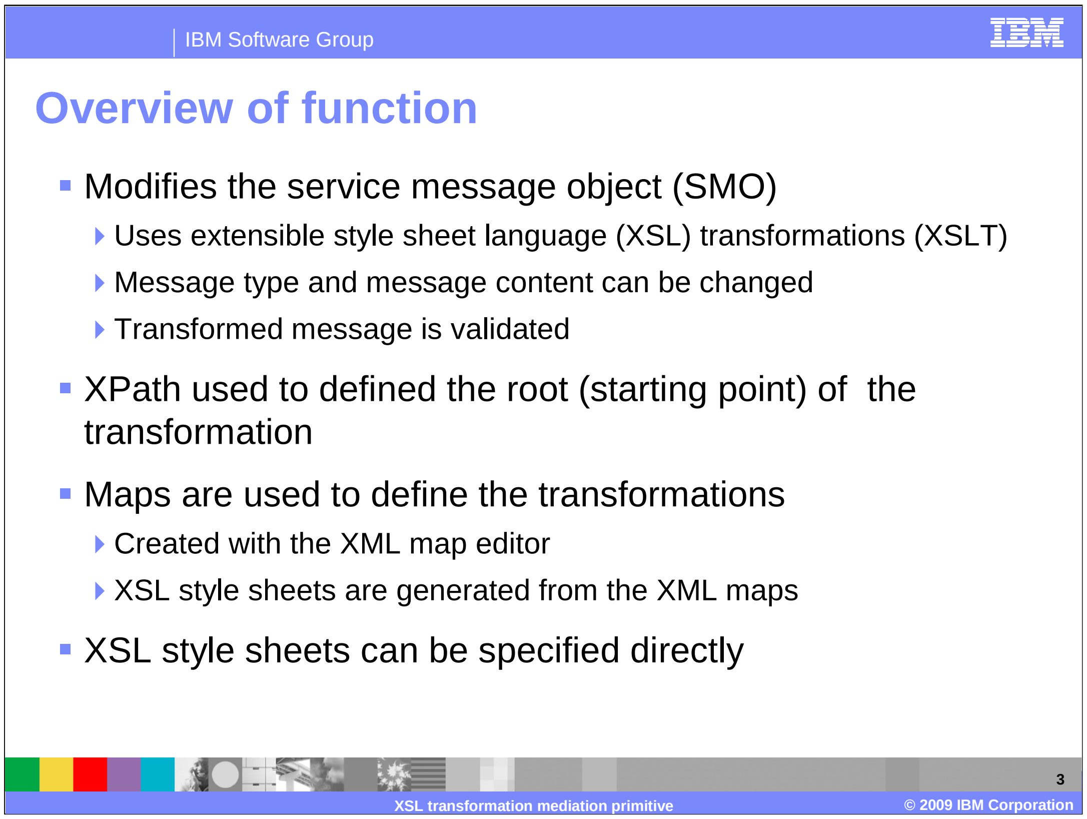
  XSL transformation

  - 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 presentations that cover common elements of all mediation primitives, such as properties, terminals, wiring and the use of promoted properties. The general knowledge of mediation primitives they provide is needed to understand the XSL transformation primitive specific material in this presentation.

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

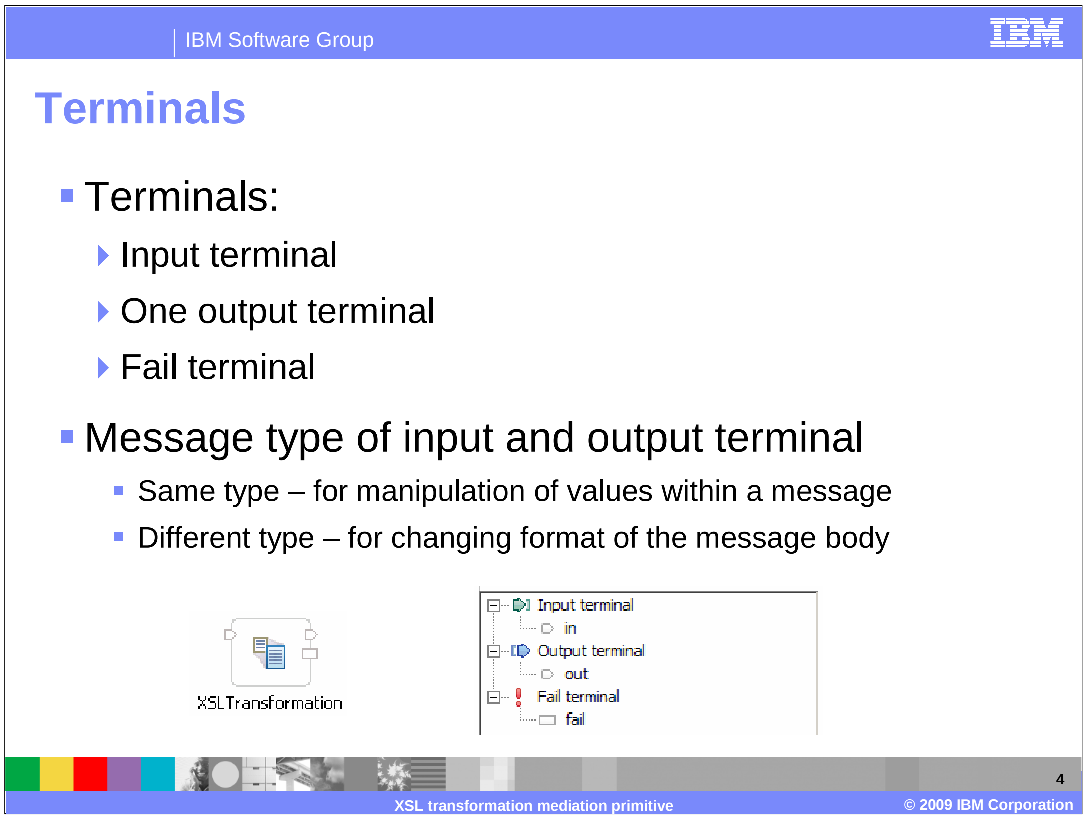
- Modifies the service message object (SMO)
  - Uses extensible style sheet language (XSL) transformations (XSLT)
  - Message type and message content can be changed
  - Transformed message is validated
- XPath used to defined the root (starting point) of the transformation
- Maps are used to define 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.

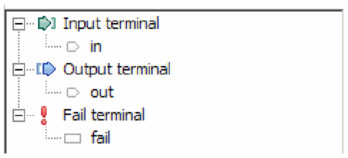
If you have an existing XSL style sheet that meets the requirements of your transformation, you can specify that XSL style sheet directly rather than defining the transformation with an XML map.



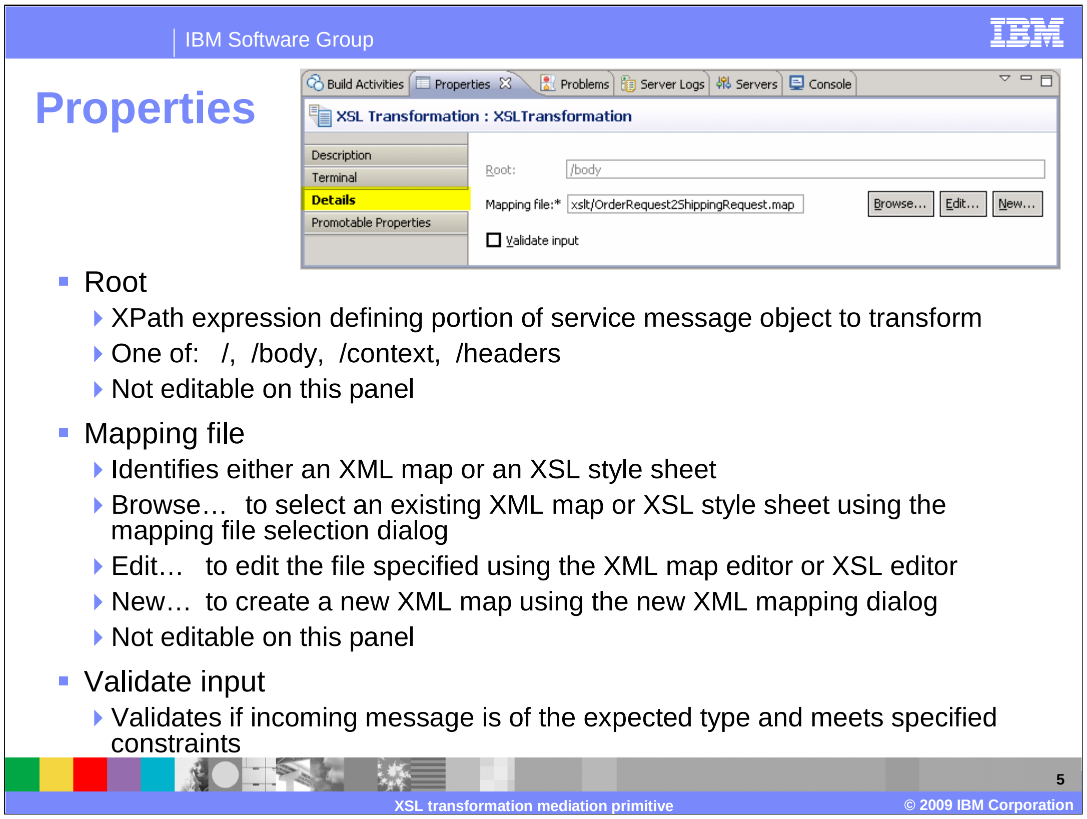
Terminals

- 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

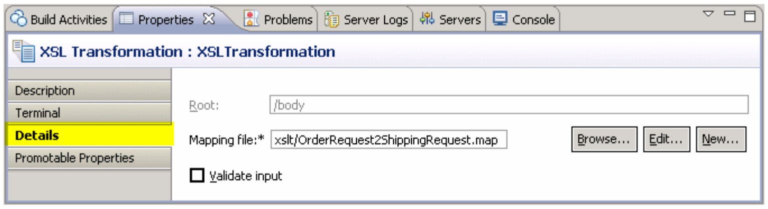
XSLTransformation



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



XSL Transformation : XSLTransformation

Root: /body

Mapping file:* xslt/OrderRequest2ShippingRequest.map

Validate input

- 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
- Validate input
  - Validates if incoming message is of the expected type and meets specified constraints

