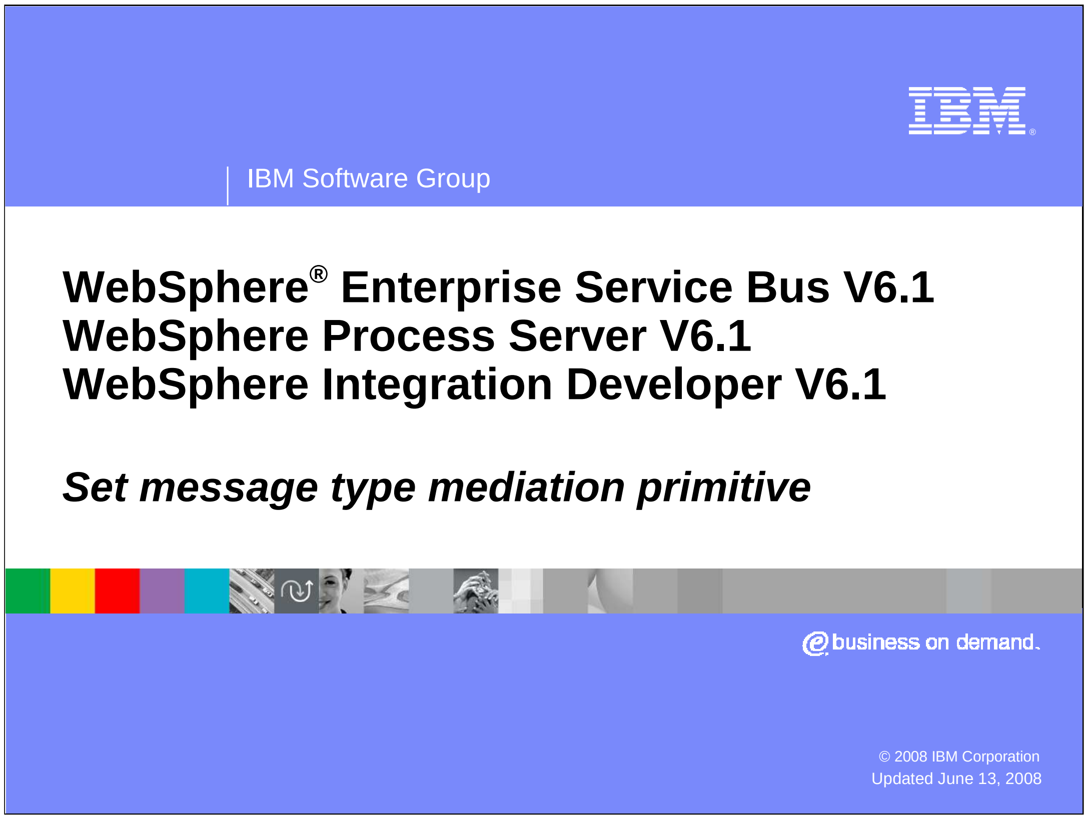


IBM Software Group

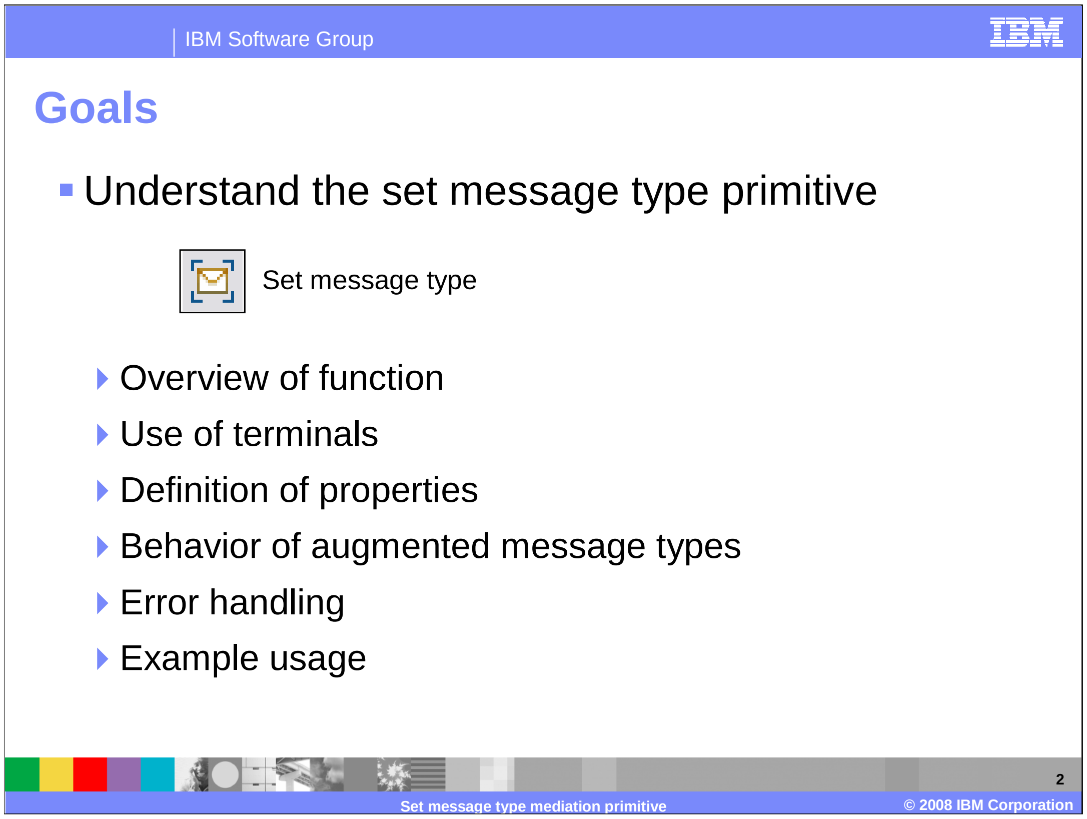
WebSphere® Enterprise Service Bus V6.1 WebSphere Process Server V6.1 WebSphere Integration Developer V6.1

Set message type mediation primitive

business on demand.

Updated June 13, 2008

This presentation provides a detailed look at the set message type mediation primitive.



