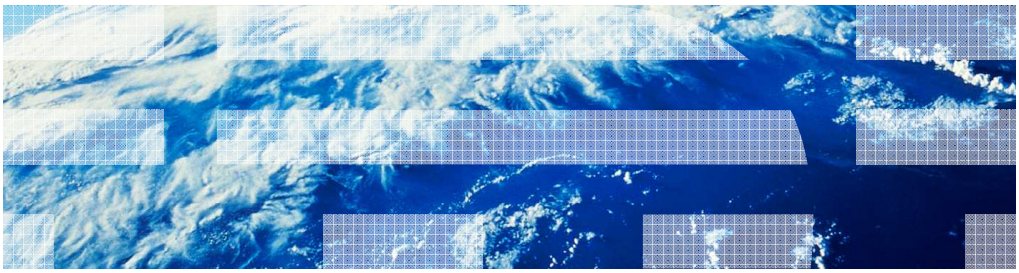
WebSphere Business Process Management

WebSphere Integration Developer
WebSphere Enterprise Service Bus
WebSphere Process Server

Transformation primitives to manipulate message content



WebSphere software

This presentation introduces the transformation primitives that are capable of manipulating the message content.

Goals and agenda

- Goal
  - Provide an introduction to mediation primitives with these characteristics:
    - Classified in WebSphere Integration developer as transformation primitives
    - Either:
      - Capable of accessing and modifying any part of the SMO
      - Augmenting the message type without modifying the message
- Agenda
  - Introduce the basic functionality of:
    - Message element setter
    - Database lookup
    - Set message type

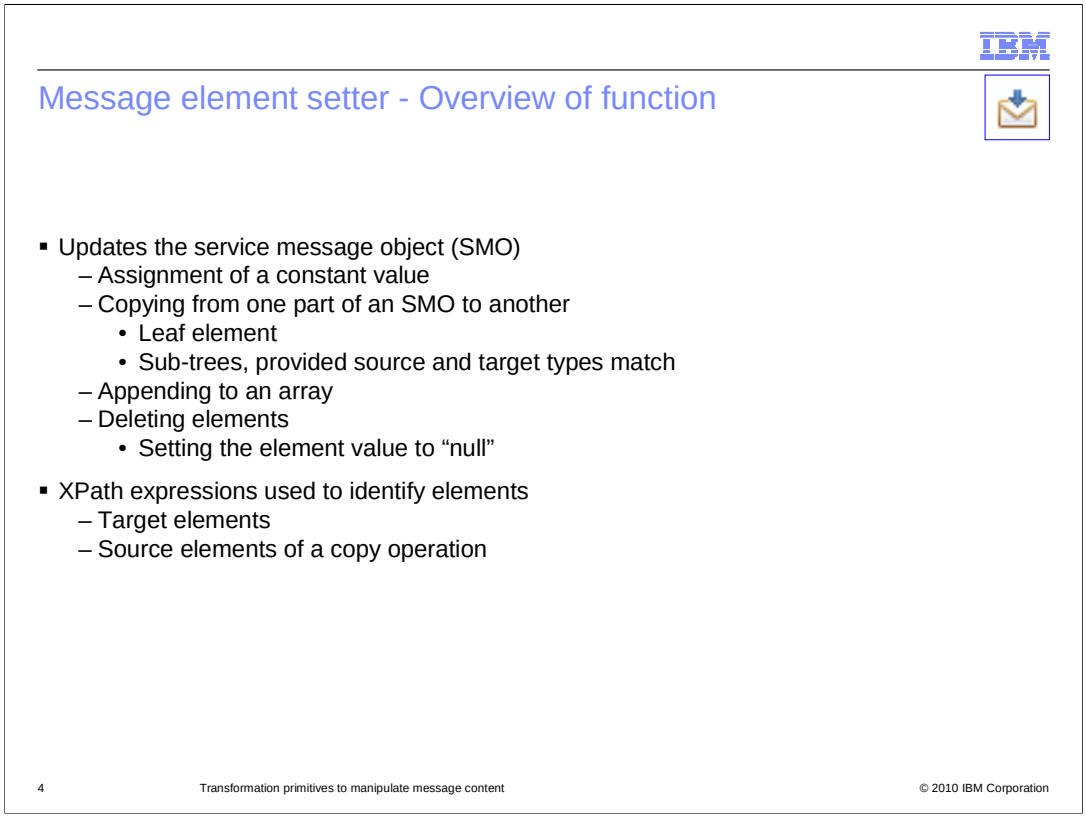
The goal of this presentation is to introduce a set of primitives that share some common characteristics. These primitives are some of those classified within WebSphere Integration Developer as transformation primitives. This is really a catch all group, and not all of the primitives have one common characteristic. A couple of the primitives are capable of modifying element values within any part of the service message object (SMO). The other one is unique, and does not modify the message, as all the other transformation primitives do. Rather, it augments the message type with additional type information for loosely typed elements.