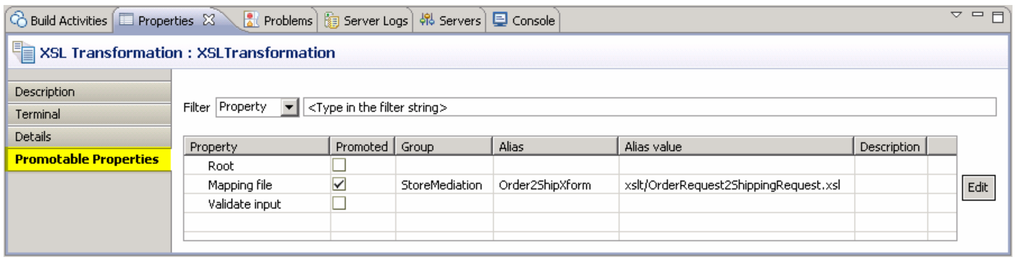
In the upper right is a screen capture of the Details panel from the Properties view for an XSL transformation primitive.

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. Only XML maps can be newly created directly from the primitive, whereas an XSL style sheet must already exist to be configured with the primitive.

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.

Promotable properties



Build Activities | Properties | Problems | Server Logs | Servers | Console

XSL Transformation : XSLTransformation

Description
Terminal
Details
Promotable Properties

Filter Property <Type in the filter string>

Property	Promoted	Group	Alias	Alias value	Description
Root	☐				
Mapping file	✓	StoreMediation	Order2ShipXform	xslt/OrderRequest2ShippingRequest.xsl	
Validate input	☐				

Edit

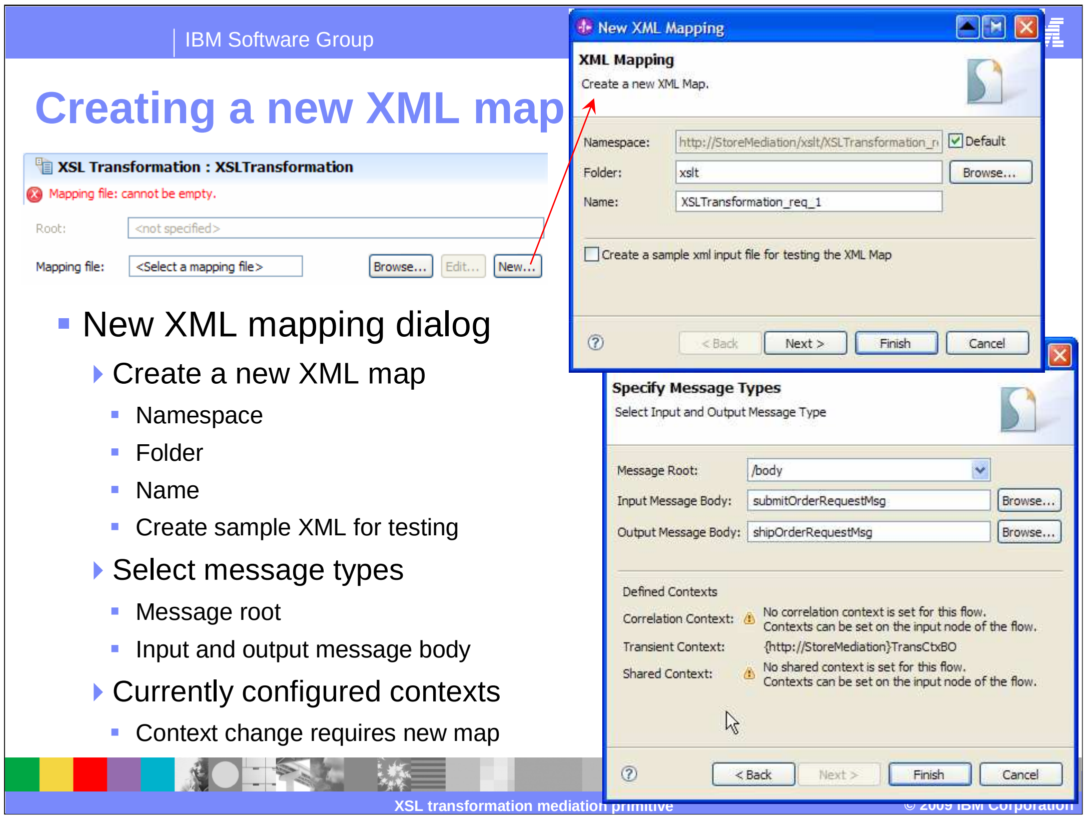
- Promotable
  - Root
  - Mapping file
  - Validate input

This slide shows the Promotable Properties panel and lists those properties that are promotable and those that are not. As you can see, all three of the XSL transformation primitive’s properties are promotable.

The Root property is promotable, however changing its value at runtime can be problematic. The value of the root property is directly associated with the XSL style sheet, so unless you also change the XSL style sheet so that it is compatible with the new root specification, runtime errors will occur.

The mapping file property is promotable. The alias value for this property is an XSL style sheet, even though the value for the property is an XML map. The reason is that the XSL style sheet generated from the XML map is what is processed at runtime, and promoted properties are used to affect the runtime behavior. If the alias value is promoted at runtime to an XSL style sheet based on a different root value, the value for the root property should also be promoted to the compatible value.

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.



IBM Software Group

Creating a new XML map

XSL Transformation : XSLTransformation

Mapping file: cannot be empty.

Root: <not specified>

Mapping file: <Select a mapping file>

- 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

New XML Mapping

XML Mapping

Create a new XML Map.

Namespace: http://StoreMediation/xslt/XSLTransformation_r ✓ Default

Folder: xslt

Name: XSLTransformation_req_1

☐ Create a sample xml input file for testing the XML Map

Specify Message Types

Select Input and Output Message Type

Message Root: /body

Input Message Body: submitOrderRequestMsg

Output Message Body: shipOrderRequestMsg

Defined Contexts

Correlation Context: No correlation context is set for this flow. Contexts can be set on the input node of the flow.

Transient Context: {http://StoreMediation}TransCtxBO

Shared Context: No shared context is set for this flow. Contexts can be set on the input node of the flow.

XSL transformation mediation primitive

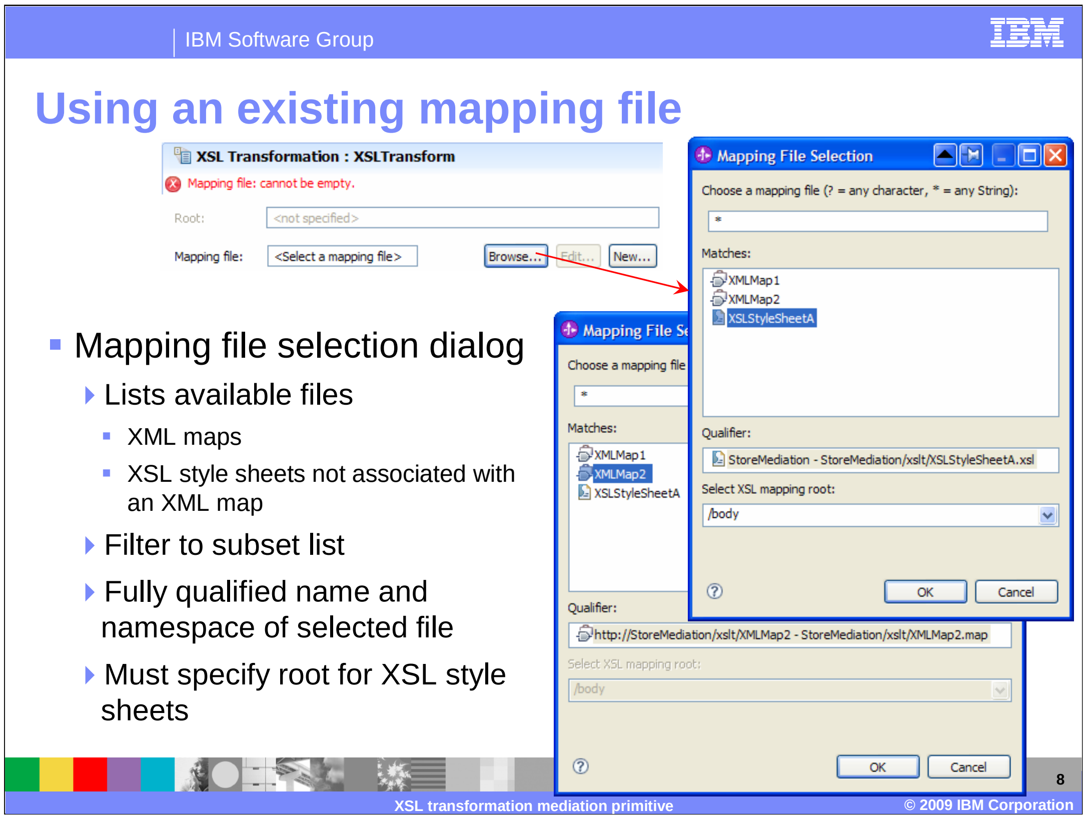
© 2009 IBM Corporation

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.