Goals

- Understand the set message type primitive

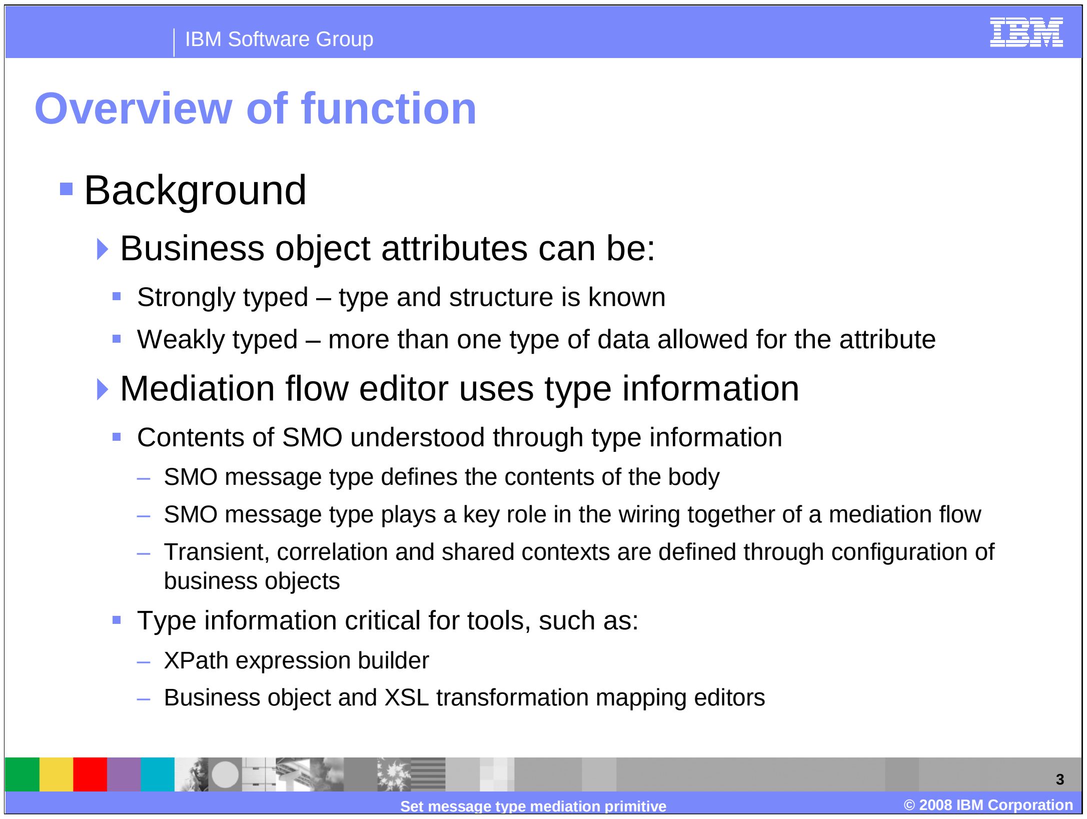
  Set message type

  - Overview of function
  - Use of terminals
  - Definition of properties
  - Behavior of augmented message types
  - Error handling
  - Example usage

The goal of this presentation is to provide you with a full understanding of the set message type mediation 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 set message type primitive is provided followed by a description of its terminals and properties. The set message type primitive augments a message type with additional type information. The resulting behavior of the augmented message type is slightly different from the general behavior attributed to message types, and these differences are explained. The presentation then provides some error handling information and concludes with an example, showing a possible use of this primitive.



IBM Software Group

Overview of function

- Background
  - Business object attributes can be:
    - Strongly typed – type and structure is known
    - Weakly typed – more than one type of data allowed for the attribute
  - 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 XSL transformation mapping editors

3

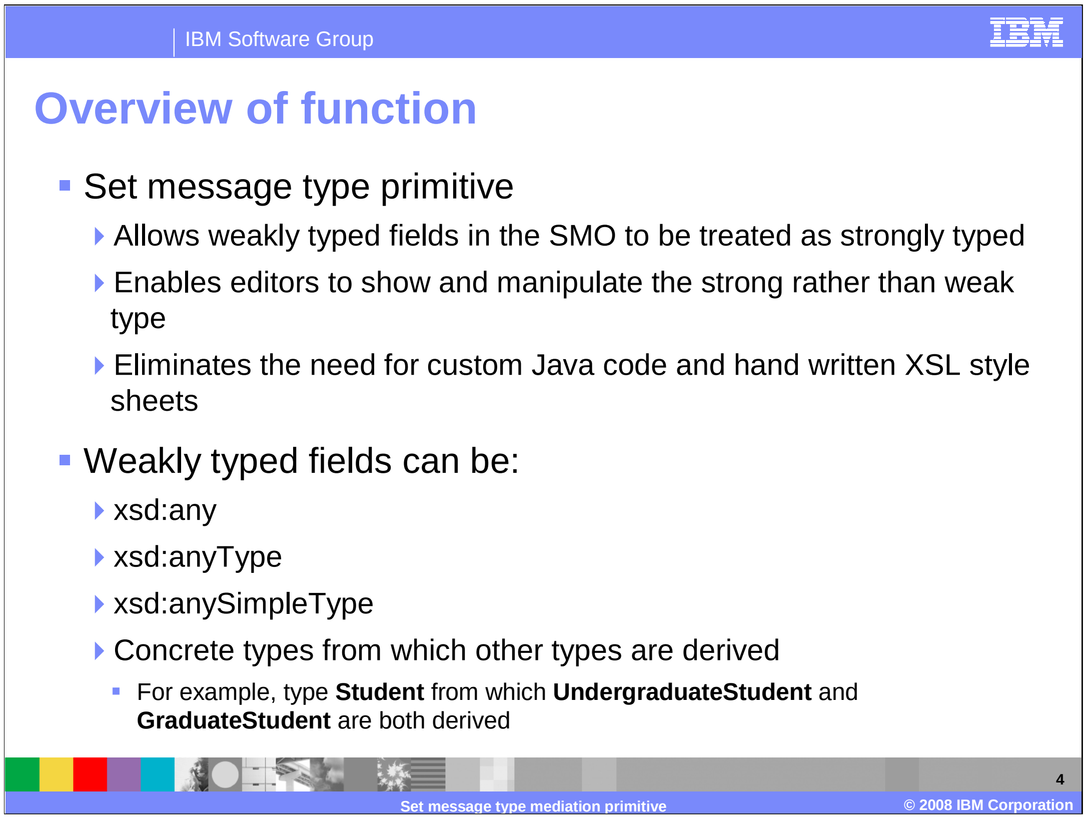
Set message type mediation primitive

© 2008 IBM Corporation

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 attributes in business objects. Some attributes are strongly typed, so that the exact type and structure of data contained in the attribute is well known. However, some attributes are weakly typed, allowing different types and structures of data to exist for that attribute.

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 (SMO) 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 attributes. This affects tools, such as the XPath expression builder, the business object map editor and the XSL transformation map editor, as they are highly dependent upon type information for the SMO context, headers and body.


