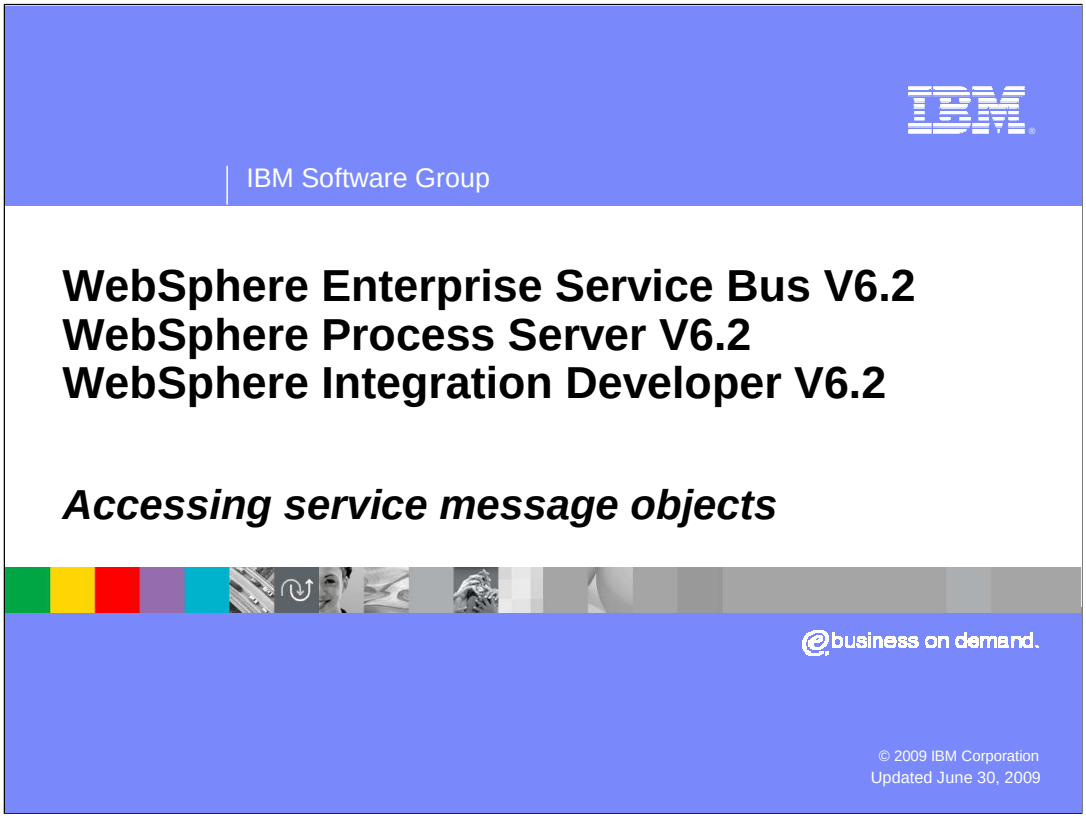


This presentation examines how service message objects are accessed from within a mediation flow.

Goals

- Understand the accessing of service message objects (SMO)
  - Access and manipulation of SMO content
  - Manipulation of message type
- Approaches used
  - XPath
  - XSL transformations (XSL Style sheets)
  - Business objects maps
  - Data handlers
  - Java™ code

The goal of this presentation is to provide you with an understanding of how to access service message objects (SMO). Accessing SMOs occurs within mediation flows and involves the reading, writing and updating of the elements of an SMO. Updating an SMO might also include modifying the message type. This occurs when the structure of the message payload, also known as the body of the message, is changed. Each of the mechanisms for accessing and manipulating service message objects is covered, which includes the use of XPath expressions, XSL transformations using XSL style sheets, business object maps, data handlers and Java code.