The primitives presented are the message element setter, the database lookup and the set message type.

Section

Message element setter

The message element setter is described in this section.



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

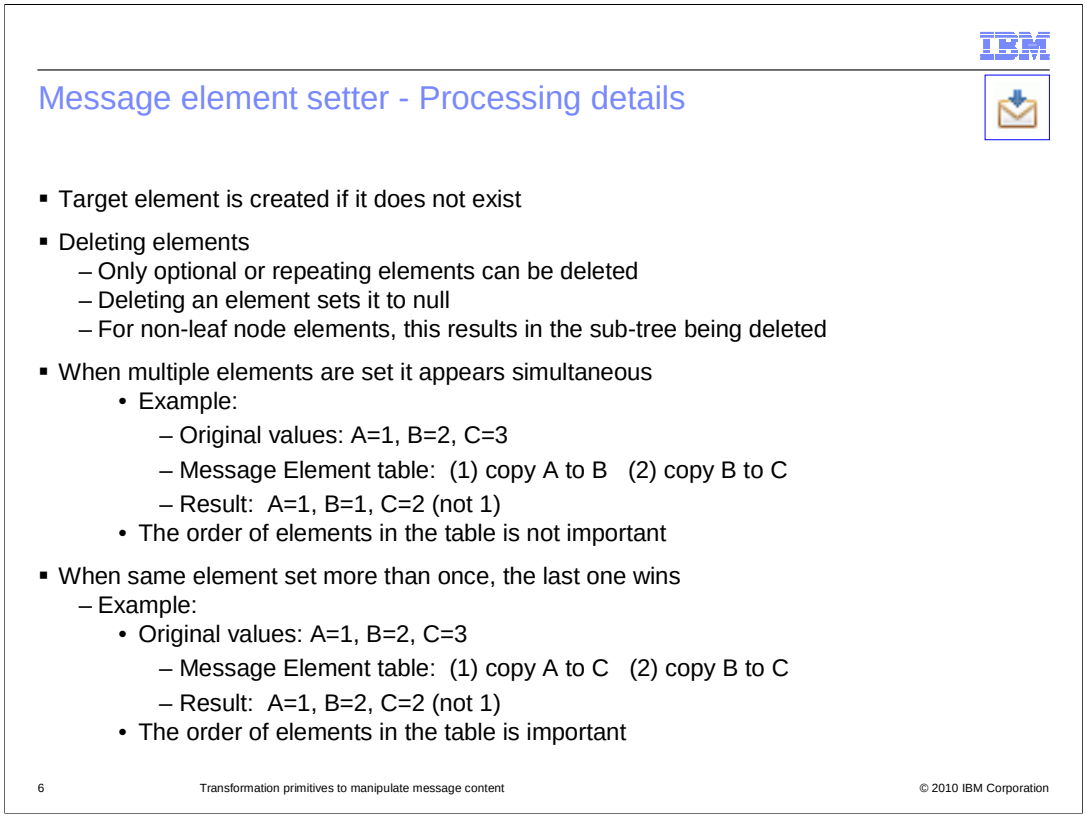
XPath expressions are used to identify the target elements in the SMO that are updated by the primitive. The source elements of copy and append operations are also identified using XPath expressions.