IBM Software Group

Overview of function

- Set message type primitive
  - 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

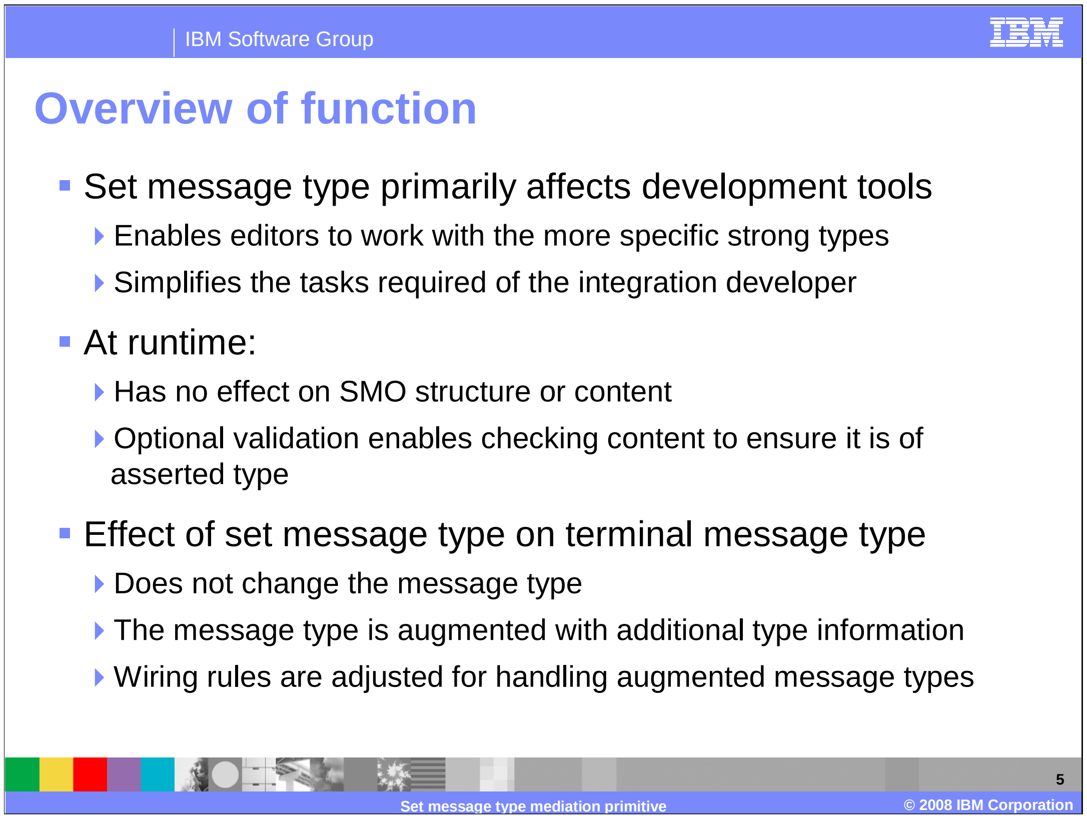
4

Set message type mediation primitive

© 2008 IBM Corporation

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.


IBM Software Group

Overview of function

- 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

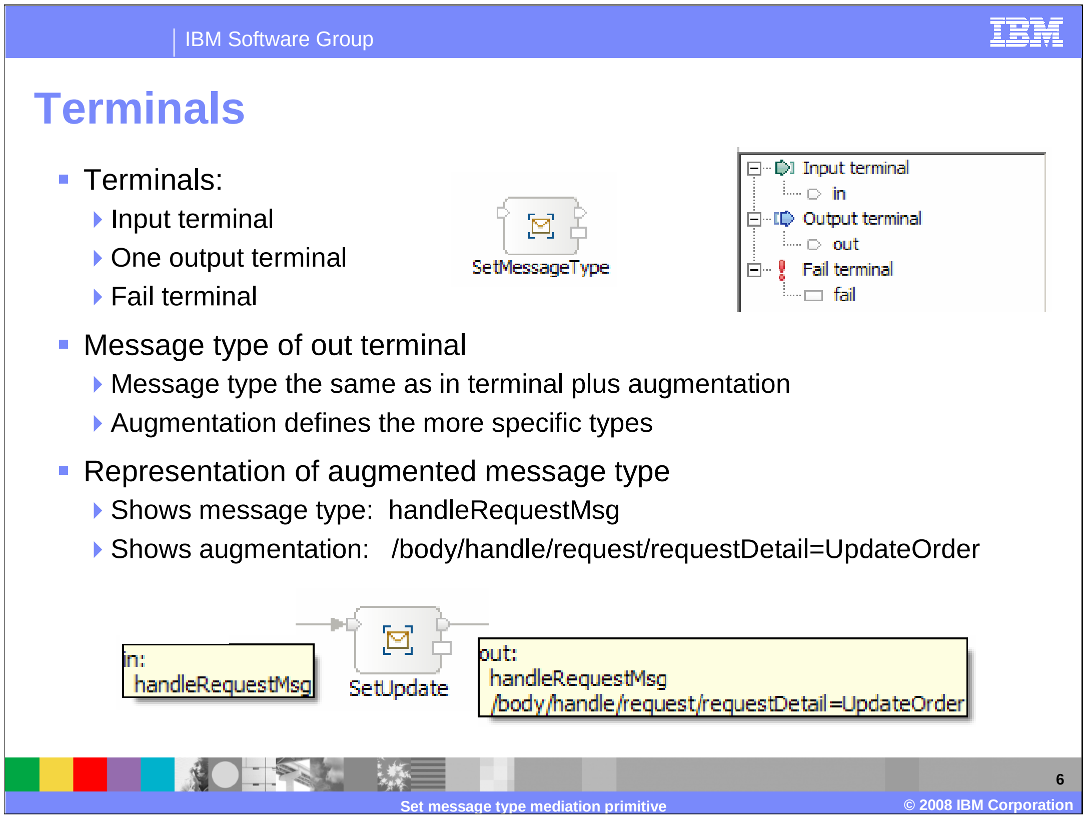
5

Set message type mediation primitive

© 2008 IBM Corporation

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.

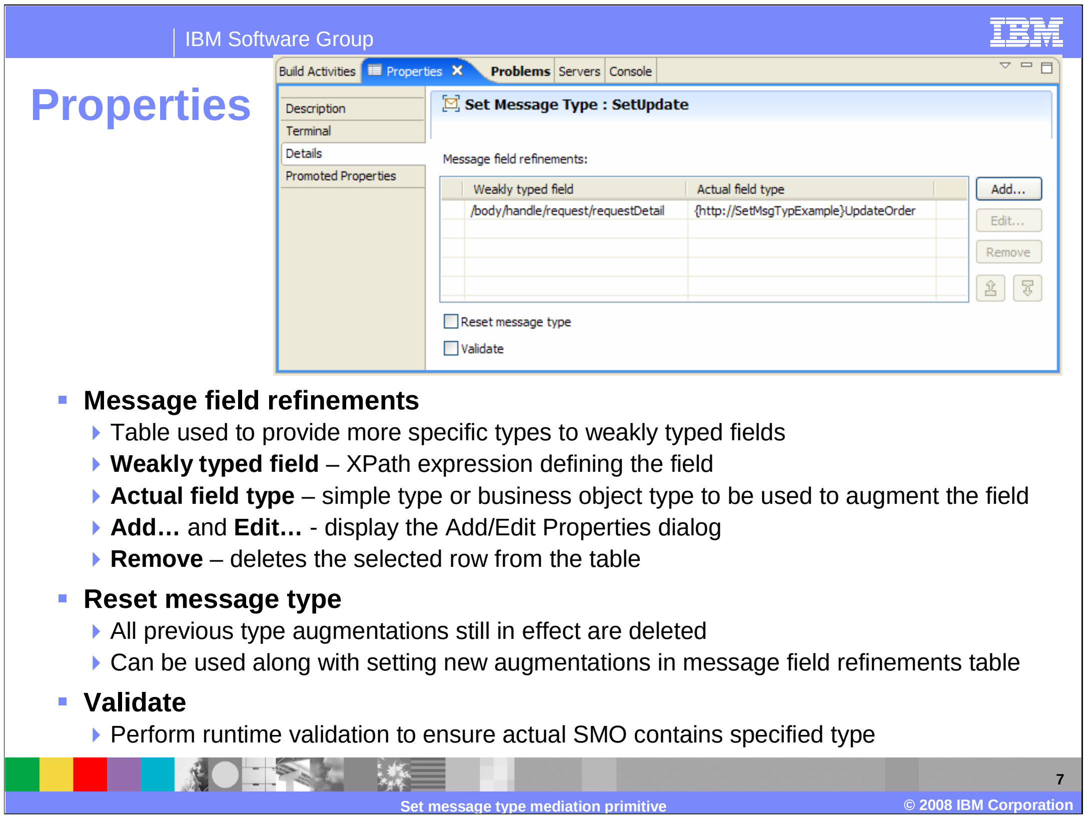
One last point about the function of this primitive is that it does not actually change the message type of the SMO that gets associated with a terminal. Rather, it augments the message type with additional type information. This results in some differences in the wiring rules for terminals and the affect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