Manipulating SMOs

- XPath 1.0 expressions
  - Primary mechanism for accessing the SMO
  - Used in some form by many of the mediation primitives
  - Identify elements to read, update or process conditional expressions
- XSL style sheets
  - Underlying mechanism of the XSL transformation mediation primitive
  - Typically used to modify SMO message type within a flow
- Business object maps
  - Underlying mechanism of the business object map mediation primitive
  - Typically used to modify SMO message type within a flow
- Data handlers
  - Underlying mechanism of the data handler mediation primitive
  - Typically used to modify SMO message type within a flow
- Java code
  - Used by the Custom Mediation primitive
  - SMO APIs and DataObject APIs
  - Used to access and update SMO content and can also modify SMO message type

This slide provides an overview of the different mechanisms for accessing and manipulating the contents of an SMO. XPath 1.0 is the primary mechanism for accessing SMOs and is employed in one form or another by almost all of the mediation primitives. XPath expressions can be used to identify elements of the SMO to read or update. They can also be conditional expressions to be evaluated. XSL style sheets are used by the XSL transformation primitive, which is typically used to modify the message type within the flow by changing the structure of the body of the message. Business object maps are used by the business object map primitive. Similar to the XSL style sheets, the typical use of a business object map is to modify the SMO message type within a flow. Data handlers, normally used by SCA imports and exports, are used by the data handler mediation primitive. These are typically used in service gateway scenarios to convert to or from native data in the SMO. In most cases, this results in changing the SMO message type. Java code is used by the custom mediation primitive to access the SMO. The generic DataObject APIs can be used, making use of XPath expressions to identify properties within the SMO. There are also SMO specific APIs, which provide type safe access to the properties within the SMO. Using Java code, you can read and update the message content and you also have the ability to modify the message type by changing the structure of the message body. In the next series of slides, each of these mechanisms is examined in detail.

Section

XPath

This section takes a closer look at the use of XPath for accessing SMOs.

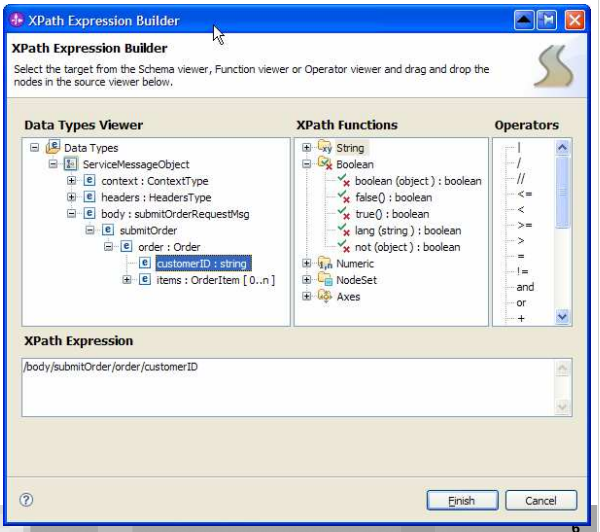
XPath

- Most mediation primitives use XPath in some form
  - XPath expression identifying a specific property within the SMO
  - XPath conditional expression to be evaluated
- Root property
  - Used to specify what part of the SMO is visible to the primitive
  - Values selected from a drop down:
    - Typically the values are: **/ /body /context /headers**
    - Some instances of root are more restrictive and some are less restrictive
- Other properties using XPath
  - Specified using XPath expression builder dialog
  - Content assist is also provided

XPath expressions are used in some form by most of the mediation primitives and are typically a qualified path based on the SMO schema that identifies a property within the SMO. The property identified might be a complex DataObject type or a simple element with a primitive type. At other times the XPath expression might be a conditional expression to be evaluated true or false relative to the value of an element in the SMO. There are generally two different ways that an XPath expression is specified within the properties for a mediation primitive. The first is the use of a root property, which identifies that portion of the SMO that is to be visible to or used by the primitive. In most cases, the root property consists of four possible values that are selected using a drop down box. The value / (forward slash) is used to indicate the entire SMO and /body indicates the body or payload of the SMO. The value /context refers to the context section of the SMO and /headers refers to the headers contained in the SMO. However, the use of the root property is not always exactly like this. Some cases are more restrictive and only provide two possible choices and other cases allow the use of an XPath expression that can identify any property within the SMO. Other uses of XPath by mediation primitives make use of the XPath expression builder dialog. The dialog allows you to traverse the SMO schema to select a particular property or element within the SMO. It enables you to build up an expression using XPath functions and operators. Input fields for XPath expressions are also enabled for content assist. This provides similar capabilities to that of the XPath expression builder dialog, but through the use of content assist proposal lists.