Message element setter - Overview of function (continue)

- Multiple elements can be updated
  - Table based specification of properties
- Easier than coding other primitive types
  - Custom mediation
  - XSL transformation
  - Business object map
- More efficient than other primitives
  - XSL transformation
  - Business object map primitives

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

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



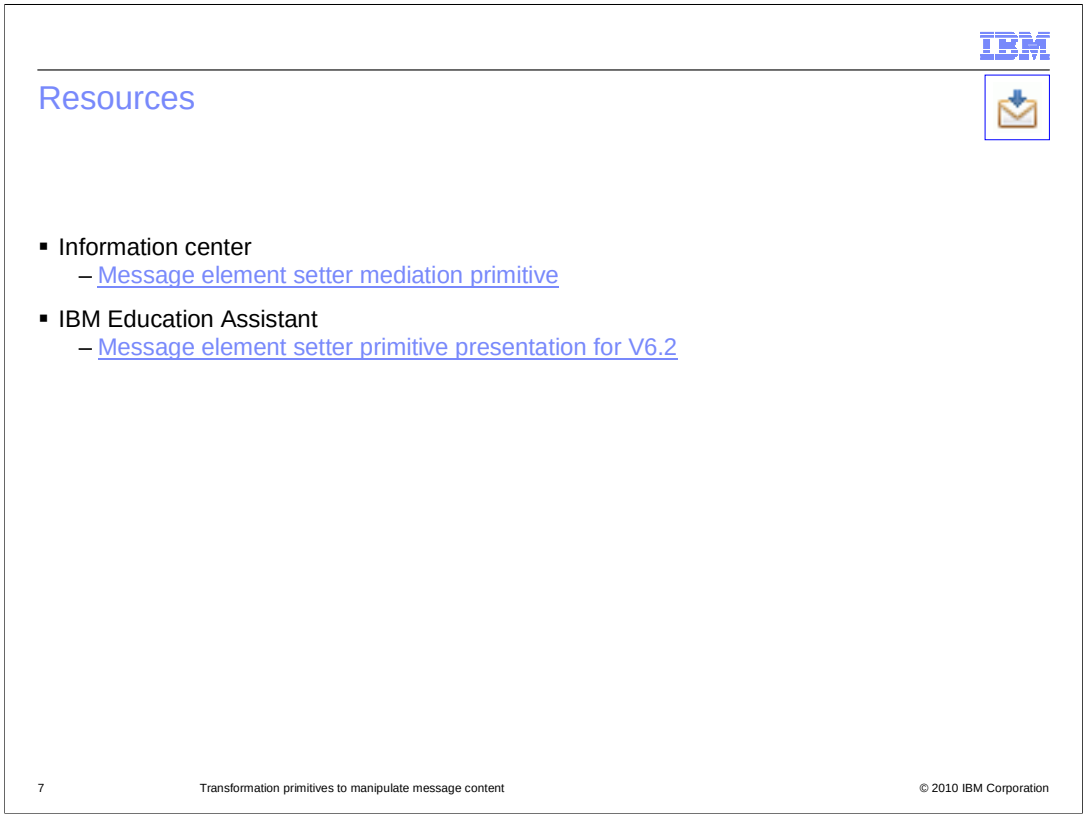
For some cases, it can be important to understand the nuances of behavior exhibited by the message element setter primitive. Several of the processing details for the primitive are provided here.

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

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

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

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



The information center provides documentation of the message element setter primitive, describing its usage, documenting its properties and providing some considerations for its use. A more complete presentation of the message element setter is provided in IBM Education Assistant for V6.2, which is still applicable to V7.

Section

Database lookup

The database lookup primitive is the topic of this section.

Database lookup primitive - Overview of function

- Updates the SMO with data from a user database
  - A key value is obtained from the message
  - The database row associated with that key is accessed
  - The message is updated with values obtained from selected fields
- You must:
  - Create a database or use an existing database
  - Configure the data source identifying the database

