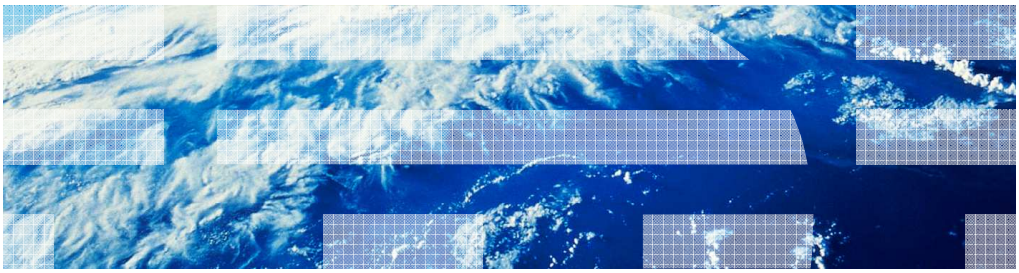
WebSphere Business Process Management

WebSphere Integration Developer
WebSphere Enterprise Service Bus
WebSphere Process Server

Transformation primitives affecting message schema and contents



WebSphere software

This presentation introduces the transformation primitives that are capable of transforming the message schema and the contents of the message.

Goal and agenda

- Goal
  - Provide an introduction to mediation primitives with these characteristics:
    - Classified in WebSphere Integration developer as transformation primitives
    - Capable of modifying the schema of the service message object (SMO)
    - Capable of modifying the contents of any part of the SMO
- Agenda
  - Describe the overall common characteristics of these primitives
  - Introduce the basic functionality of:
    - XSL transformation
    - Business object map
    - Data handler
    - Custom mediation

The goal of this presentation is to introduce a set of primitives that share some common characteristics. These primitives are classified within WebSphere Integration Developer as part of the transformation primitives group. Within that group, these are the primitives that have the capability to transform the schema of the service message object (SMO), which is the truest sense of transformation. They can access any part of the SMO to manipulate element values.

The presentation starts out by further explaining the common characteristics of these primitives, and then provides a description of the basic functionality of each. The primitives presented are the XSL transformation primitive, the business object map primitive, the data handler primitive and the custom mediation primitive.

Common characteristics

- Message schema transformation
  - Ability to transform the schema of the message
  - Input and output terminal message types need not be the same
  - Message type is not propagated across the primitive's terminals during wiring
- Configuration/customization is broader than just setting properties

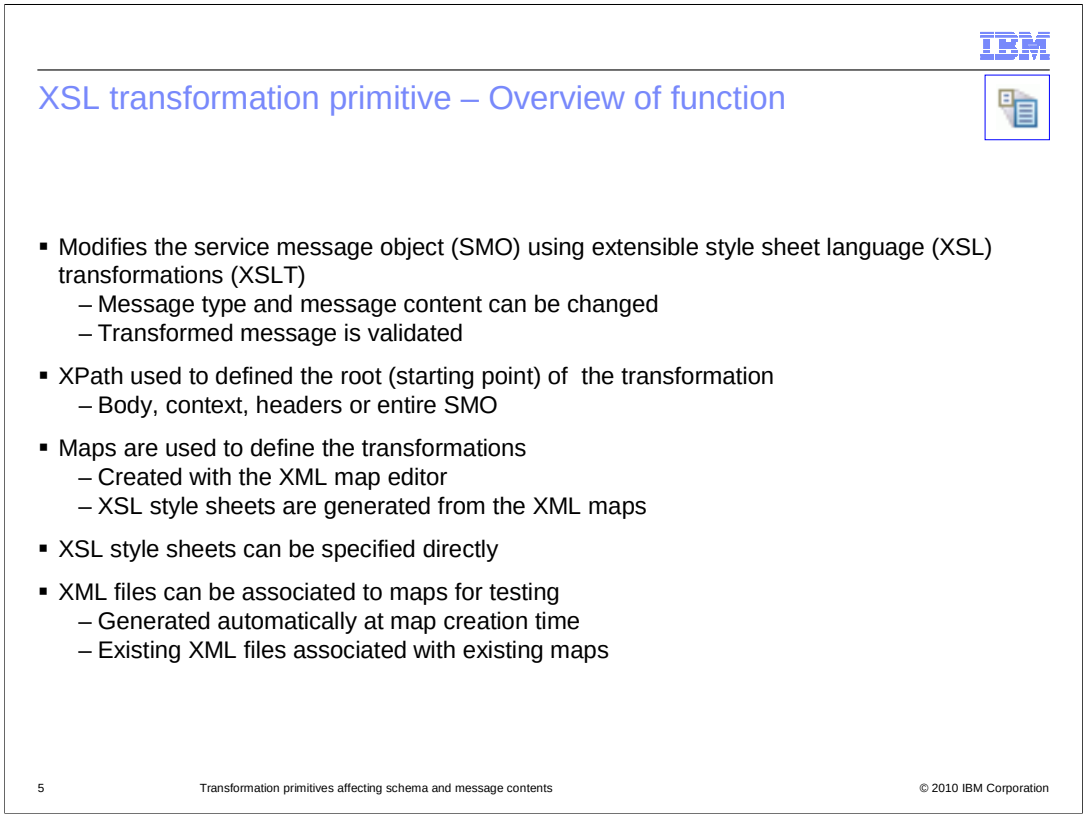
The key point of commonality between these primitives is that they are all capable of transforming the schema of the message as it flows through the mediation. Therefore, the input and output terminals need not be for the same message type, although that is possible as well. Since the input and output terminals do not need to be of the same message type, the normal message type propagation that occurs during wiring does not occur between the terminals of these primitives.