XPath expression builder dialog

- Contains three source viewers
  - Data types viewer
    - Shows schema elements and attributes available to use
    - Shown collapsed, with twisties to open sections and drill down
  - XPath functions
    - Shows top level groups with twisties to show functions
  - Operators
    - Expanded list of available operators
- XPath expression
  - View where expression is constructed
  - Built by obtaining appropriate elements from source viewers
    - Drag-and-drop
    - Double click
  - Can type directly into the field
- Launched using:
  - **Edit…** button for XPath property field
  - **…** button in an XPath table cell



This slide examines the XPath expression builder dialog. The dialog has four panes, three for making selections, and the fourth where the XPath expression is constructed.

The data type viewer shows the schema for the SMO in a collapsed form. You can expand the schema elements selectively and drill down to the element that you want to include in the expression.

The next pane lets you select XPath functions to be used in the expression. These functions are grouped by category.

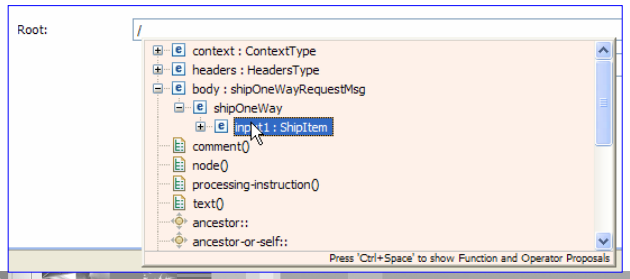
The third pane is for the operators, providing an expanded list of the XPath operators that you can select for building the expression.

At the bottom of the slide is the entry field for the XPath expression being constructed. You can drag elements from any of the three panes and drop them into the expression being built. Double clicking on a schema element, function or operator in a pane adds that element to the expression. You can also type directly into the field.

This dialog is launched by using the Edit… button found next to text entry fields requiring an XPath expression. In a table, a cell that takes an XPath expression has a … button to access this dialog.

XPath content assist

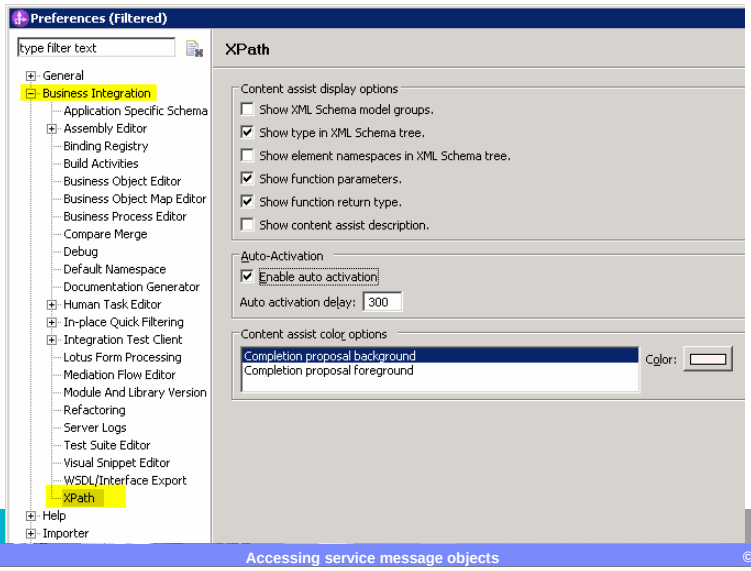
- XPath content assist enabled fields
  - Accessed using ctrl-space
  - If auto activation enabled, entry of symbol: **/ ([,**
- Two different proposal lists
  - Node and variable
  - Functions and operator
  - ctrl-space to toggle lists – status bar indicates this can be done



Fields for XPath expressions are normally enabled for content assist. Similar to other content assist enabled entry fields in WebSphere® Integration Developer, the context assist proposal lists are accessed using the ctrl-space key combination. Auto activation is a user selectable option. When activated, the symbols “/” (forward slash), “(“ (left parenthesis), “[“ (left square bracket) and “,” (comma) will cause the proposal list to be displayed. There are two proposal lists. The first is the node and variable list, where you can select schema elements from the SMO similar to the data types viewer in the XPath expression builder dialog. The other proposal list provides the functions and operators. The ctrl-space key combination can be used to toggle back and forth between the two proposal lists. The status bar at the bottom of the list provides a prompt to remind you that you can toggle between the lists.