Terminals

- Terminals:
  - Input terminal
  - One output terminal
  - Fail terminal
- Message type of out terminal
  - Message type the same as in terminal plus augmentation
  - Augmentation defines the more specific types
- Representation of augmented message type
  - Shows message type: handleRequestMsg
  - Shows augmentation: /body/handle/request/requestDetail=UpdateOrder

The set message type primitive has an input terminal, one output terminal and a fail terminal. This can be seen at the top of the slide in the screen capture of a set message type primitive and the terminals panel from the properties view. The message type of the out terminal is the same as the message type of the in terminal, with additional type augmentation resulting from the declaration of the more specific types. At the bottom of the slide you can see a screen capture that shows a set message type primitive along with the message type of its in terminal and the augmented message type of the out terminal. You can see that the message type, handleRequestMsg has not changed, but that the field at /body/handle/request/requestDetail has been declared to be of type UpdateOrder.



IBM Software Group

Properties

Build Activities | Properties | Problems | Servers | Console

Description
Terminal
Details
Promoted Properties

Set Message Type : SetUpdate

Message field refinements:

Weakly typed field	Actual field type
/body/handle/request/requestDetail	{http://SetMsgTypExample}UpdateOrder

Add... Edit... Remove

☐ Reset message type

☐ Validate

- **Message field refinements**
  - Table used to provide more specific types to weakly typed fields
  - **Weakly typed field** – XPath expression defining the field
  - **Actual field type** – simple type or business object type to be used to augment the field
  - **Add…** and **Edit…** - display the Add/Edit Properties dialog
  - **Remove** – deletes the selected row from the table
- **Reset message type**
  - All previous type augmentations still in effect are deleted
  - Can be used along with setting new augmentations in message field refinements table
- **Validate**
  - Perform runtime validation to ensure actual SMO contains specified type

7

Set message type mediation primitive

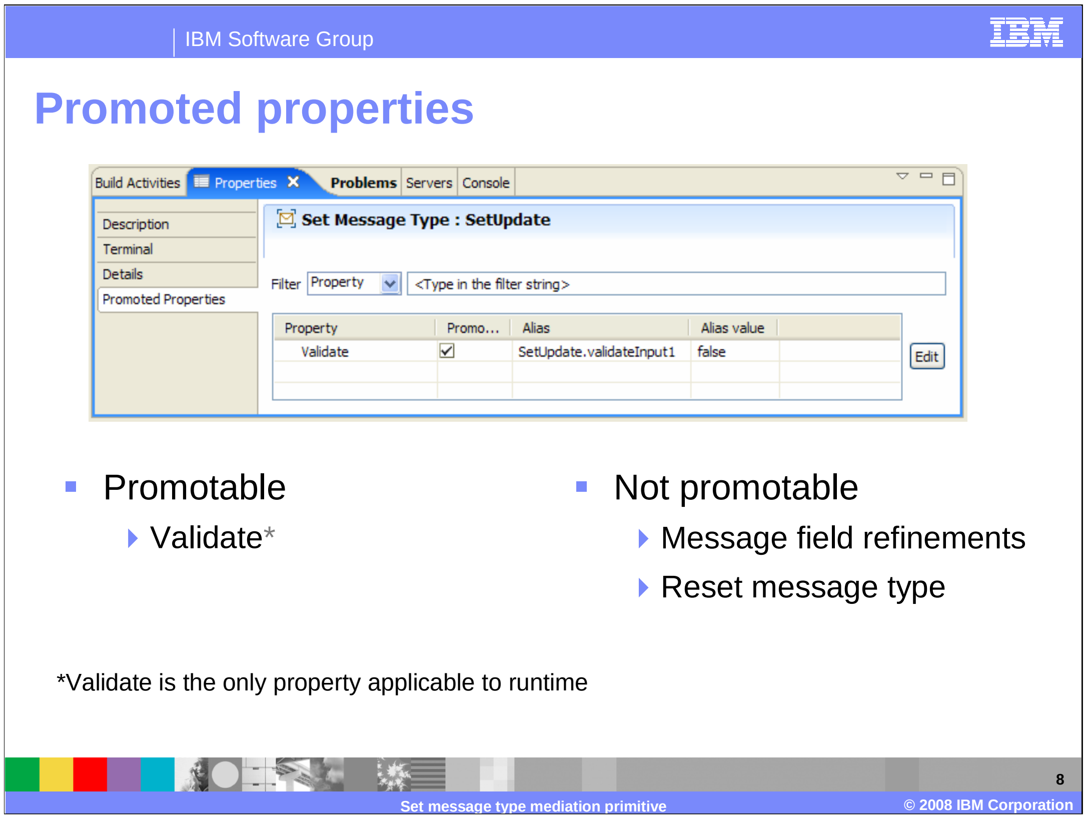
© 2008 IBM Corporation

In the upper right corner of the slide is a screen capture showing the Details panel of the Properties view for a set message type primitive, showing the primitive's three properties.