For most mediation primitives, there is a defined function that is configured through the use of properties. These primitives also make use of properties, but the real configuration and customization of these is broader than the setting of some properties. Two of the primitives make use of maps that get generated into runtime artifacts defining the actual transformation. The other two make use of Java code to perform the transformation, one packaged as a data handler and the other with code specifically added to the primitive. This broader scope of how these primitives are customized somewhat separates them from most of the other primitives.

Section

XSL transformation primitive

This section introduces the XSL transformation primitive and provides links to additional resources for understanding its usage.



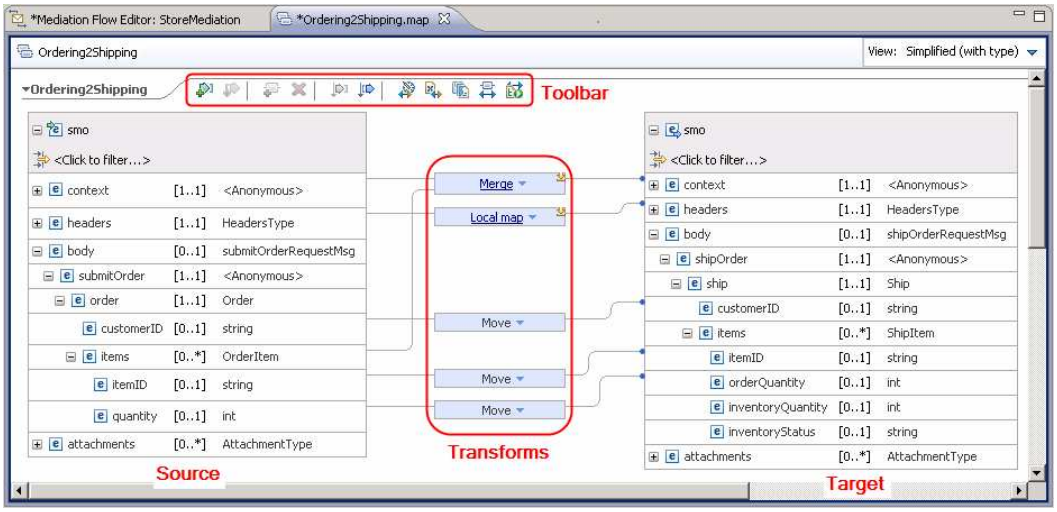
The purpose of the XSL transformation primitive is to modify the service message object (SMO). This is done using extensible style sheet language (XSL) transformations, also called XSLT. The XSL transformation primitive is capable of modifying the content of the SMO and of modifying the message type by restructuring the message body. The transformed message is validated to ensure it conforms to the constraints specified by the schema. 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. This can be set to transform the entire SMO or to only transform a major section of the SMO, namely the body, context or headers.

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.

WebSphere Integration Developer provides an automatic testing capability. It takes an XML file associated with the map and uses it as input to the map whenever the map file is saved, creating the resulting output XML file. 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. The other approach is to associate an existing XML file with a map.

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 field 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 the mapping of like fields between source and target.