XPath content assist configuration

- Content assist support is configurable
  - Window → Preferences… → Business Integration → XPath



The use of content assist for XPath expressions is configurable through a preferences dialog. To access the panel, select Window, Preferences… from the menu bar and then select Business Integration, XPath from the left side of the preferences dialog. The configuration parameters available to you are shown in the screen capture.

Section

XSL transformation (XSL style sheets)

This section takes a closer look at the use of XSL transformations, which use XSL style sheets for accessing SMOs.

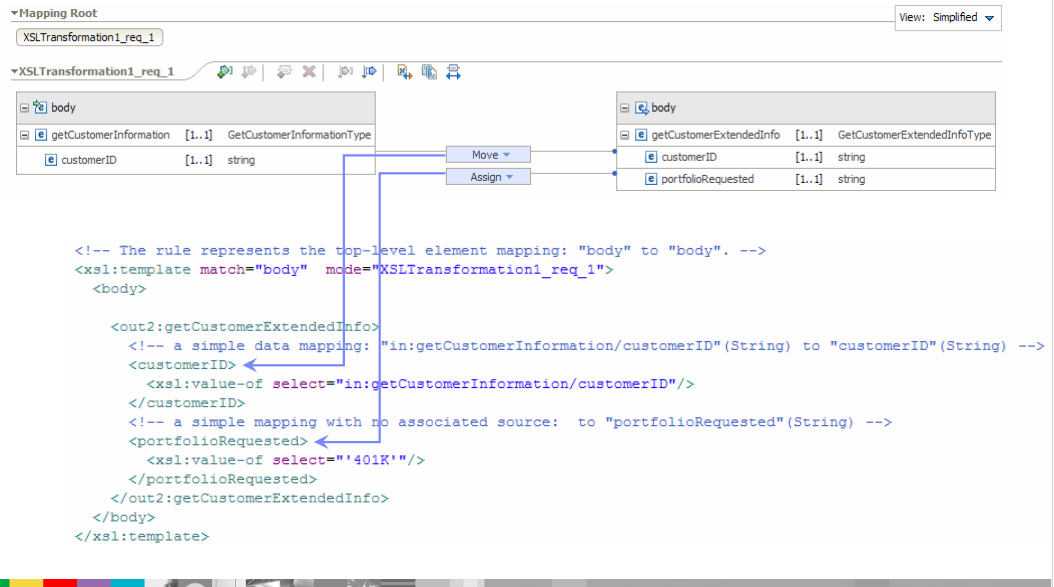
XSL style sheets

- XSL transformation primitives process XSL style sheets at runtime
- An XSL style sheet used by the XSL transformation primitive can be:
  - Generated from an XML map created with the XML mapping editor
  - Created directly using the XSL editor
- Typical use is for modification of message type in a flow
  - When input and callout nodes have different message types
  - To reply using the input response node in a request flow
  - Before or after service invoke primitives
  - To reply with a fault using the input fault node to report a flow error

The XSL transformation primitives use XSL style sheets at runtime to manipulate the SMO. There are two ways to define an XSL style sheet. The first is to use the XML mapping editor to define a mapping, which is then used to generate an XSL style sheet. The second approach is to the use the XSL editor to edit the style sheet directly. See the XSL transformation primitive presentation for information about the XML editor and the specification of an XML map versus an XSL style sheet.

Typically, the XSL style sheet is used within a mediation flow to modify the message type. There are many different reasons why this might need to be done. It is required when a mediation flow has an input node and a callout node that have terminals for different message types. Also, if the input response node is going to be used in the request flow, the request message must be transformed into a response message. Use of the service invoke primitive normally requires changing the message type before and after its use in the flow, as its message types are dictated by the external service being called. Another possible instance requiring changing the message type is when an error is detected in the request flow and the message has to be transformed to a fault message to be returned using the input fault node.