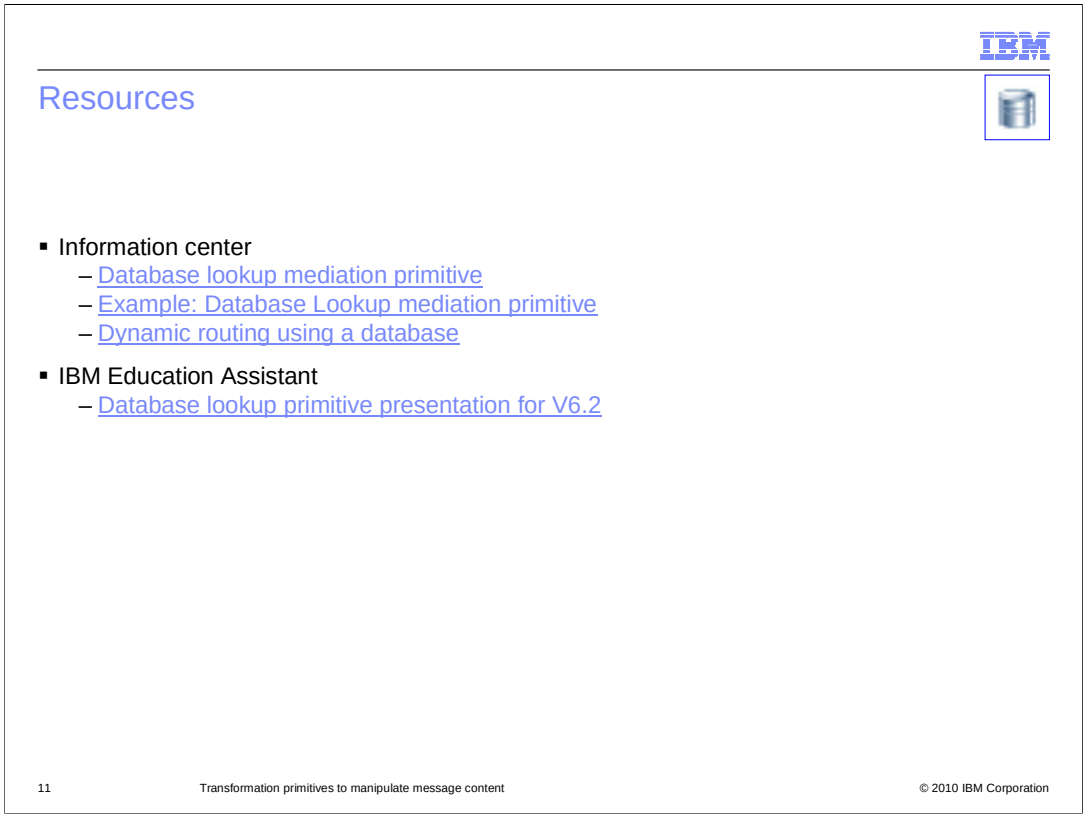
The purpose of the database lookup is to update the service message object with data obtained from a user database.

The database lookup primitive first obtains a key value from the SMO and uses that key value to access a row from a database table. Values from selected fields in that row are then used to update the SMO. Depending upon your requirements, you must either create a new database or use an existing database. You must configure a data source that is used to identify the database.

Database lookup primitive - Overview of function (continue)

- Configuration data used to define database access
  - Data source JNDI name
  - Table name
  - Key column
  - Data columns
- Configuration information for the SMO:
  - XPath expression identifying the element containing the key value
  - XPath expressions identifying the message elements to update
  - Validation of input SMO

There is quite a bit of configuration data associated with a database lookup primitive. Some of the configuration data is associated with the database and other configuration information refers to the SMO. The database configuration information includes the JNDI name for the data source and the name of the table on which to do the lookup. It also includes the column used for the key and the columns from which to extract data. The SMO configuration data includes the XPath expression that identifies the key value element in the SMO and the XPath expressions identifying the elements in the SMO to be updated with values from the database. There is also a configuration option to request validation of the incoming SMO.



The information center provides documentation of the database lookup primitive, describing its usage, documenting its properties and providing some considerations for its use. There is an example of using the database lookup showing how you might configure a database lookup primitive to lookup information from an employee database. Additionally, there is a discussion on how you might use a database, rather than a registry, to implement dynamic routing.

The IBM Education Assistant for V6.2 provides a more complete presentation of the database lookup primitive that is still applicable to V7.

Section

Set message type

This section introduces the set message type primitive.

Set message type primitive - Background

- Business object fields can be:
  - Strongly typed – type and structure is known
  - Weakly typed – more than one type of data allowed for the field
- Mediation flow editor uses type information
  - Contents of SMO understood through type information
    - SMO message type defines the contents of the body
    - SMO message type plays a key role in the wiring together of a mediation flow
    - Transient, correlation and shared contexts are defined through configuration of business objects
  - Type information critical for tools, such as:
    - XPath expression builder
    - Business object and XML mapping editors

Before explaining the function of the set message type primitive, it is useful to highlight some background information that provides a basis for understanding the function it provides.

The first key point to understand relates to the definition of fields in business objects. Some fields are strongly typed, so that the exact type and structure of data contained in the field is well known. However, some fields are weakly typed, allowing different types and structures of data to exist for that field.