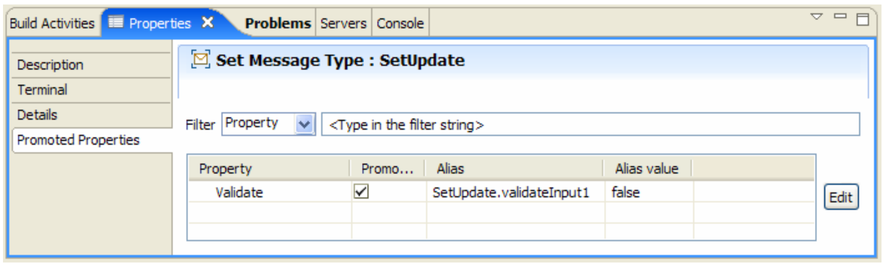
The **Message field refinements** property is a table allowing you to declare more specific types for weakly typed fields. Because it is a table, you are able to declare stronger types for multiple fields in the SMO. The table has two columns, the **Weakly typed field** column contains an XPath expression identifying the weakly typed field in the SMO. The **Actual field type** column is used to declare the more specific type to be associated with the field. It can be either a simple type or a complex type defined by a business object. The **Add…** and **Edit…** buttons are used to open the add and edit properties dialog. This dialog enables you to use the XPath expression builder to construct the XPath for the weakly typed field and the data type selection dialog for selecting the actual field type. The **Remove** button can be used to remove the selected row from the table.

The **Reset message type** property is used to remove all previous augmentations that are still in effect. When this is not checked, the resulting message type includes the existing augmentations plus the refinements added by this primitive. When this is checked, the resulting message type only includes the refinements you defined in this primitive.

Finally, the **Validate** property is used to request runtime validation. It is only when this property is selected that this primitive has any effect on the runtime behavior. It requests that the SMO be checked to ensure that the weakly typed field in the SMO does indeed contain the declared stronger type. If it does not, a MediationBusinessException occurs.



Promoted properties



Set Message Type : SetUpdate

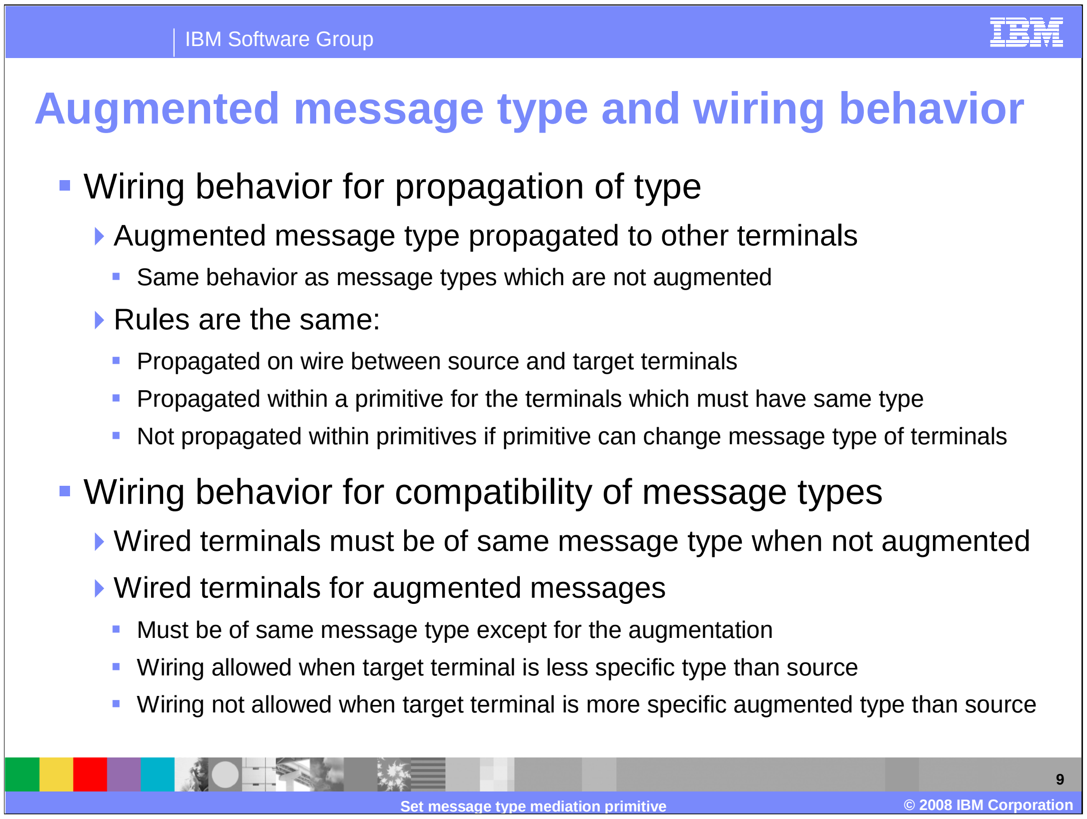
Filter Property <Type in the filter string>

Property	Promo...	Alias	Alias value
Validate	✓	SetUpdate.validateInput1	false

- Promotable
  - Validate*
- Not promotable
  - Message field refinements
  - Reset message type

*Validate is the only property applicable to runtime

This slide shows the Promoted Properties panel for the set message type primitive. Since the validate property is the only property that affects the runtime environment, it is the only property that is promotable. Therefore, the message field refinements table and reset message type properties are not promotable.


IBM Software Group

Augmented message type and wiring behavior

- Wiring behavior for propagation of type
  - Augmented message type propagated to other terminals
    - Same behavior as message types which are not augmented
  - Rules are the same:
    - Propagated on wire between source and target terminals
    - Propagated within a primitive for the terminals which must have same type
    - Not propagated within primitives if primitive can change message type of terminals
- Wiring behavior for compatibility of message types
  - Wired terminals must be of same message type when not augmented
  - Wired terminals for augmented messages
    - Must be of same message type except for the augmentation
    - Wiring allowed when target terminal is less specific type than source
    - Wiring not allowed when target terminal is more specific augmented type than source

9

Set message type mediation primitive

© 2008 IBM Corporation

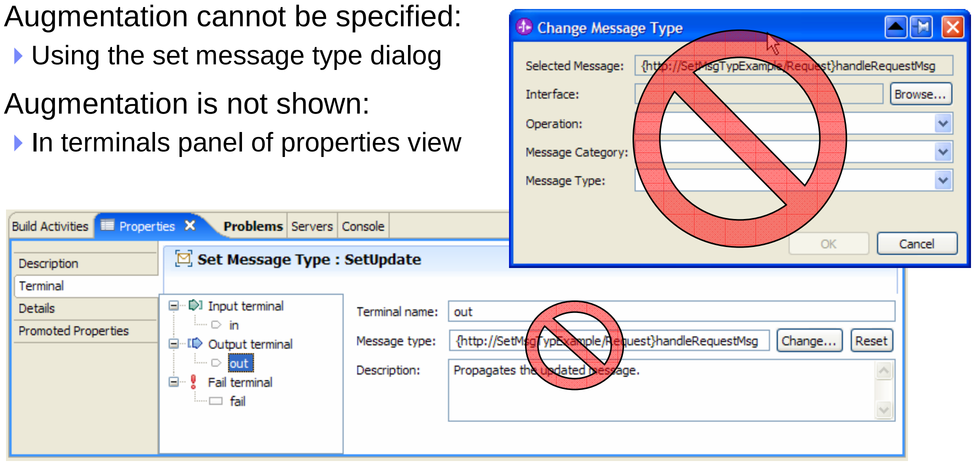
When considering the wiring of mediation primitives to construct a flow, there are two aspects that must be considered. This slide addresses these relative to the handling of augmented message types.