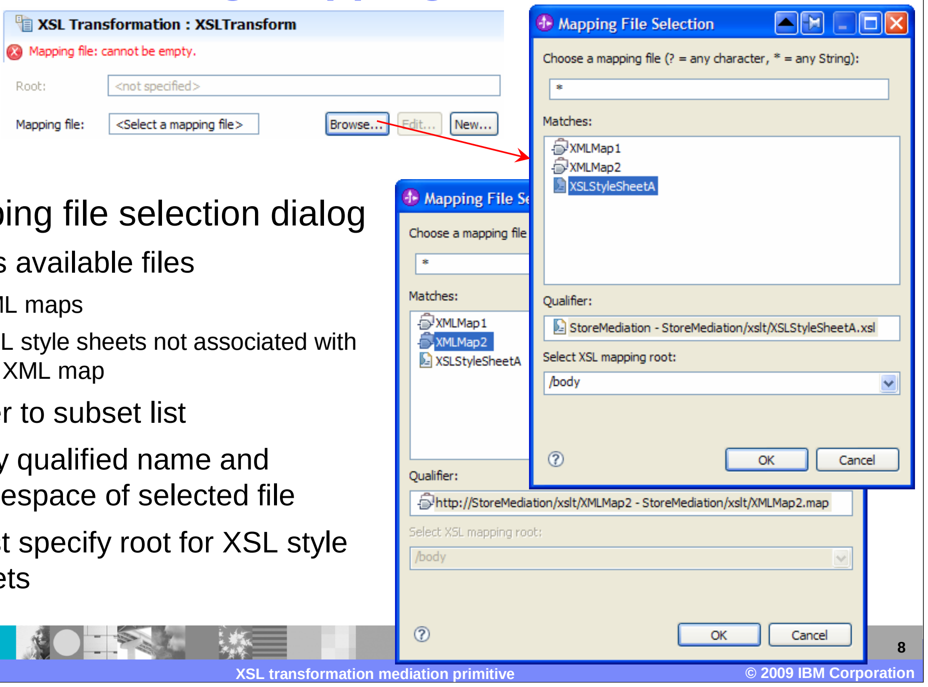
The second panel of the dialog, seen in the lower screen capture, allows you to specify the message root, indicating if the whole SMO, or just the context, headers or body is being transformed. You also specify on this panel the message type of both input and output messages. If the primitive's terminal 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



XSL Transformation : XSLTransform

Mapping file: cannot be empty.

Root: <not specified>

Mapping file: <Select a mapping file>

Mapping File Selection

Choose a mapping file (? = any character, * = any String):

*

Matches:

XMLMap1
XMLMap2
XSLStyleSheetA

Qualifier: StoreMediation - StoreMediation/xslt/XSLStyleSheetA.xsl

Select XSL mapping root: /body

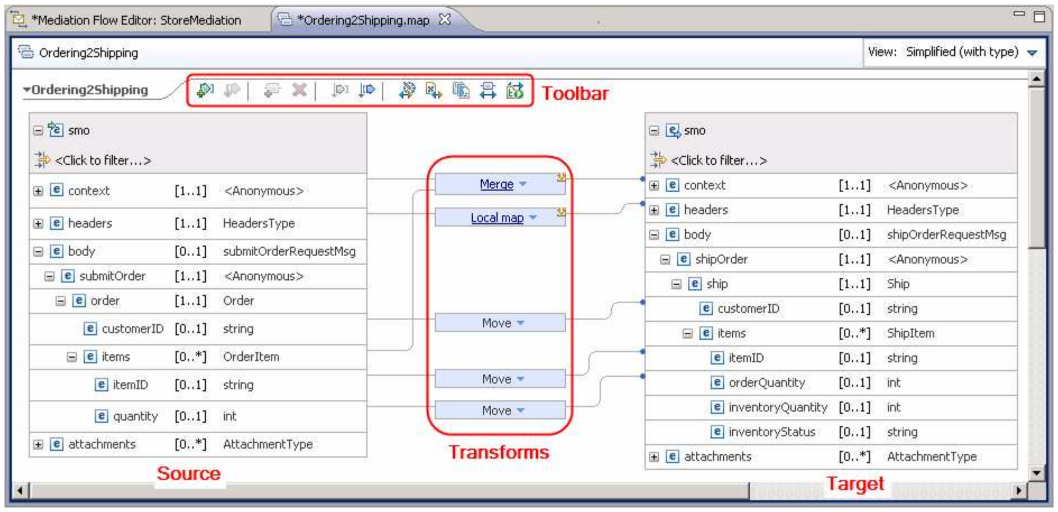
Qualifier: http://StoreMediation/xslt/XMLMap2 - StoreMediation/xslt/XMLMap2.map

Select XSL mapping root: /body

- 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



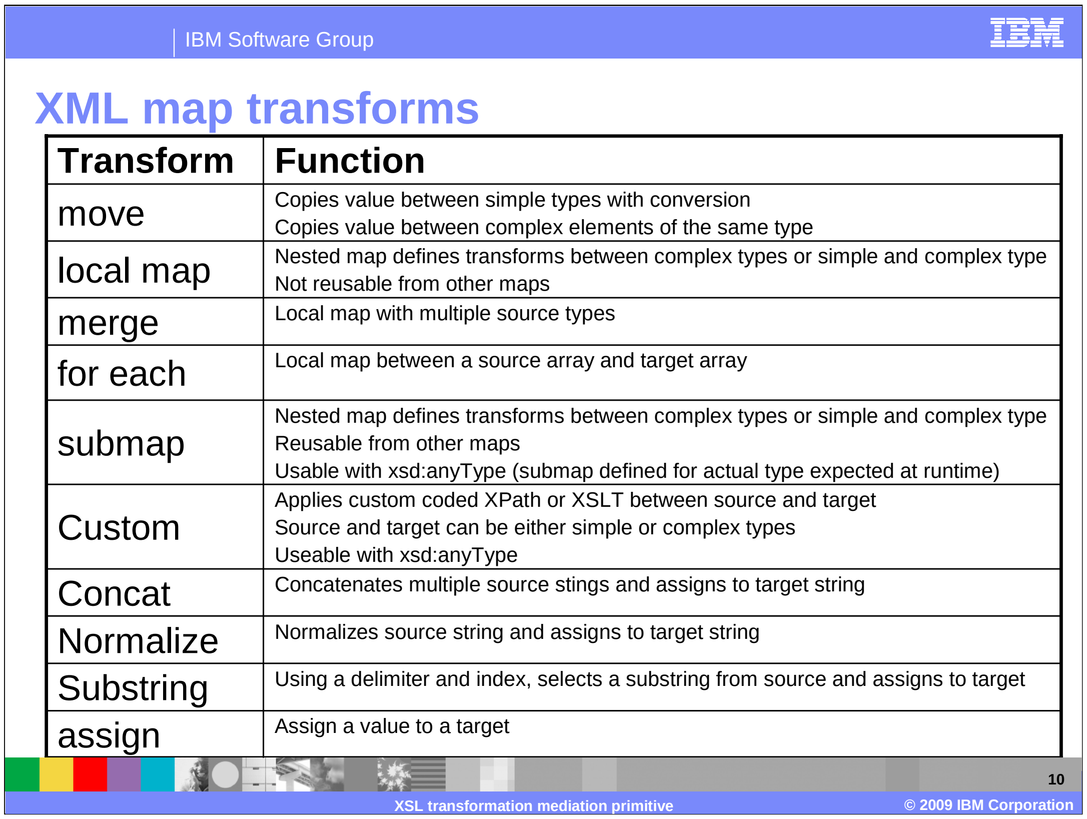
Source:

smo		
<Click to filter…>		
context	[1..1]	<Anonymous>
headers	[1..1]	HeadersType
body	[0..1]	submitOrderRequestMsg
submitOrder	[1..1]	<Anonymous>
order	[1..1]	Order
customerID	[0..1]	string
items	[0..*]	OrderItem
itemID	[0..1]	string
quantity	[0..1]	int
attachments	[0..*]	AttachmentType

Target:

smo		
<Click to filter…>		
context	[1..1]	<Anonymous>
headers	[1..1]	HeadersType
body	[0..1]	shipOrderRequestMsg
shipOrder	[1..1]	<Anonymous>
ship	[1..1]	Ship
customerID	[0..1]	string
items	[0..*]	ShipItem
itemID	[0..1]	string
orderQuantity	[0..1]	int
inventoryQuantity	[0..1]	int
inventoryStatus	[0..1]	string
attachments	[0..*]	AttachmentType

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.


IBM Software Group

XML map transforms

Transform	Function
move	Copies value between simple types with conversion Copies value between complex elements of the same type
local map	Nested map defines transforms between complex types or simple and complex type Not reusable from other maps
merge	Local map with multiple source types
for each	Local map between a source array and target array
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 stings 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 © 2009 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 local map transform is a nested map used to map between complex types or between a simple and complex type. The local map is embedded in the map containing the local map transform and is therefore not reusable.

The merge transform is the same as a local map transform, except that it has more than one source type.

The for each transform is also the same as a local map transform, but is used to map from a source array to a target array.

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.