There are several resources available to help you further understand the XSL transformation primitive and its usage. The first is a series of articles in developerWorks addressing the use of the mapping editor and considerations when mapping between complex XML structures. The next links are to the information center to a general topic of transforming messages and then the section specifically on XSL transformations. Finally, there is IBM Education Assistant content. The first link is to a presentation addressing the XSL transformation primitive in V6.2 and the second is to a presentation with several slides highlighting the enhancements to XML mapping that were introduced in V7.

Section

Business object map primitive

This section introduces the business object map primitive and provides links to additional resources to help with further understanding of this primitive's usage.

Business object map primitive – Overview of function

- Modifies the service message object (SMO) using generated Java code
  - Message type and message content can be changed
- XPath used to defined the root (starting point) of the transformation
  - Body, context, headers or entire SMO
- Maps defining the transformations are created with the business object map editor
  - Editor creates XML to represent the map, which is then used to generate Java code
  - Editor layout and basic interaction similar to the XML map editor
- In addition to use in mediations, business object maps that are used for:
  - Mapping service APIs
  - Parameter mapping in interface maps (deprecated)
- Business object maps support:
  - The relationship service
  - The use of business graphs with change summary and event summary
  - Configuration of event settings to raise CEI events

The business object map primitive is used to modify the service message object using generated Java code. In most cases, it is used to change the structure of the SMO body, thus changing the message type of the SMO. However, in some cases the business object map might be used to update the contents of the SMO without changing the message type. The entire SMO can be affected by this primitive, or it can be scoped to only act upon the context, or the headers or the body. The root property is an XPath expression used to identify which part of the SMO is transformed

The business object map editor is used in WebSphere Integration Developer to create and edit business object maps. The editor generates an XML representation of the map which is later generated into executable Java code providing the runtime implementation of the map. The actual editor layout is very similar to that of the XML map editor used by the XSL transformation primitive.

In addition to their use in mediation flows, business object maps can be called directly using the mapping service APIs. They also serve as parameter maps, called from within an interface map to transform parameter data. It should be noted that the use of interface maps has been deprecated, with the preferred implementation being the use of a mediation flow component with either a business object map or XSL transformation primitive.

Business object maps support the use of the relationship service for both static and identity relationships. They also support the business objects that are contained in business graphs, where a change summary and event summary is maintained. Also, within a business object map, CEI events can be configured to cause events to be generated based on the transformations in the map.

Business object map versus XSL transformation

- The business object map primitive and XSL transformation primitive perform similar function
  - Raises the question of which to use
- General guideline → Use XSL transformation unless you have a specific reason not to
- Reasons to use the business object map primitive instead of XSL transformation
  - You already have existing maps that can be reused
  - The mapping operation needs to use identity relationships
  - Your business objects are encapsulated in business graphs with change summaries
  - There is a requirement to perform individual transformations in a specified order
  - You are performing custom transformations based on SDO access

There is a great deal of similarity in the business object map primitive and the XSL transformation primitive, both in terms of the mapping editor interfaces and in the resulting transformations. Because of this, there might be confusion about which one should be used for any particular scenario. As a general rule, you should always use the XSL transformation primitive unless there is a specific reason why the business object map primitive is needed to meet your requirements. There are a handful of reasons to use the business object map primitive rather than the XSL transformation primitive.

One of the reasons is that you might already have existing business object maps that perform the required transformations. When this is the case, it makes sense to reuse the existing maps.

Another reason is if your mapping operation has a requirement to use the relationship service. There are two kinds of relationships, identity relationships and static relationships. You must use a business object map to make use of the relationship service for either of these relationship types. However, XML maps do provide a functional equivalent to static relationship capabilities which might meet your requirements.

Some integration scenarios make use of business graphs, which are used to wrapper business objects and maintain change summaries and event summaries. Only business object maps support the updating of the business graph with summary information regarding the transformations performed by the map.

In some mapping scenarios, the order in which the transformations are performed is critical to obtaining the correct result. Only business object maps allow you to explicitly specify the order in which the transformations are performed.

One other reason to use a business object map is to perform custom transformations that directly access the SDO representation of the business object.

As you can see, there are several reasons to choose a business object map primitive over an XSL transformation primitive, but remember that the general guideline is to use XSL transformation primitives in all other cases.