The first of these is the propagation of message type to other terminals as part of the wiring process. The propagation behavior associated with a type augmentation is the same as the behavior for the message type. Basically, the rules are that propagation occurs on a wire between source and target terminals, in either direction. Within a primitive, propagation occurs between terminals which must have the same message type. Finally, no propagation occurs within a primitive’s terminals when they can have different message types.

The second aspect to consider is the rules for compatibility between message types of terminals that are wired together. For message types that are not augmented, they must always be of the same message type to be wired together. For augmented message types, there is a variation to how this works. Other than the augmentation, the message types of the two terminals must be the same. However, when there is an augmentation, if the augmentation is not the same, the wiring is still allowed if the target terminal of the wire is of a less specific type than the source.

Specification of message type

- Augmentation can only be specified when:
  - Using set message type primitive
  - Using fan out primitive configured for iteration (augmentation is implicit)
- Augmentation cannot be specified:
  - Using the set message type dialog
- Augmentation is not shown:
  - In terminals panel of properties view

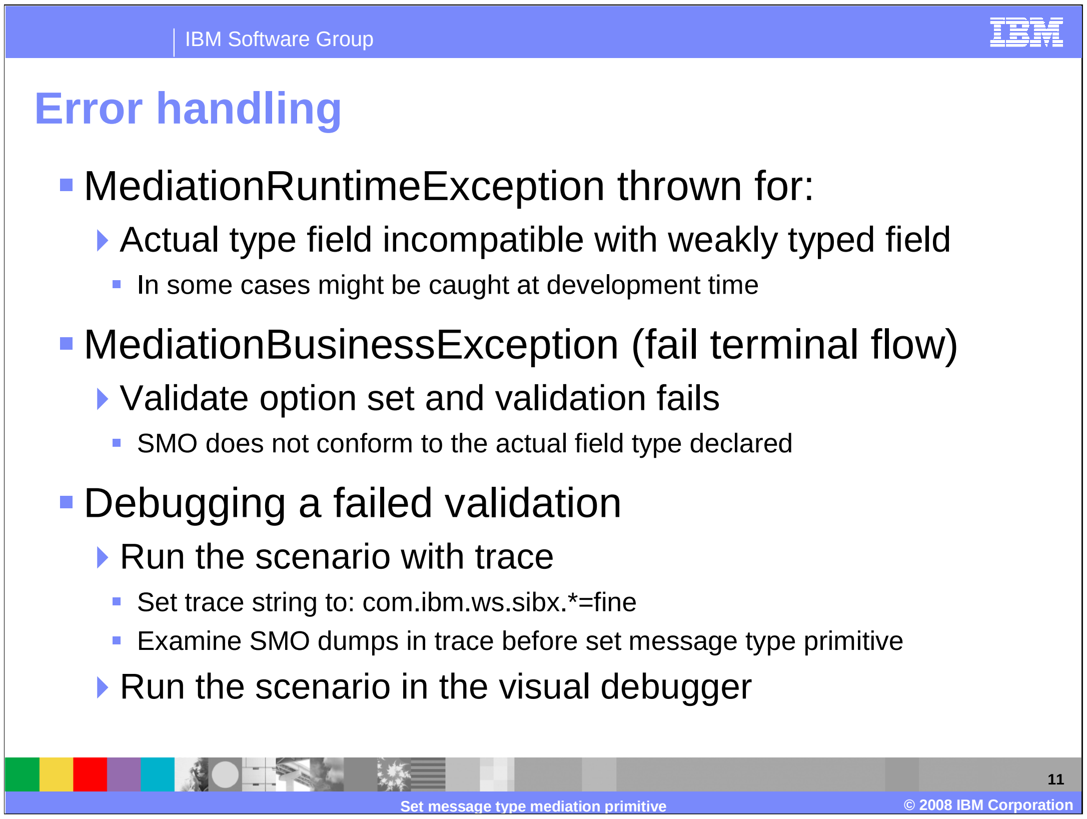


Selected Message: {http://SetMsgTypExample/Request}handleRequestMsg
Interface:
Operation:
Message Category:
Message Type:

Terminal name: out
Message type: {http://SetMsgTypExample/Request}handleRequestMsg
Description: Propagates the updated message.

The augmentation of message types does not completely follow the model for message types and how they are specified. There are only two ways that an augmentation can be added to a message type. The first is the use of the set message type primitive which is discussed in this presentation. The second happens implicitly, when a fan out primitive is configured in iterate mode. In this case, the fan out context is augmented to the type of the array element being iterated.

Other mechanisms for setting and displaying message type do not include message type augmentation information. Specifically, the change message type dialog does not allow you to include message type augmentation when setting the message type. Also, the terminal panel of the properties view for a primitive does not display the augmentation information for the terminal’s message type. .



Error handling

- MediationRuntimeException thrown for:
  - Actual type field incompatible with weakly typed field
    - In some cases might be caught at development time
- MediationBusinessException (fail terminal flow)
  - Validate option set and validation fails
    - SMO does not conform to the actual field type declared
- Debugging a failed validation
  - Run the scenario with trace
    - Set trace string to: com.ibm.ws.sibx.*=fine
    - Examine SMO dumps in trace before set message type primitive
  - Run the scenario in the visual debugger