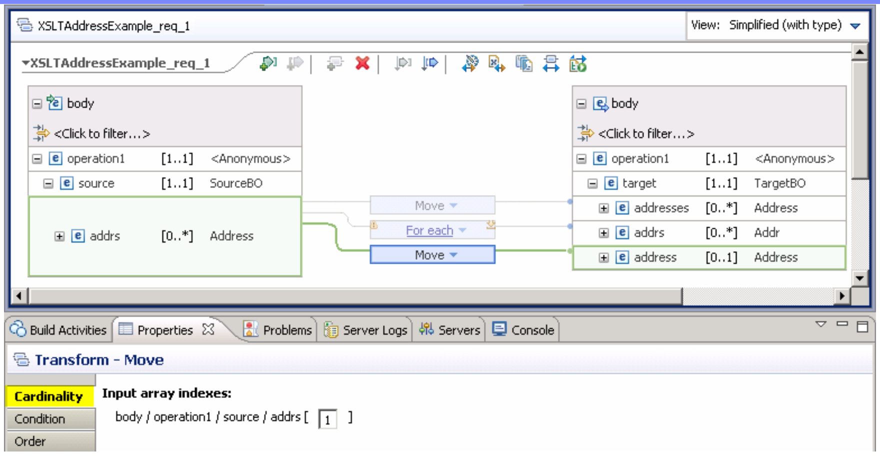
The concat transform concatenates multiple source strings and assigns them to a target string.

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

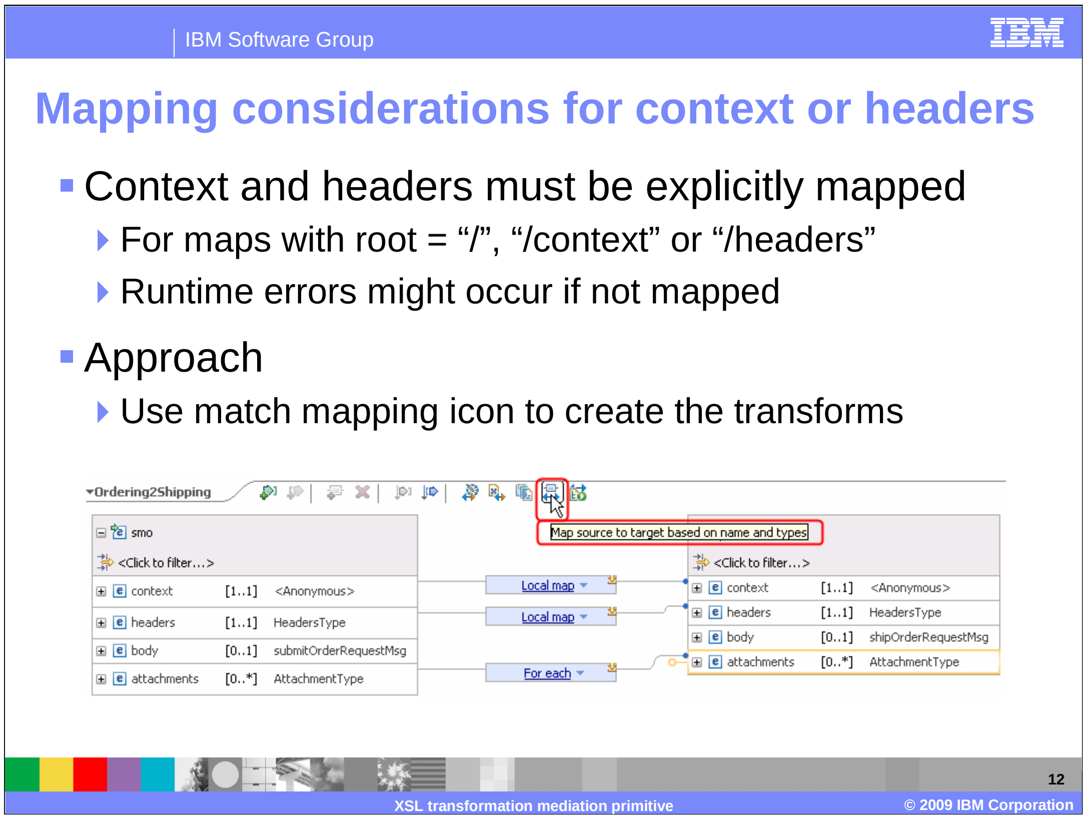


Input array indexes:

body / operation1 / source / addrs [1]

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

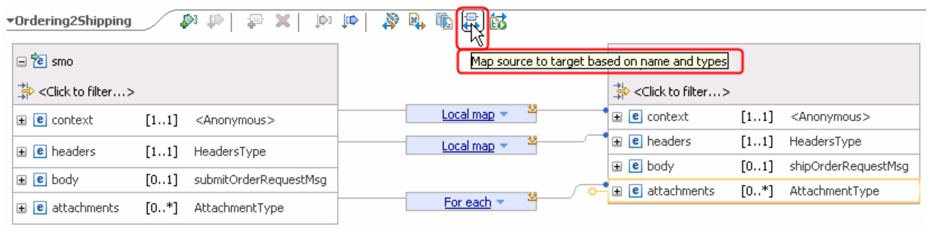
This slide looks at the considerations for arrays that are contained in the source or the target. When source and target arrays are of like type, a move transform can be used to copy the source array to the target. However, if the arrays are of unlike type, a for each or a submap needs to be used to 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 a for each 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.



IBM Software Group

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
  - Use match mapping icon to create the transforms



context	[1..1]	<Anonymous>
headers	[1..1]	HeadersType
body	[0..1]	submitOrderRequestMsg
attachments	[0..*]	AttachmentType

context	[1..1]	<Anonymous>
headers	[1..1]	HeadersType
body	[0..1]	shipOrderRequestMsg
attachments	[0..*]	AttachmentType

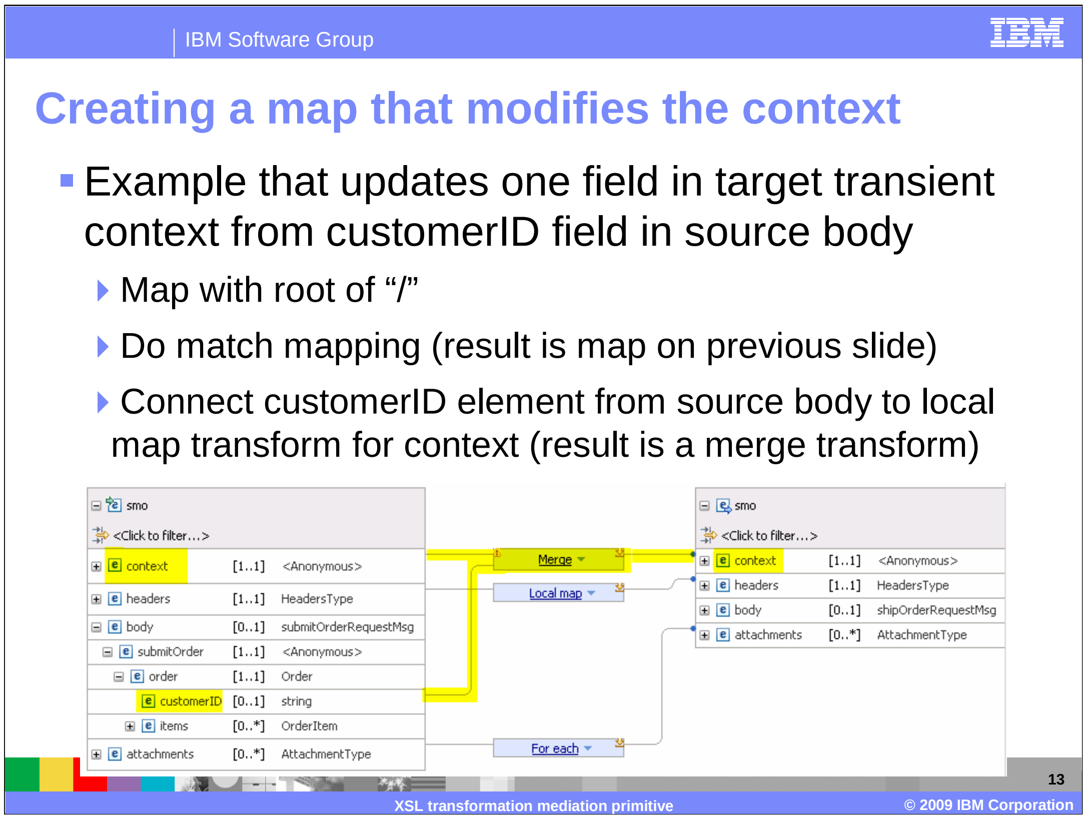
12

XSL transformation mediation primitive © 2009 IBM Corporation

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 tool is to first start your map by using it. The result of this is shown in the screen capture. For both the context and headers, it creates a local map. Inside the local maps it creates nested local maps for complex types and moves for simple types. This nesting recursively occurs until all the contained complex types have been expanded to their simple types and have been mapped. When arrays needs to be mapped, a for each transform is created, as can be seen for attachments in the screen capture. The for each is actually a local map that is applied to each element of the array.

If all you want to do is map some source body elements to the target body, and have the context and headers unchanged, this approach creates what you need for the context and headers. In fact, in this case you can actually use moves rather than local maps for the context and headers. However, the best approach is for you to create a map with a root of /body and then you do not need to consider the context and headers. Considering this, it is likely that when you have a root of slash (“/”), you want to explicitly map something to one or more fields in the context or headers. The next few slides examine the correct approach to developing a map for this case.