Resources

- Information center
  - Business object map mediation primitive
  - Transforming data using a business object map
  - XML maps versus business object maps
- IBM Education Assistant
  - Business object map primitive presentation for V6.2

There are resources available to help you further understand the business object map primitive and its usage. In this slide there are links to the information center for topics covering business object maps. There is also IBM Education Assistant content in a presentation addressing the business object map primitive in V6.2. This covers the primitive in more detail and is useful in that there have been no significant changes in the primitive since that version.

Section

Data handler primitive

This section introduces the data handler primitive and provides links to additional information useful for understanding when and how to use this primitive.

Data handler primitive – Overview of function

- The data handler primitive:
  - Enables data transformation between:
    - Native formats and business objects
    - Business objects and native formats
  - Provides the same data transformation capabilities in a mediation flow as is normally done by an export or import
  - Is configured using a binding resource configuration
    - Combination of a data handler implementation and configuration parameters
- Data handlers and resource configurations can be:
  - Prepackaged implementations provided with the product
  - Custom implementations provided by you

The purpose of the data handler primitive is to enable transformations between business objects and native data formats within a mediation flow. The type of transformations enabled are the same as those normally done at the edges of the flow by an export or an import. Similar to the export and import, the data handler is configured with a binding resource configuration. Binding resource configurations combine a data handler implementation with configuration parameters, the combination of which define the transformations that the primitive performs.

Some data handler implementations are provided with the product, as are some binding resource configurations. You can use a provided resource configuration if it meets your requirements. Alternatively, you can create a new resource configuration with a provided data handler, or you can provide your own custom implementation of a data handler if needed.

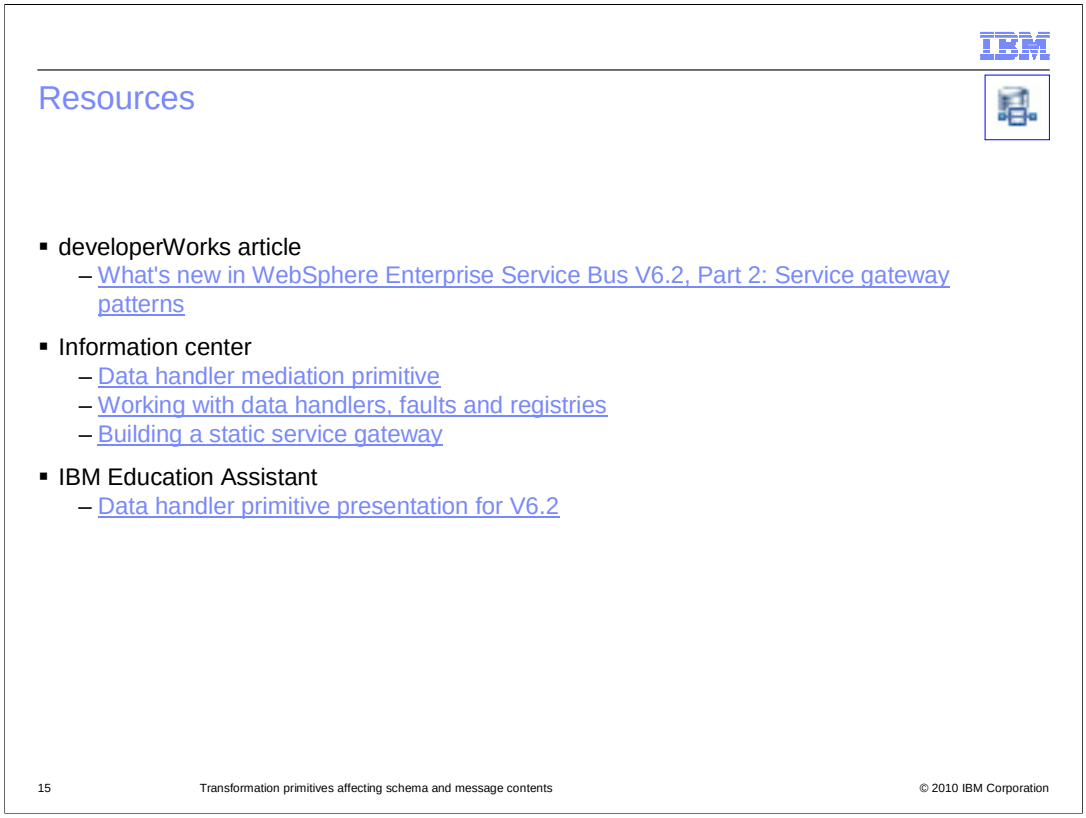
Data handler primitive usage

- Data handler primitive usage
  - Static service gateway pattern
    - Export passes the incoming native data without any transformation
    - Native data examined to determine transformation needed
      - For example, using a type filter primitive
    - Mediation flows to appropriate data handler for required transformation to a business object
    - Reverse transformation occur on the response flow
  - Encoded data embedded within the message
    - SMO contains field containing encoded data
      - For example, a string of comma separated values
    - Encoded data needs to be transformed to a business object representation within the SMO
    - Reverse transformation also applies