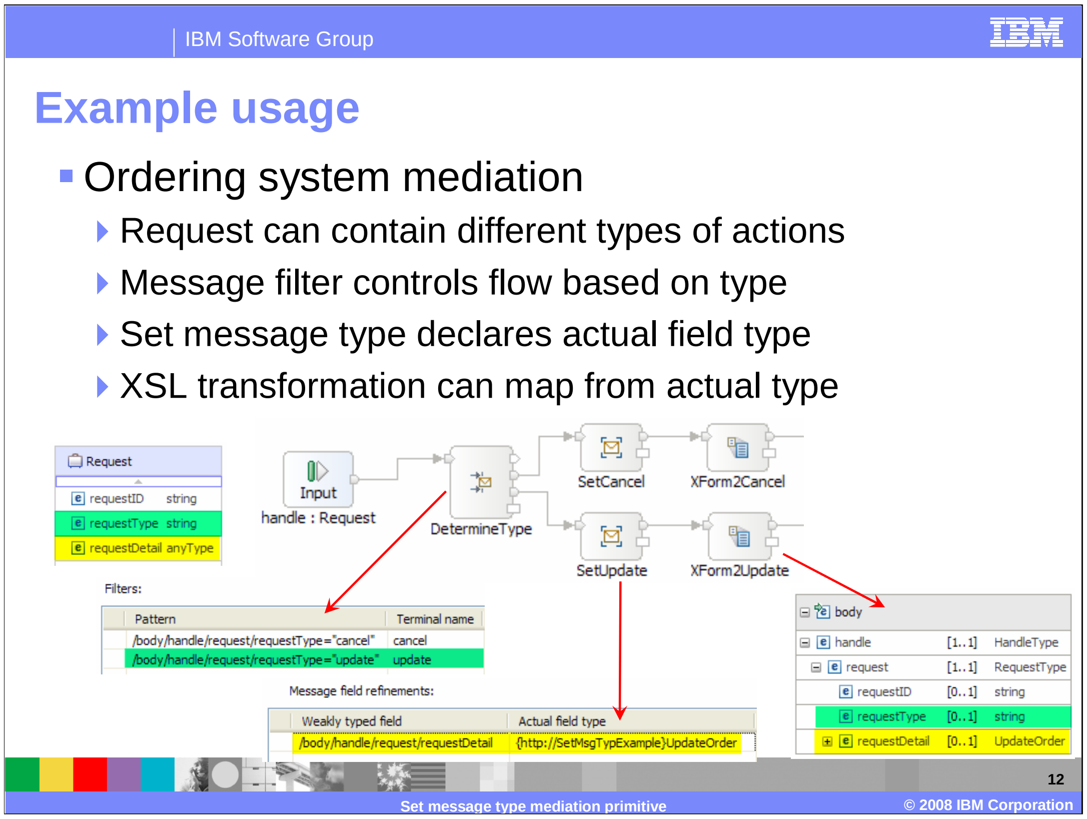
This slide addresses some of the error situations you might encounter.

The first of these is when you declare a type in a set message type primitive which is not compatible with the original weakly typed field, or possibly the original field isn't even a weak type. For example, suppose you use a set message type primitive to declare that a field in the SMO that is a string is actually an Order object. This results in the throwing of a MediationRuntimeException. This type of error, in some cases, might be flagged by the development tools as not being a valid refinement. Unfortunately, some of these cases are not caught until runtime.

The next error you might see is when you use the validate property and at runtime the SMO does not actually contain the declared type. This results in a MediationBusinessException. When you make use of the validate property, it is likely that you will also include a flow from the fail terminal of the set message type primitive to handle this condition.

When encountering a situation where the SMO does not contain the declared type, it might be useful to use trace to debug the problem. This can be done by setting the trace string to com.ibm.ws.sibx.*=fine, then running the failing scenario and examining the trace file. There are dumps of the SMO before and after every primitive in the flow. These can help you to understand what type the SMO contains that is causing the validation to fail, and hopefully why the SMO contains the incorrect type.

Alternatively, you can run the scenario using the visual debugger and examine the contents of the SMO as it progresses through the flow.



Example usage

- Ordering system mediation
  - Request can contain different types of actions
  - Message filter controls flow based on type
  - Set message type declares actual field type
  - XSL transformation can map from actual type

Filters:

Pattern	Terminal name
/body/handle/request/requestType="cancel"	cancel
/body/handle/request/requestType="update"	update

Message field refinements:

Weakly typed field	Actual field type
/body/handle/request/requestDetail	{http://SetMsgTypExample}UpdateOrder

body		
handle	[1..1]	HandleType
request	[1..1]	RequestType
requestID	[0..1]	string
requestType	[0..1]	string
requestDetail	[0..1]	UpdateOrder

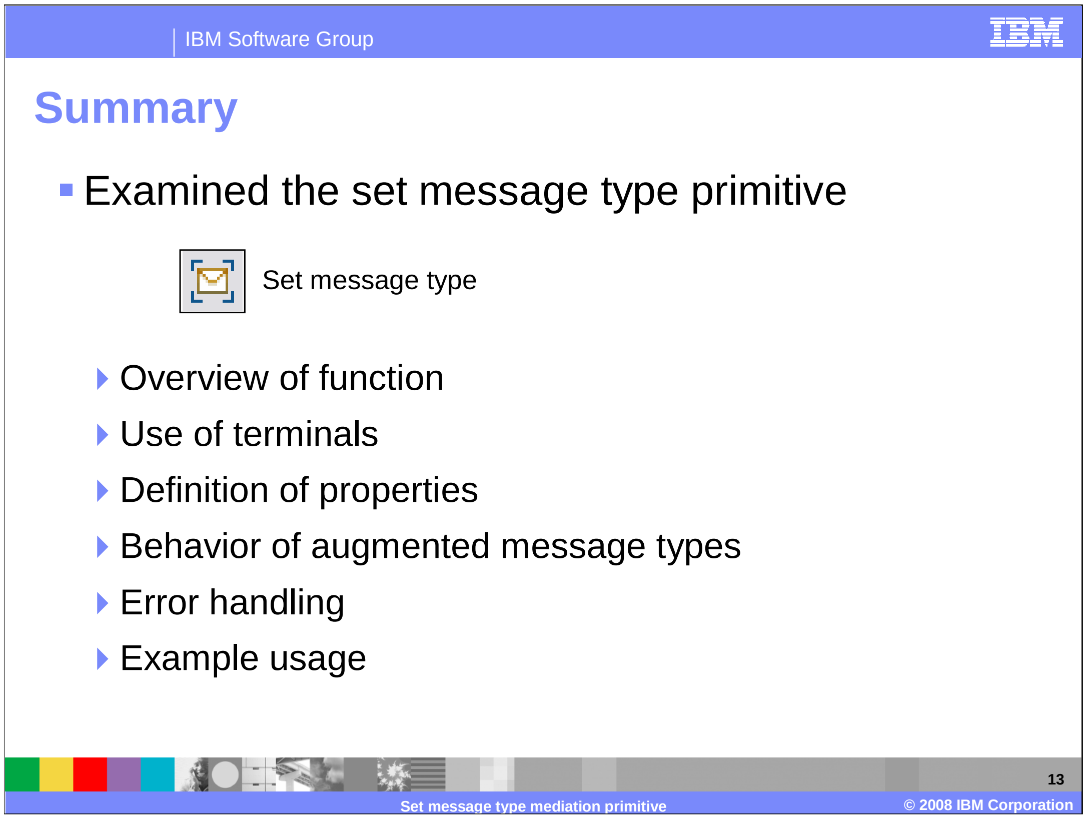
This slide illustrates a possible use of the set message type primitive using an ordering system mediation as an example. In this example, various types of requests can be sent to the ordering mediation, where each request type has unique detailed data. The mediation determines what type of request it is, transforms the request data, and passes it on to the appropriate back end system.

The requests come into the mediation as a Request business object which you see in the upper left of the screen captures. There is a requestType field that contains a string identifying what type of request it is. There is also a requestDetail field, defined as an xsd:anyType, which holds a business object of a type defined by the type field. For simplicity of the example, the flow only handles two types of requests, cancel and update.

The first thing in the flow is a message filter primitive, labeled DetermineType, which propagates the message through an output terminal based on the value of the requestType field. The properties to do this are shown in the slide.