XML mapping editor and XSL style sheet



This slide shows a screen capture of the XML mapping editor with a simple XML map of a message body. It also shows the resulting segment of the XSL style sheet. The mapping is from a getCustomerInformation operation to a getCustomerExtendedInfo operation. Each operation contains a customer ID field, which is moved from source to target. The target also has a portfolio requested field, which is set by the map to a default value of “401K”. At the bottom is a screen capture of that portion of the XSL style sheet that reflects this mapping. The arrows from the map to the style sheet show the generated move and the generated assign.

Section

Business object maps

This section takes a look at how business object maps are used for accessing SMOs.

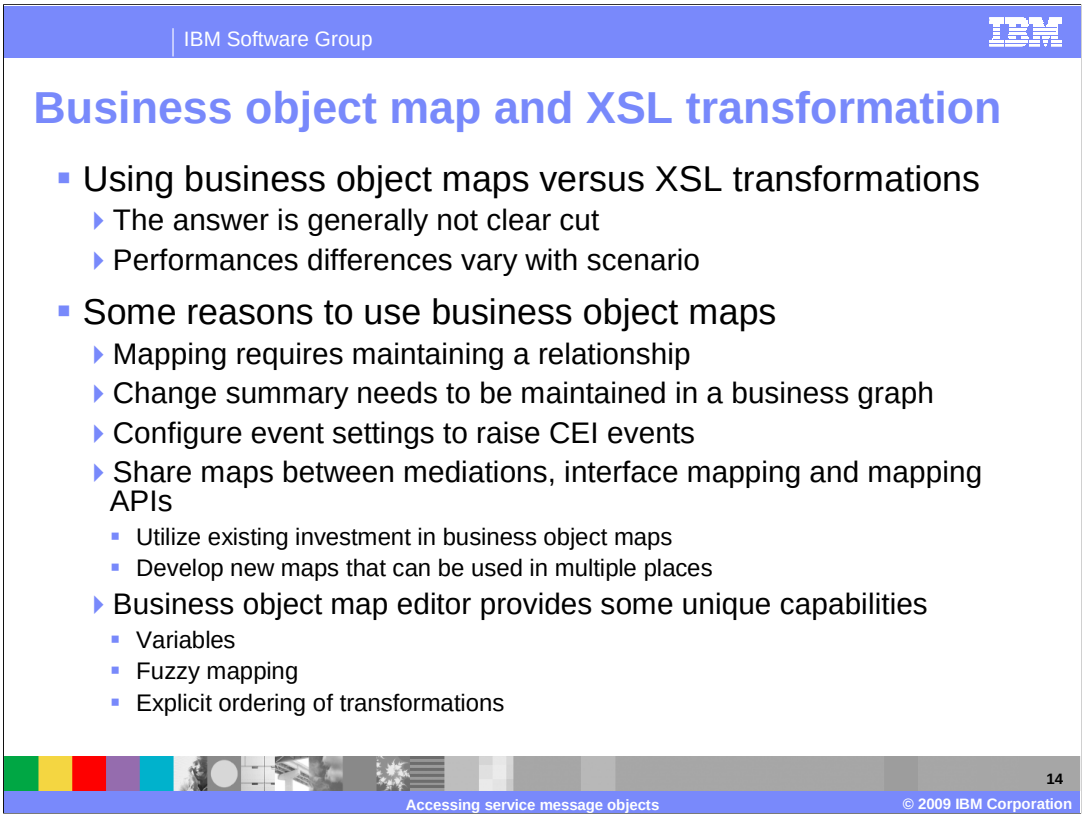
Business object maps

- Business object maps run generated Java code at runtime
- Business objects maps are used by business object map primitives
  - New maps can be created with the business object map editor
  - Existing maps can be selected
- These are the same maps used by mapping service APIs and for parameter mapping in interface maps
  - Possibly use mapping service APIs from custom mediation primitives
- Typical use is for modification of message type in a flow
- There is overlapping function provided by XSL style sheets and business object maps

Unlike the XSL style sheets that manipulate an XML version of the SMO at runtime, business object maps result in generated Java code that accesses an SDO representation of the SMO at runtime. The business object maps are used by the business object map primitive to manipulate the SMO within a mediation flow. When configuring a business object map primitive, you can launch the business object map editor for creating a new map or you can select an existing business object map to be used. These business object maps are the same maps that are used in WebSphere Process Server for performing parameter mapping within interface maps and are also used with the mapping service APIs. It is possible to make use of business object maps within a mediation flow by using the mapping service APIs from a custom mediation primitive. However, their typical use within a flow is through use of the business object map primitive.