IBM Software Group

Creating a map that modifies the context

- Example that updates one field in target transient context from customerID field in source body
  - Map with root of “/”
  - Do match mapping (result is map on previous slide)
  - Connect customerID element from source body to local map transform for context (result is a merge transform)

smo		
<Click to filter…>		
context	[1..1]	<Anonymous>
headers	[1..1]	HeadersType
body	[0..1]	submitOrderRequestMsg
submitOrder	[1..1]	<Anonymous>
order	[1..1]	Order
customerID	[0..1]	string
items	[0..*]	OrderItem
attachments	[0..*]	AttachmentType

Merge

Local map

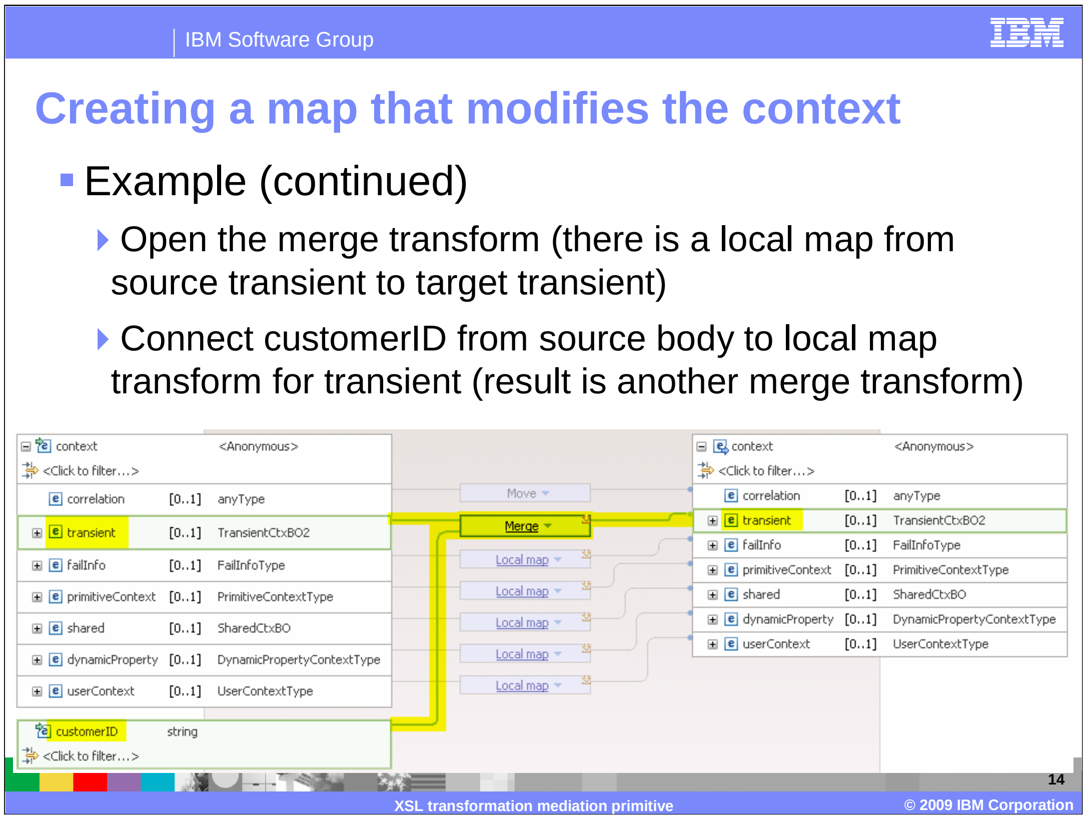
For each

smo		
<Click to filter…>		
context	[1..1]	<Anonymous>
headers	[1..1]	HeadersType
body	[0..1]	shipOrderRequestMsg
attachments	[0..*]	AttachmentType

13

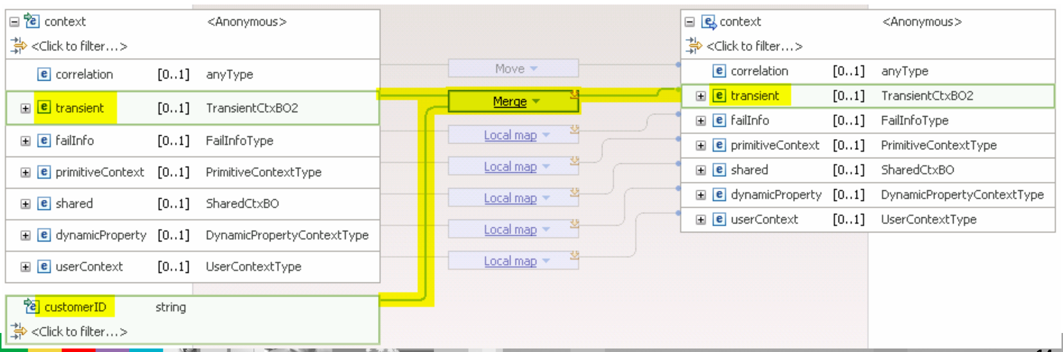
XSL transformation mediation primitive © 2009 IBM Corporation

In these slides there is an example of how to create a map that copies all like fields for the context and headers, except for one field in the transient context. The field is named customer, and rather than setting it from the target transient context, it is to be set from the customerID field in the body of the message. To do this, start by creating a new map with a root of slash (“/”). Next use the match mapping tool to create a map with like fields mapped to each other. The result of doing that is the map shown on the previous slide. Since it is the customerID field from the body that needs to be set in the transient context, add a connection from customerID to the local map transform for context. When you do this, the local map transform is changed to a merge transform. The result of this is shown in the screen capture on this slide.



Creating a map that modifies the context

- Example (continued)
  - Open the merge transform (there is a local map from source transient to target transient)
  - Connect customerID from source body to local map transform for transient (result is another merge transform)



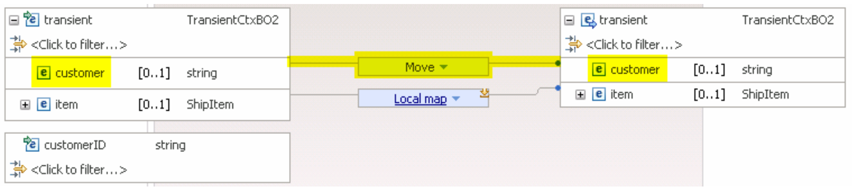
context		<Anonymous>
<Click to filter…>		
correlation	[0..1]	anyType
transient	[0..1]	TransientCtxBO2
failInfo	[0..1]	FailInfoType
primitiveContext	[0..1]	PrimitiveContextType
shared	[0..1]	SharedCtxBO
dynamicProperty	[0..1]	DynamicPropertyContextType
userContext	[0..1]	UserContextType
customerID		string
<Click to filter…>		

context		<Anonymous>
<Click to filter…>		
correlation	[0..1]	anyType
transient	[0..1]	TransientCtxBO2
failInfo	[0..1]	FailInfoType
primitiveContext	[0..1]	PrimitiveContextType
shared	[0..1]	SharedCtxBO
dynamicProperty	[0..1]	DynamicPropertyContextType
userContext	[0..1]	UserContextType

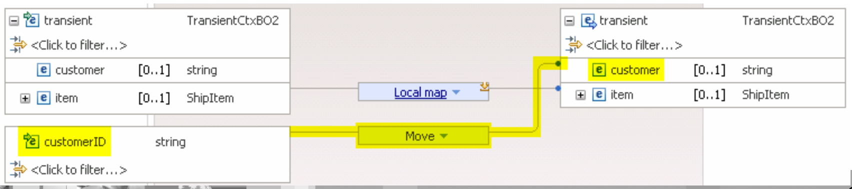
Continuing with the example, the next thing to do is open the merge transform. It is a local map with a source consisting of all of the context plus the customerID from the body and all the context as the target. Each simple type in the context is mapped with a move transform, such as is seen for correlation. Each complex type is mapped with a local map, as is seen for several of the others, such as failInfo and shared. The next thing to do is to connect the customerID to the local map transform for transient, which results in it becoming a merge transform. The screen capture on this slide shows the map after these steps have been taken.