The next thing in the flow is two set message type primitives, one on each flow path. The path for update is illustrated, with the message field refinements property for the SetUpdate primitive showing. It takes the requestDetail field, and declares the actual field type to be of type UpdateOrder.

The next primitive on the flow path for update is an XSL transformation, labeled XForm2Update. The source SMO for this transformation is shown, and you can see that the requestDetail field is an UpdateOrder rather than an xsd:anyType. This allows the map to be created by expanding UpdateOrder and mapping from the fields it contains.



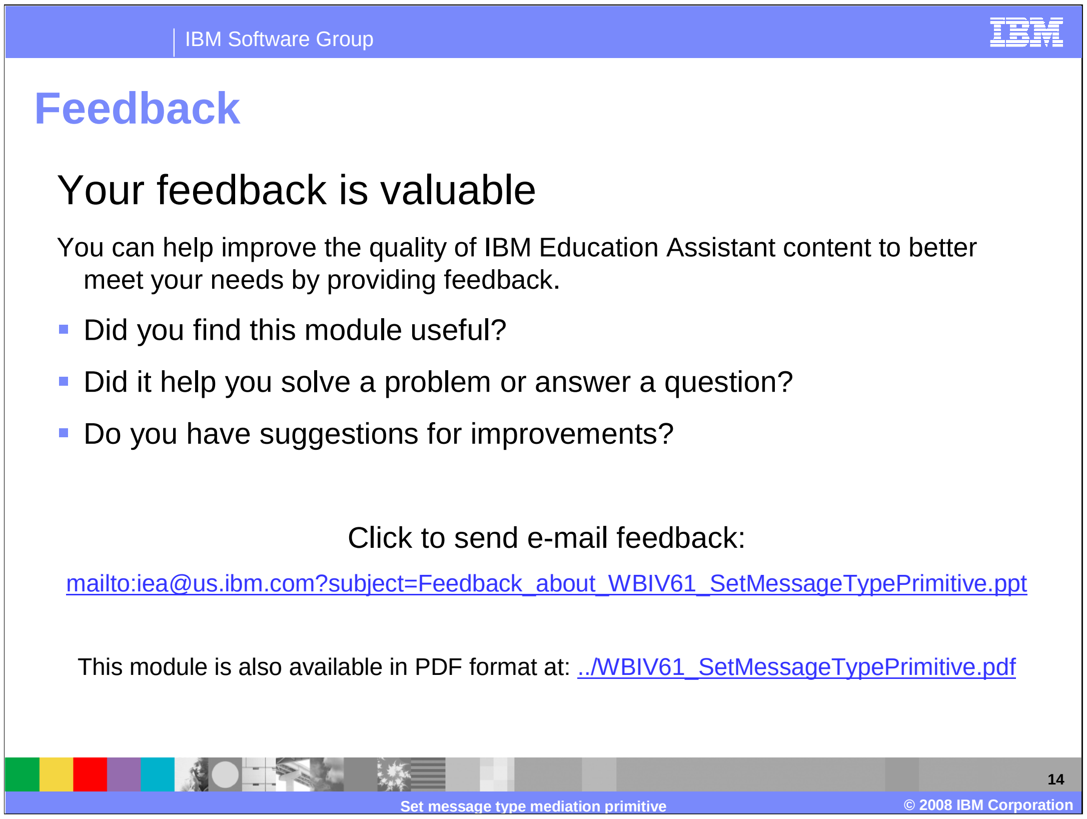
Summary

- Examined the set message type primitive

  Set message type

  - Overview of function
  - Use of terminals
  - Definition of properties
  - Behavior of augmented message types
  - Error handling
  - Example usage

In summary, this presentation provided details regarding the set message type primitive, starting with an overview of the primitive followed by a description of its terminals and properties. The behavior associated with augmented message types was explained and contrasted with typical behavior associated with message types. The presentation also provided some error handling information and concluded with an example, showing a possible use of this 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.

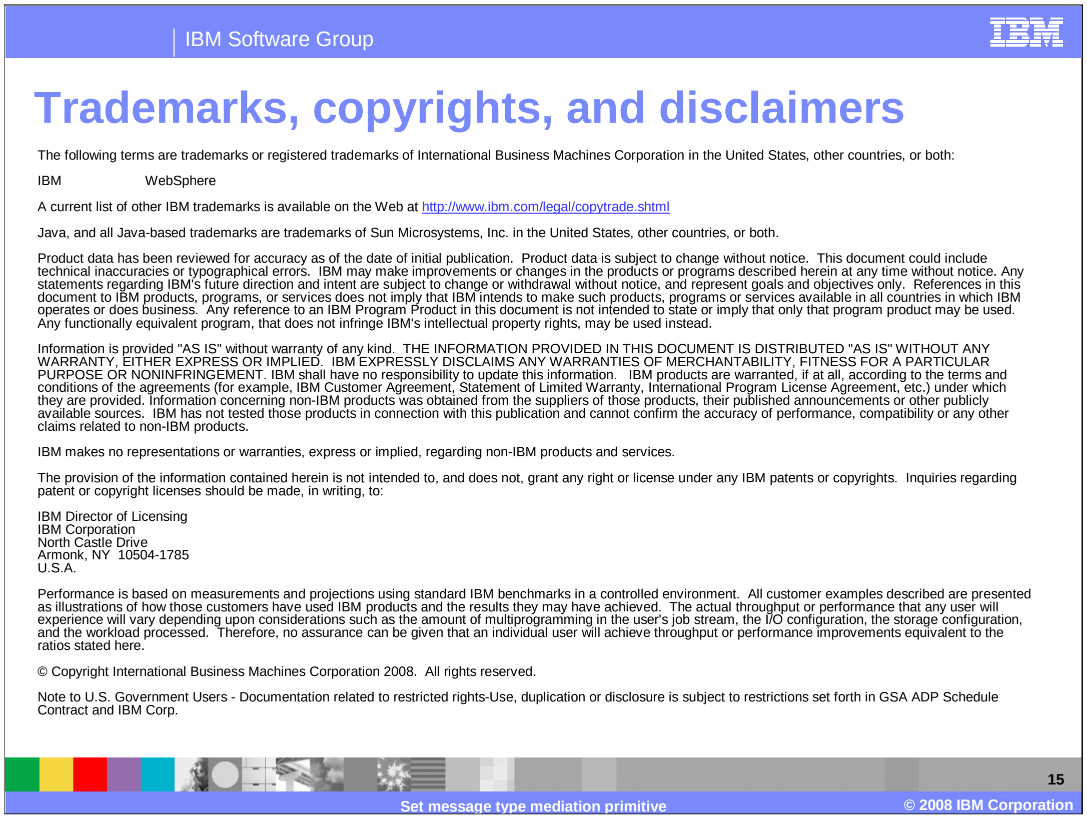
- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send e-mail feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_WBIV61_SetMessageTypePrimitive.ppt

This module is also available in PDF format at: ../WBIV61_SetMessageTypePrimitive.pdf

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



Trademarks, copyrights, and disclaimers