Similar to an XSL transformation, the typical use of a business object map within a mediation flow is to change the message type of the SMO. All the same reasons documented on the previous XSL transformation slide apply equally to business object maps.

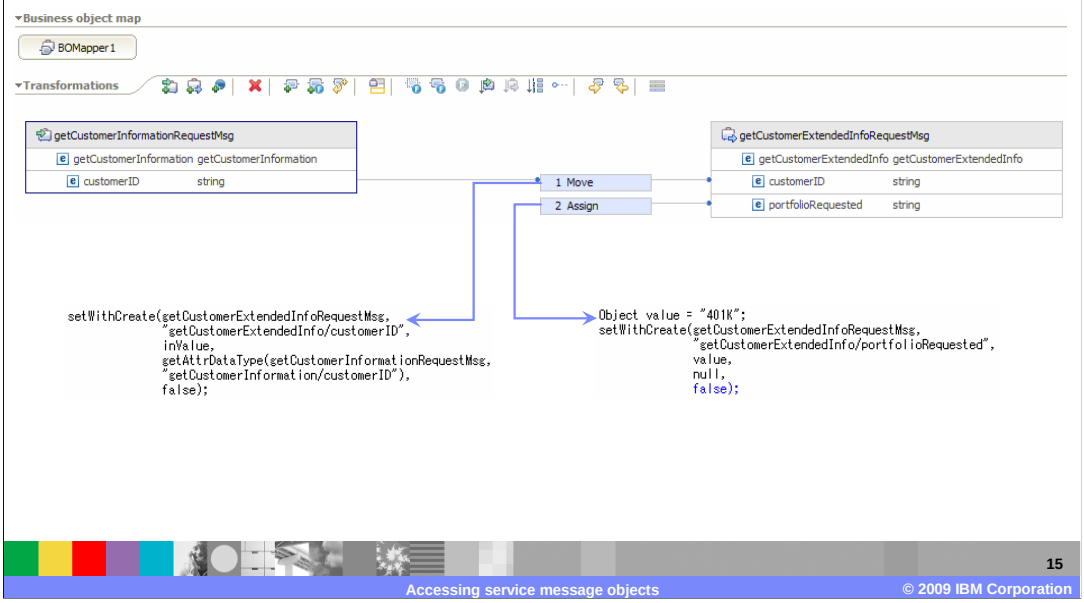
At this point is should be clear that there is overlapping function provided by XSL style sheets and business object maps, accomplishing basically the same function through different approaches. The next slide takes a closer look at this overlap.



With the overlapping functional capabilities of business object maps and XSL transformations it is worth considering which type to use in your mediation flows. To start out, the answer to this question is not generally clear cut and there is not a set of rules or guidelines to direct you in making a decision. In many cases, it will come down to a personal preference between the business object map editor and the XML mapping editor, but even these are very similar. One consideration might be runtime performance, but this is also not clear cut. It depends upon the specific scenario as to which option might be better, especially when you consider the very different underlying implementations. A discussion of those considerations is complex and goes beyond the scope of this presentation.

For particular requirements, however, there are some clear cut situations where using a business object map is preferable to using an XSL transformation. The relationship service, which maintains identity and lookup relationships, only works with business object maps. Likewise, the use of a business graph to maintain a change summary with your business object is also only supported with business object maps. You might have a situation where you want to do the same mapping in a mediation flow and in an interface map or mapping service API call. In this case you want to use a business object map to enable the reuse. This can be a situation where you are reusing existing business object maps or developing new maps to be used from multiple places. Although the two mapping editors are extremely similar, the business object map editor does provide for the use of variables, fuzzy mapping and the explicit ordering of transformations.