Creating a map that modifies the context

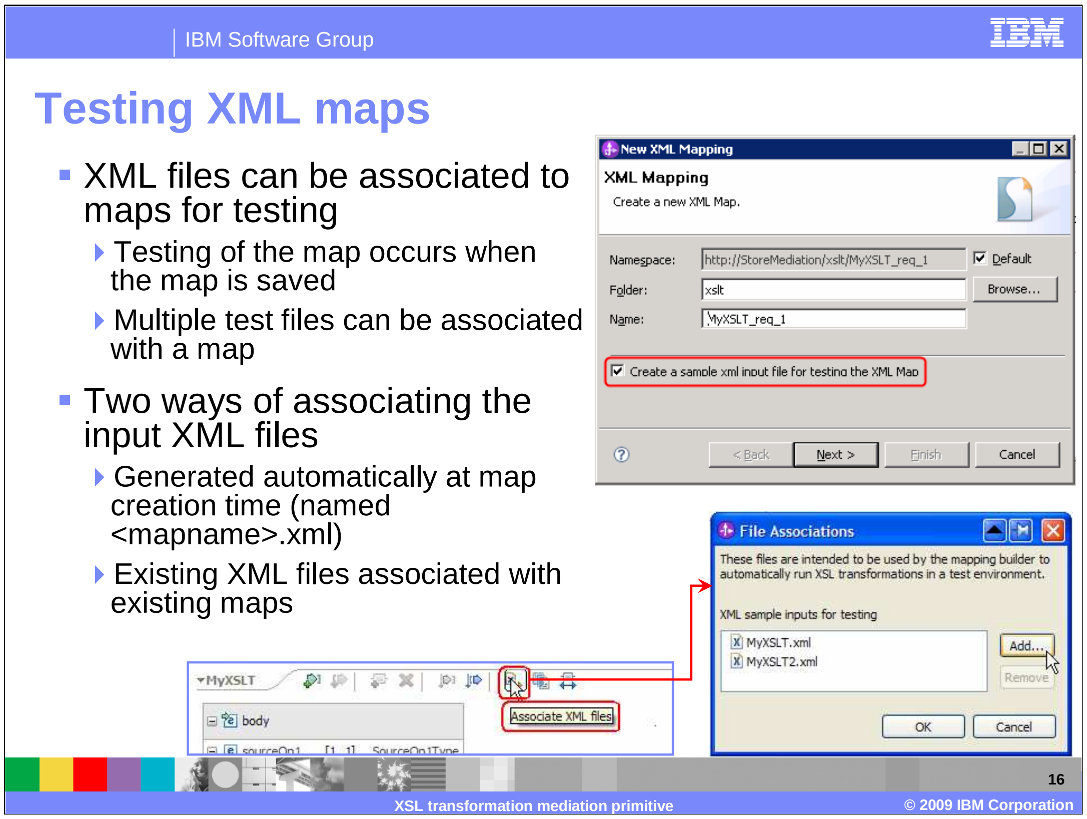
- Example (continued)
  - Open the merge transform (there is a move from source customer to target customer)



  - Delete connection from source customer to move transform and add connection from customerID



Finishing up the example, open the merge transform. Similar to the previous local map, simple types are moved with a move and complex types are mapped with a local map. The customer field from the source transient context is moved to the customer field in the target transient context. This is shown in the upper screen capture. To complete this example, the connection from the source customer to the move transform must be deleted, and a connection from the customerID added. When this is done, there is now a move transform setting the customer field in the target transient context from the customerID field in the message body. The result is seen in the lower screen capture.



IBM Software Group

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

New XML Mapping

XML Mapping

Create a new XML Map.

Namespace: http://StoreMediation/xslt/MyXSLT_req_1 ☑ Default

Folder: xslt

Name: MyXSLT_req_1

☑ Create a sample xml input file for testing the XML Map

File Associations

These files are intended to be used by the mapping builder to automatically run XSL transformations in a test environment.

XML sample inputs for testing

MyXSLT.xml

MyXSLT2.xml

16

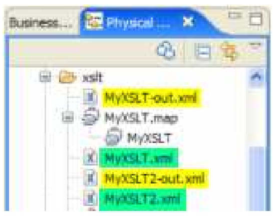
XSL transformation mediation primitive

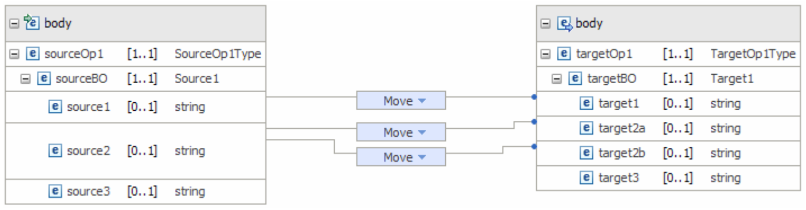
© 2009 IBM Corporation

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





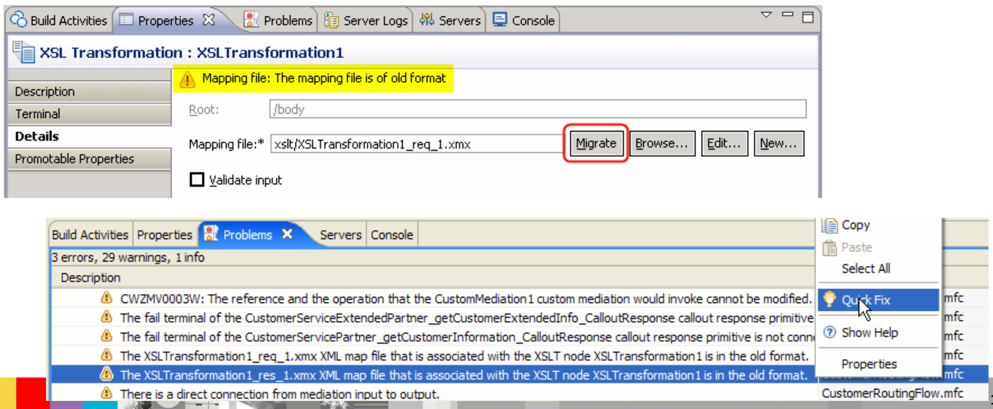
```
<?xml version="1.0" encoding="UTF-
<body xmlns:p="http://StoreMediati
  <tns:sourceOp1>
    <sourceBO>
      <source1>source1</source1>
      <source2>source2</source2>
      <source3>source3</source3>
    </sourceBO>
  </tns:sourceOp1>
</body>
```

```
<?xml version="1.0" encoding="UTF-8
<body xmlns:out2="http://StoreMedia
  <out2:targetOp1>
    <targetBO>
      <target1>source1</target1>
      <target2a>source2</target2a>
      <target2b>source2</target2b>
    </targetBO>
  </out2:targetOp1>
</body>
```

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 – New map editor in V6.1

- A new XML map editor was introduced 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



XSL Transformation : XSLTransformation1

Mapping file: The mapping file is of old format

Root: /body

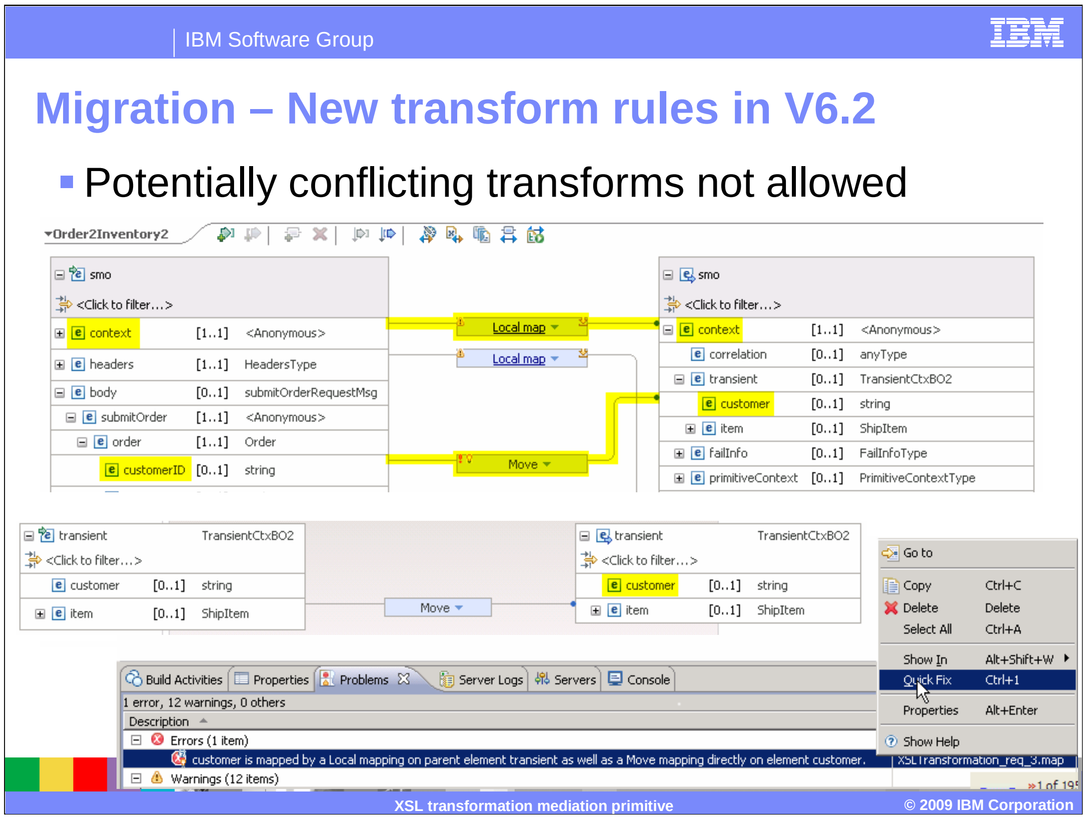
Mapping file:* xslt/XSLTransformation1_req_1.xmx

☐ Validate input

3 errors, 29 warnings, 1 info

Description
- CWZMV0003W: The reference and the operation that the CustomMediation1 custom mediation would invoke cannot be modified. mfc
- The fail terminal of the CustomerServiceExtendedPartner_getCustomerExtendedInfo_CalloutResponse callout response primitive mfc
- The fail terminal of the CustomerServicePartner_getCustomerInformation_CalloutResponse callout response primitive is not conn mfc
- The XSLTransformation1_req_1.xmx XML map file that is associated with the XSLT node XSLTransformation1 is in the old format. mfc
- The XSLTransformation1_res_1.xmx XML map file that is associated with the XSLT node XSLTransformation1 is in the old format. mfc
- There is a direct connection from mediation input to output. CustomerRoutingFlow.mfc

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 and that there is a Migrate button for the mapping file. Also, the Edit… button opens a migration panel rather than an editor. 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 becomes operational. 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.