The next major point is that the mediation flow editor makes extensive use of type information. At any point in a mediation flow, the service message object has an explicit message type which defines the contents of the message body. This message type defines the operation the message represents, including its parameters, which can be simple types or complex types defined by business objects. Message types are critical in the role that they play when constructing a mediation, particularly when it comes to wiring together the primitives in the flow. Additionally, the transient, correlation and shared contexts are all defined by business objects. Therefore, the SMO body and contexts can potentially have elements that are defined by weakly typed business object fields. This affects tools, such as the XPath expression builder, the business object map editor and the XML map editor, as they are highly dependent upon type information for the SMO context, headers and body.

Set message type - Overview of function

- Allows weakly typed fields in the SMO to be treated as strongly typed
- Enables editors to show and manipulate the strong rather than weak type
- Eliminates the need for custom Java code and hand written XSL style sheets
- Weakly typed fields can be:
  - xsd:any
  - xsd:anyType
  - xsd:anySimpleType
  - Concrete types from which other types are derived
    - For example, type Student from which UndergraduateStudent and GraduateStudent are both derived

With the information provided on the previous slide as background, the purpose of the set message type primitive can now be described. The set message type primitive provides a mechanism that allows weakly typed fields in the SMO to be treated as if they were strongly typed. If you are familiar with programming languages such as Java or C++, the use of a cast operation in these languages is a good analogy for what this primitive provides within a mediation flow. It declares a weakly typed field to be of a stronger type, therefore enabling the editors and other tools to show and manipulate the strong rather than the weak type. If you wanted to map elements within a weak type, you need to write custom Java code for business object maps or hand written XSL style sheets for XSL transformations. By using the strong type instead, you can create the maps directly in the editor without custom coding.

Being more specific about what is meant by a weakly typed field, the types xsd:any, xsd:anyType and xsd:anySimpleType are all used to define weakly typed fields. However, concrete types can also be considered weakly typed if they are inherited by other types, referred to as derived types. As an example of this, suppose there is a business object Student containing fields common to all students. From this, two other business objects are derived, UndergraduateStudent and GraduateStudent, each containing the appropriate additional fields for that type of student. An operation that defined a parameter of Student results in a weakly typed field in the SMO body.

Set message type - Overview of function (continue)

- Set message type primarily affects development tools
  - Enables editors to work with the more specific strong types
  - Simplifies the tasks required of the integration developer
- At runtime:
  - Has no effect on SMO structure or content
  - Optional validation enables checking content to ensure it is of asserted type
- Effect of set message type on terminal message type
  - Does not change the message type
  - The message type is augmented with additional type information
  - Wiring rules are adjusted for handling augmented message types

The set message type is different from all the other mediation primitives because it essentially provides only development time function. Use of this primitive enables editors to work with more specific types, thus making your tasks as an integration developer much easier. At runtime, this mediation primitive has no effect on the SMO in either structure or content. The only runtime functionality it provides is optional, allowing you to choose to have the set message type validate the SMO at runtime to ensure it contains the stronger type asserted by the primitive.

One last point about the function of this primitive is that it does not actually change the message type of the SMO that gets associated with a terminal. Rather, it augments the message type with additional type information. This results in some differences in the wiring rules for terminals and the affect that message type has on defining a flow.

Resources

- Information center
  - Set message type mediation primitive
  - Mapping weakly typed elements using the set message type mediation primitive
  - Building a static service gateway
- IBM Education Assistant
  - Set message type primitive presentation for V6.2

The information center provides documentation of the set message type primitive, providing details of its usage, describing its properties and providing some considerations for its use. There is another section on mapping weakly typed fields with the set message type primitive that goes into more details on its usage. Because the set message type primitive has a key role in the logic of a static service gateway, a link is provided to the section of the information center describing how to build a static service gateway.

The IBM Education Assistant for V6.2 provides a more complete presentation of the set message type primitive that is still applicable to V7.

Summary

- Introduced the basic functionality of:
  - Message element setter
  - Database lookup
  - Set message type

In this presentation you were introduced to the basic functionality of three of the transformation primitives, specifically the message element setter, the database lookup and the set message type 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_XFormContentPrimitives.ppt

This module is also available in PDF format at: ../WBPMv7_XFormContentPrimitives.pdf

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

Trademarks, disclaimer, and copyright information