Business object maps and Java code



On this slide you can see a screen capture of the business object map editor with a simple business object mapping of a message body. This is the same mapping as was shown previously in the XML mapping editor. The mapping is from a getCustomerInformation operation to a getCustomerExtendedInfo operation. Each operation contains a customer ID field, which is moved from source to target. The target also has a portfolio requested field, which is set by the map to a default value of “401K”. At the bottom are screen captures of code extracted from the generated Java code for the map. This is not something you normally see or have any reason to look at, and these are just two statements extracted from a rather complex generated Java class. They are shown here to highlight again that business object maps result in generated Java classes that operate on an SDO representation of the SMO.

Section

Data handlers

This section takes a closer look at the use of data handlers for accessing SMOs.

Data handlers

- Data handlers convert between native data formats and business object formats
- Data handlers configured with a resource configuration
  - Data handler provides the implementation
  - Resource configuration provides configuration properties used to customize the data handler
- Data handlers normally used with SCA exports and imports
  - Resource configuration associated with export or import
  - Converts inbound message format to business objects format
  - Converts from business object format to outbound message format
  - Conversions are protocol independent
  - Data handlers used here are not manipulating the SMO

Data handlers are used to convert between any type of native data format and business object format. A data handler is an implementation that provides for the conversion in both directions. It normally has some properties that can be used to customize its behavior. For example, a data handler can be implemented to handle delimited strings, and the implementation uses a configurable property to specify the character used for the delimiter. A resource configuration is used to combine a data handler implementation with a specific set of configuration data.

Data handlers are normally used on the edges, in SCA exports and imports. A resource configuration, combining a data handler implementation and set of configuration properties, is configured on the export or import. For inbound messages, the data handler in the export converts from the native data in the message to business object format. For outbound messages, the data handler in the import converts from business object format to the native data format needed in the message. These conversions are protocol independent, so the same resource configuration can be used with imports and exports with different binding types. This explanation of data handler usage by imports and exports is independent of their use for manipulating SMOs, but provides you with background of their function that can help to understand their usage in a mediation flow. This is looked at on the next slide.

Data handler primitive

- The data handler primitive:
  - Is configured with a resource configuration
  - Provides the same data transformation capabilities as is normally done by an export or import
  - Used when native data is contained in an element of the SMO
  - Normally results in the SMO message type being changed
- Typical use
  - Service gateway scenarios where inbound native data is not transformed by the export
  - Any scenario where encoded data is passed in a business object field

Similar to its use with imports and exports, the data handler primitive is configured with a resource configuration and provides the same data transformation capabilities of converting between native data and business object formats. The data handler converts between an SMO element containing native data and a section of the SMO representing a business object. Normally, this involves the changing of the SMO body and thus changing the message type of the SMO.

The primary purpose for the data handler primitive is to support service gateway scenarios. In these scenarios, the inbound data is not converted by the export and is passed unchanged into the mediation flow wrapped in an SMO element. The data handler primitive is then used to convert the wrapped native data to business object format in the SMO.

In other scenarios, it is possible that an SMO element contains some form of encoded data, and the data handler primitive can also be used in this case.

Section

Java code

This section takes a closer look at the use of Java code for accessing SMOs.

Java code

- Java code can be used in custom mediation primitives
- Custom mediation operation:
  - Is passed the SMO as input parameter (which is a type of DataObject)
  - Fires an output terminal passing it the SMO
  - Input and output message types must match terminals of custom mediation
- DataObject API (commonj.sdo.DataObject)
  - Defined by the service data object (SDO) specification
  - Provides a dynamic loosely typed interface to access an SMO
  - Augmented subset of XPath 1.0 can be used with accessor methods
  - Javadoc and specification: http://help.eclipse.org/help32/index.jsp
- ServiceMessageObject API
  - Provides strongly typed interface for well defined portion of SMO
    - Everything except the contents of the body and the transient, correlation and shared contexts
  - Javadoc available from the Information Center

Custom mediation primitives contain Java code defining the logic for the primitive. The Java operation called by the custom mediation takes the SMO as input. The SMO is a type of DataObject, and can be accessed using either DataObject or ServiceMessageObject APIs. The body portion of the SMO that is passed in conforms to the message type defined for the input terminal. The code in the custom mediation primitive is required to pass the SMO to the fire method of the output terminal object. The body portion of the SMO that is passed to the output terminal object's fire method must conform to the message type of the output terminal of the custom mediation primitive.