IBM Software Group

Migration – New transform rules in V6.2

- Potentially conflicting transforms not allowed

1 error, 12 warnings, 0 others

Description

Errors (1 item)

customer is mapped by a Local mapping on parent element transient as well as a Move mapping directly on element customer. XSLTransformation_req_3.map

Warnings (12 items)

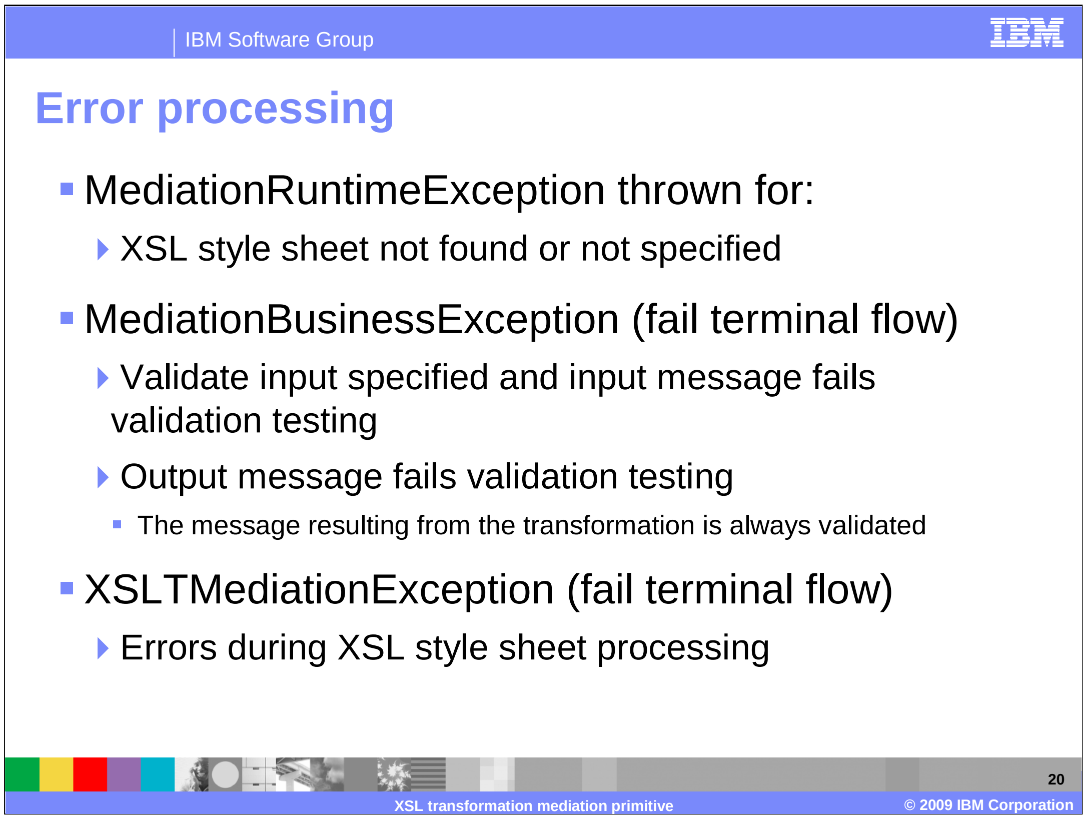
XSL transformation mediation primitive © 2009 IBM Corporation

This slide looks at new rules for the structure of an XML map that took effect starting in version 6.2 and how to migrate your maps from earlier versions.

To illustrate this, the example shown on previous slides is revisited. It showed how to create a map that updated a field embedded within the transient context. What that example showed was the correct way to do it, by using merge transforms and drilling down to the level where the field can be set.

What is shown here is an example of a legal XML map in version 6.1 that is no longer allowed as of version 6.2. In the top screen capture is an XML map that uses a local map to copy the source context to target context. It also contains a move of the customerID field from the body to the customer field in the transient context. In the middle of the slide is a screen capture of the local map for the transient context, and notice that there is no transformation to set the customer field. In version 6.1 this was a legal XML map. However, there are cases where a construct similar to this might not yield the expected results. Therefore, in version 6.2 this is no longer a valid map. See the screen capture at the bottom of the slide with the error stating: “customer is mapped by a local mapping on parent element transient as well as a Move mapping directly on the element customer”.

It is possible you have version 6.1 projects that contain XML maps similar to this. In order to aid migration, there is a Quick Fix option provided on the pop-up menu for the error message. Selecting this should correct your map so that it uses the structure now supported. Before using the quick fix, make sure you understand why your existing map is being flagged and what the proper solution is. After the quick fix completes, ensure that your map was updated as you expected.



Error processing

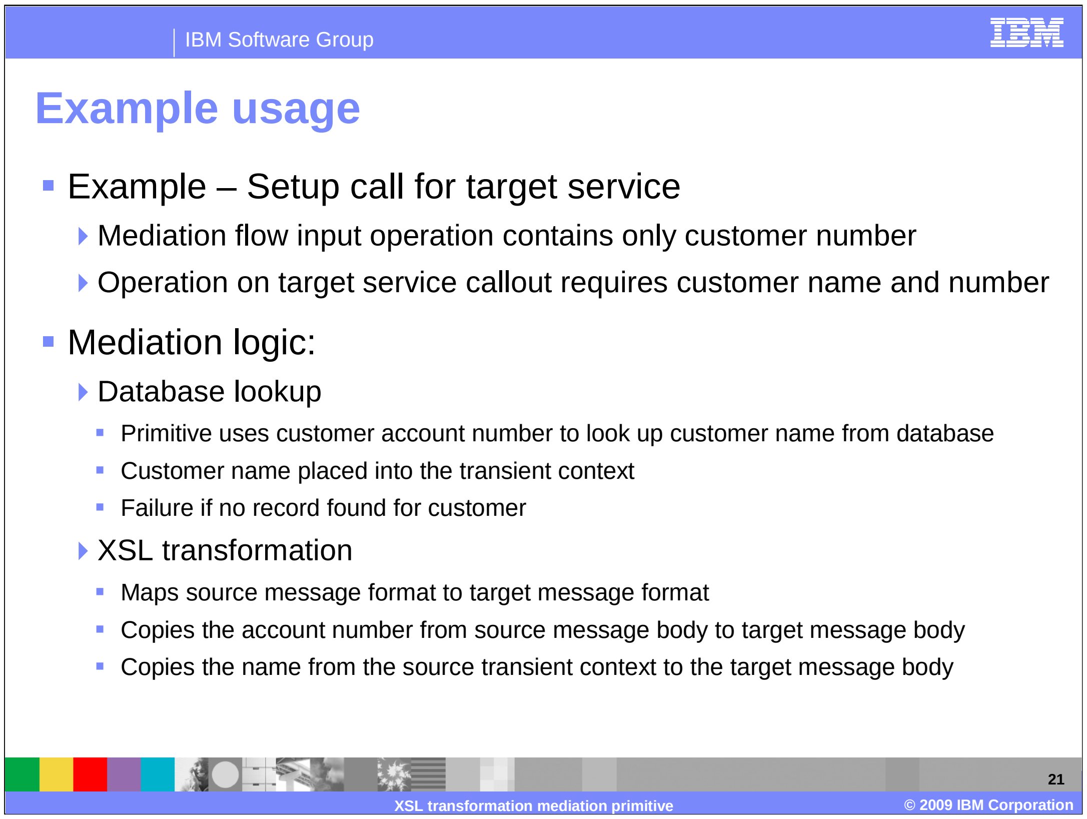
- MediationRuntimeException thrown for:
  - XSL style sheet not found or not specified
- MediationBusinessException (fail terminal flow)
  - Validate input specified and input message fails validation testing
  - Output message fails validation testing
    - The message resulting from the transformation 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.



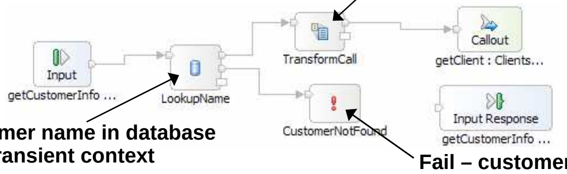
Example usage

- 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
    - Primitive uses customer account number to look 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.

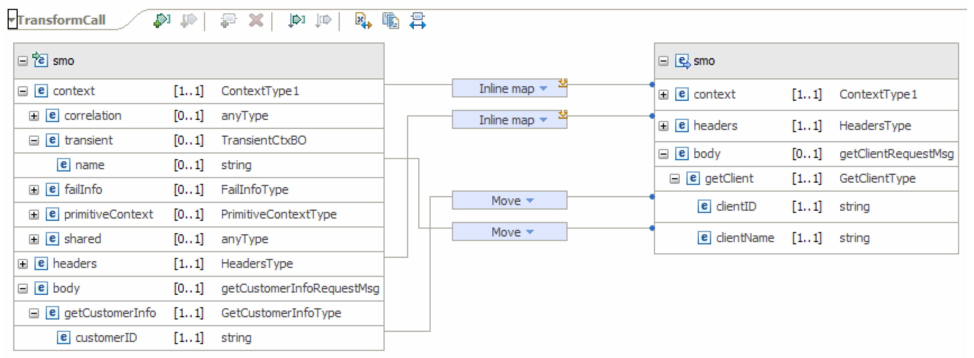
Example usage (continued)