In some scenarios it makes sense to perform transformations between business objects and native formats within a mediation flow, rather than on the edges as is normally done using imports and exports. It is in these scenarios that the data handler primitive is useful.

One such scenario is a mediation implementing a static service gateway pattern. With this pattern, the export supports the generic service gateway interface, and the imports support some concrete interfaces. The export passes the data to the mediation flow without any transformation of the inbound native data. Within the flow, the native data is examined to determine the type of transformation that is needed, which might be done using a type filter, message filter or custom mediation primitive. The mediation then flows to an appropriate data handler primitive to perform the native format to business object transformation. The response flow will contain a data handler primitive to perform the business object to native format transformation of the return data.

Another scenario for use of the data handler primitive is when a business object contains a field with encoded data, such as a comma separated value string. Within the flow, the data handler primitive can be used to perform the transformation of the encoded data to a business object. Reverse transformations from business object to encoded data can also be done when required to build an outbound message.



There are resources available to help you further understand the data handler primitive and its usage. The first is to a developerWorks article that addresses the introduction of service gateway patterns in V6.2. This article provides more details for using a data handler primitive in a service gateway. There are also some links to the information center for topics covering data handler primitives and their usage. Finally, there is IBM Education Assistant content in a presentation addressing the data handler primitive in V6.2. This covers the primitive in more detail and is useful in that there have been no significant changes in the primitive since that version.

Section

Custom mediation primitive

The custom mediation primitive is addressed in this section. An overview of its function is provided along with links to additional resources.

Custom mediation primitive - Overview of function

- Enables the use of custom mediation logic
  - Use when no built-in primitive provides a needed function
  - Logic might be for simple tasks
  - Also enables complex routing and transformation logic
- Logic implemented in Java™ and defined as a:
  - Visual snippet
  - Java snippet
- Access to entire SMO using:
  - ServiceMessageObject APIs
  - BusinessObject APIs
  - DataObject APIs

The custom mediation primitive enables you to define your own custom mediation logic for use when the built-in primitives do not provide the needed functionality. In some cases this might be simple logic, such as formatting and printing a message to the system log file. You might want to do this in a case where the format is something that can not be accomplished with the trace primitive. Additionally, the custom mediation primitive is able to be used for complex transformation and routing logic, which enhances the overall possibilities for what you can do in a mediation flow.

The logic in a custom mediation primitive is implemented in Java. There are two approaches to doing this, using either a visual snippet or a Java snippet.

Within a custom mediation primitive, you have access to the entire SMO. This enables you to access and manipulate the contents of the SMO using the strongly typed ServiceMessageObject APIs, the BusinessObject APIs or the loosely typed DataObject APIs.

Custom mediation primitive - Overview of function (continue)