The Java code can access the SMO using the DataObject APIs, which are defined by the Service Data Object specification, also known as SDO. These APIs provide a loosely typed interface for accessing the SMO and have accessor methods that make use of an augmented subset of XPath 1.0 to identify properties within the DataObject. The full Javadoc for these APIs and the SDO specification are available from the eclipse.org Web site at the address on the slide. There is also a set of ServiceMessageObject APIs, which provide a strongly typed interface for accessing the well-defined portion of the SMO. These APIs understand the schema for the SMO except for the body, the transient context, the correlation context and the shared context, each of which is unique to each individual flow. There is Javadoc describing these APIs in the information center.

Java code examples

- Compare DataObject API use to SMO API use
  - Code to access the MessageUUID field contained in the SMOHeader
    - Using DataObject with full path

```
// Get Message UUID using DataObject and XPath
String uuid = smo.getString("headers/SMOHeader/MessageUUID");
```

    - Using DataObject and traversing down through each property

```
// Get Message UUID using DataObject and traverse properties
DataObject header = smo.getDataObject("headers");
DataObject SMOHeader = header.getDataObject("SMOHeader");
String uuid = SMOHeader.getString("MessageUUID");
```

    - Using the ServiceMessageObject strongly typed APIs

```
// Get Message UUID using SMO strongly typed APIs
HeadersType headers = smo.getHeaders();
SMOHeaderType SMOHeader = headers.getSMOHeader();
String uuid = SMOHeader.getMessageUUID();
```

- Discovery of coding errors
  - Loosely typed DataObject errors such as misspelling a property name are not discovered until runtime
  - Strongly typed SMO errors are caught at compile time

This slide compares the use of the DataObject APIs to the ServiceMessageObject APIs by using three examples that do the same thing. They each access the MessageUUID field that is in the SMOHeader.

The first code example uses the loosely typed DataObject APIs and a fully qualified XPath expression to identify the field. The XPath expression headers/SMOHeader/MessageUUID is used to obtain the MessageUUID string from the SMO DataObject.

The second code example also uses the loosely typed DataObject APIs, but in this case traverses down through each property individually. First, the headers DataObject is obtained from the SMO DataObject. Then the SMOHeader DataObject is obtained from headers DataObject. Finally, the MessageUUID string is obtained from SMOHeader DataObject.

In the third code example, the strongly typed SMO APIs are used. Using the ServiceMessageObject APIs, the getHeaders operation is used to obtain the headers from the SMO. From the headers object, the getSMOHeader operation is used to obtain the SMOHeader. Finally, the getMessageUUID operation is used to get the MessageUUID string from the SMOHeader.

Contrasting these two approaches with respect to coding errors, the loosely typed DataObject APIs generally have coding errors surface at runtime, whereas the strongly typed SMO APIs tend to have coding errors caught at compile time. Therefore, the SMO APIs might be preferred over the DataObject APIs for ease in development and debugging.

Section

Summary

This section presents a summary of the presentation.

Summary

- Examined service message objects in terms of:
  - Access and manipulation of SMO content
  - Manipulation of message type
- Approaches used
  - XPath
  - XSL transformations (XSL style sheets)
  - Business object maps
  - Data handlers
  - Java code

This presentation presented details about accessing service message objects, including modifying their content and changing their message type. The use of XPath expressions, XSL transformations using XSL style sheets, business object maps, data handlers, and the use of Java code in custom mediation primitives were examined.

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_AccessingSMOs.ppt

This module is also available in PDF format at: ../WBPMv62_AccessingSMOs.pdf

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

Trademarks, copyrights, and disclaimers

IBM, the IBM logo, ibm.com, and the following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

WebSphere

If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of other IBM trademarks is available on the Web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml

Java, Javadoc, and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements or changes in the products or programs described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (for example, IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.