Transform format of message body and initialize with customer number and name



Lookup customer name in database and place in transient context

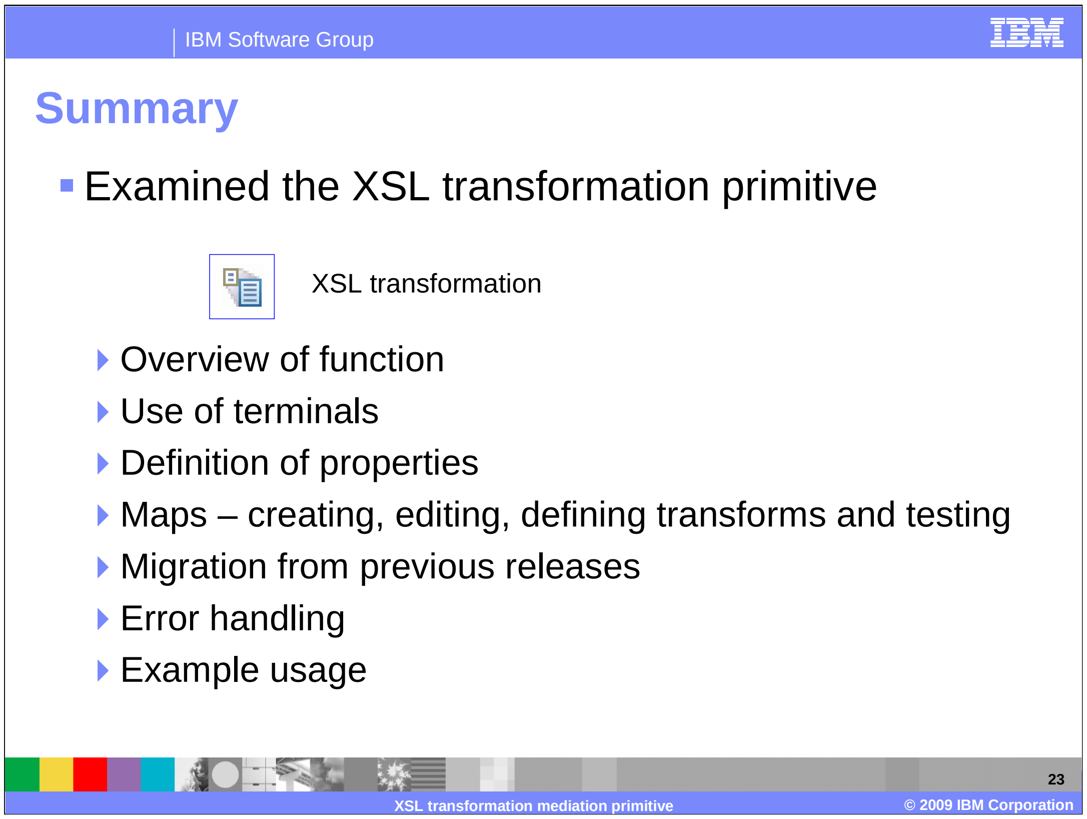
Fail – customer record not in database



smo		
context	[1..1]	ContextType1
correlation	[0..1]	anyType
transient	[0..1]	TransientCtxBO
name	[0..1]	string
failInfo	[0..1]	FailInfoType
primitiveContext	[0..1]	PrimitiveContextType
shared	[0..1]	anyType
headers	[1..1]	HeadersType
body	[0..1]	getCustomerInfoRequestMsg
getCustomerInfo	[1..1]	GetCustomerInfoType
customerID	[1..1]	string

smo		
context	[1..1]	ContextType1
headers	[1..1]	HeadersType
body	[0..1]	getClientRequestMsg
getClient	[1..1]	GetClientType
clientID	[1..1]	string
clientName	[1..1]	string

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. Finally, there are transforms that move the entire context and the entire header. On the screen capture, which was taken using version 6.1, the transforms are labeled inline map, but starting in version 6.2 these are now called local maps.



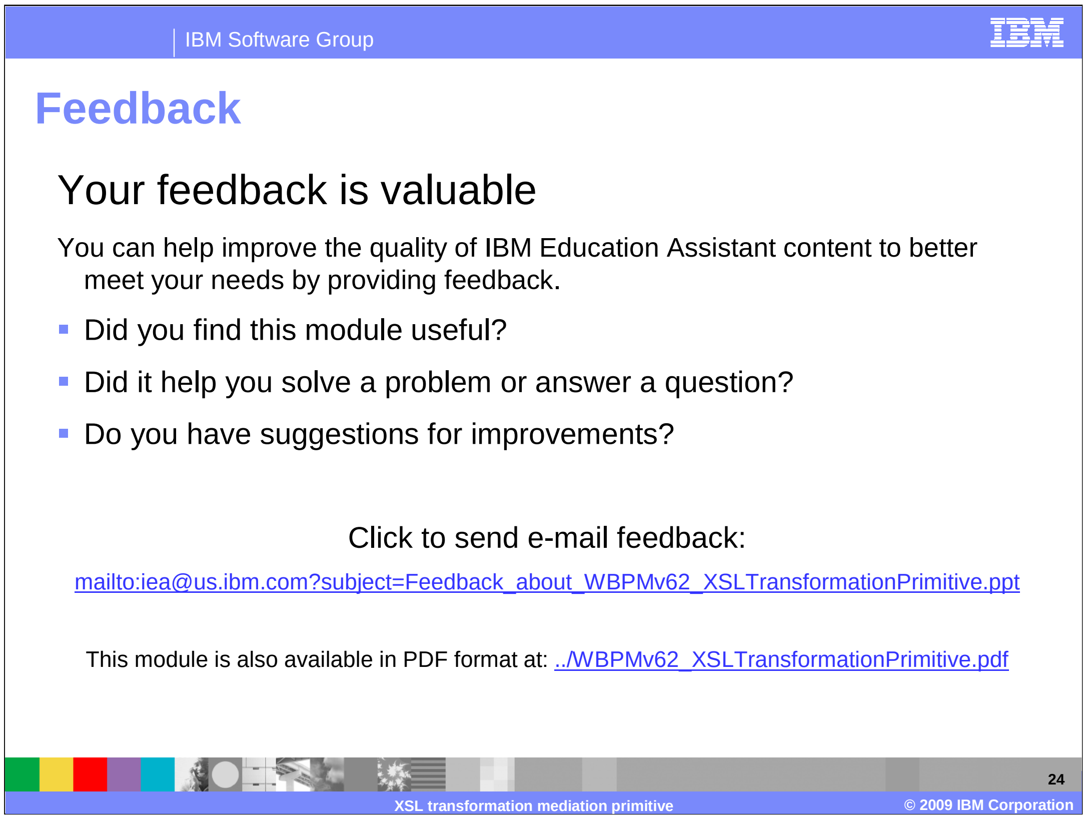
Summary

- 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.

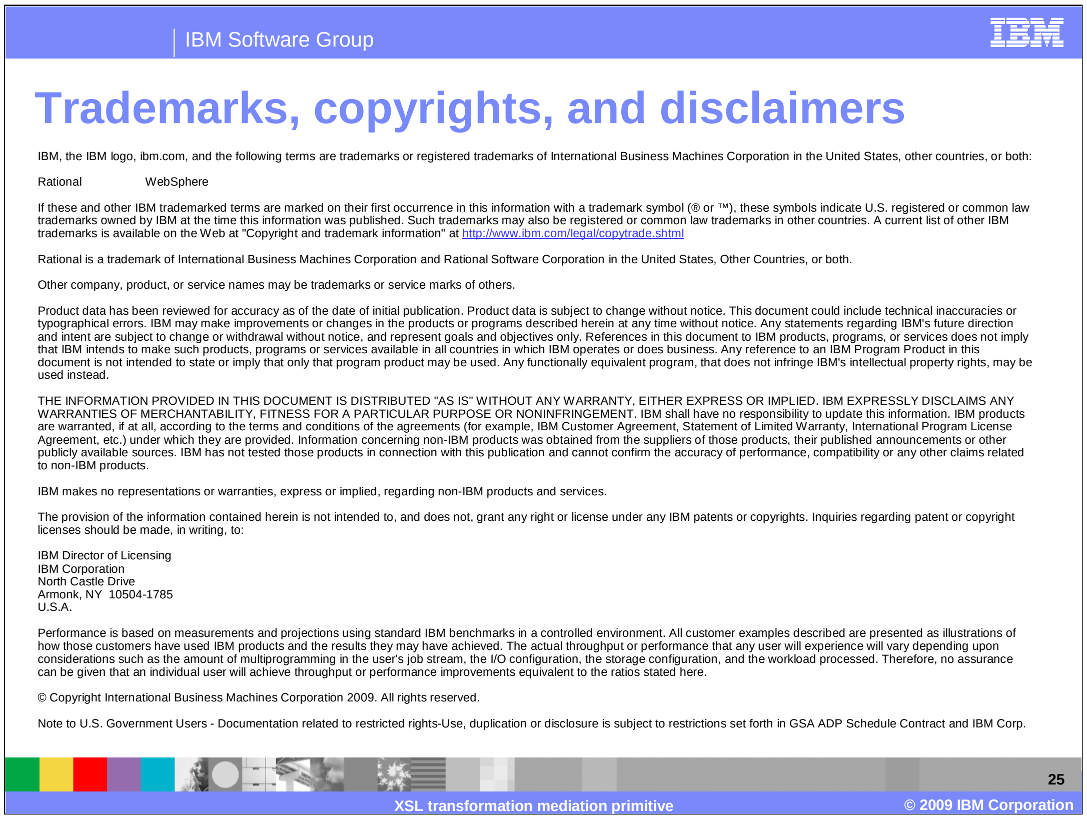
- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send e-mail feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_WBPMv62_XSLTransformationPrimitive.ppt

This module is also available in PDF format at: ../WBPMv62_XSLTransformationPrimitive.pdf

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



Trademarks, copyrights, and disclaimers