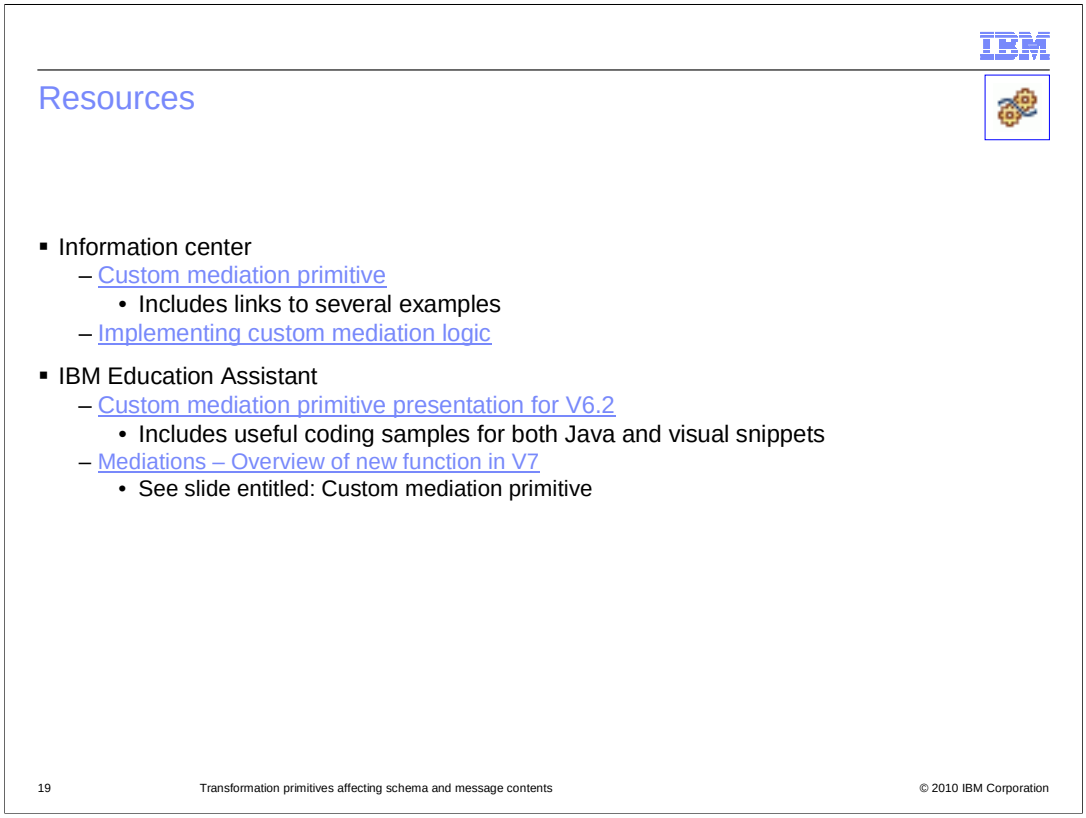
- Variable number of user defined input terminals
  - Terminals can be for differing message types
  - Input terminal object can be queried in code to determine which terminal received the inbound message
- Variable number of user defined output terminals
  - Terminals can be for differing message types
  - Outbound message can be fired on one or more terminals
- User defined properties that are promotable
  - Allows the code to be configurable
  - Can be used for administrative control, policy control or subflow invocation

The custom mediation primitive allows you to define how many input and output terminals you require, each of which can support a different message type. This is one of the key elements that makes the custom mediation primitive capable of complex transformation and routing logic.

For inbound messages, there is an input terminal object which is passed as a parameter. It can be queried to obtain the name of the terminal through which the message was received. This allows the code to then perform the processing that is appropriate for that particular message type.

For outbound messages, one or more output terminals can be fired during the processing logic of the custom mediation primitive. The message type of the SMO can be changed between firing of terminals, thus enabling multiple flow paths to be invoked with messages of differing type.

The ability to define user properties which are promotable enables you to expose aspects of your custom mediation primitive implementation. This allows you to design code that is configurable and enables administrative control or policy control at runtime. It can also be used for configuration of a subflow invocation.



The links on this slide bring you to additional resources for understanding the custom mediation primitive. The first couple of links are to the information center. The first link documents the primitive, and at the bottom of that page you will find links to several different code examples. The next link brings you to a page that is basically just links to more specific topics on the use and implementation of custom mediation primitives.

Following that are links are to IBM Education Assistant. There is a detailed presentation on the custom mediation primitive in V6.2. This includes some code samples that are useful when implementing a custom mediation, such as how to access the SMO, determining the input terminal used and how to fire an output terminal.

Although the custom mediation primitive itself did not change from V6.2, the next link is to a presentation describing the new function in V7. It describes a new API which was added for accessing WebSphere Service Registry and Repository. The API is intended to be used from within a custom mediation primitive.

Summary

- Described the overall common characteristics of these primitives
- Introduced the basic functionality of:
  - XSL transformation
  - Business object map
  - Data handler
  - Custom mediation

In summary, the presentation started out by explaining the common characteristics of this group of primitives that are capable of transforming the message schema. It then described the basic functionality of each. The primitives presented were the XSL transformation primitive, the business object map primitive, the data handler primitive and the custom mediation primitive.

Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

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

Click to send e-mail feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_WBPMv7_XFormSchemaPrimitives.ppt

This module is also available in PDF format at: ../WBPMv7_XFormSchemaPrimitives.pdf

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

Trademarks, disclaimer, and copyright information