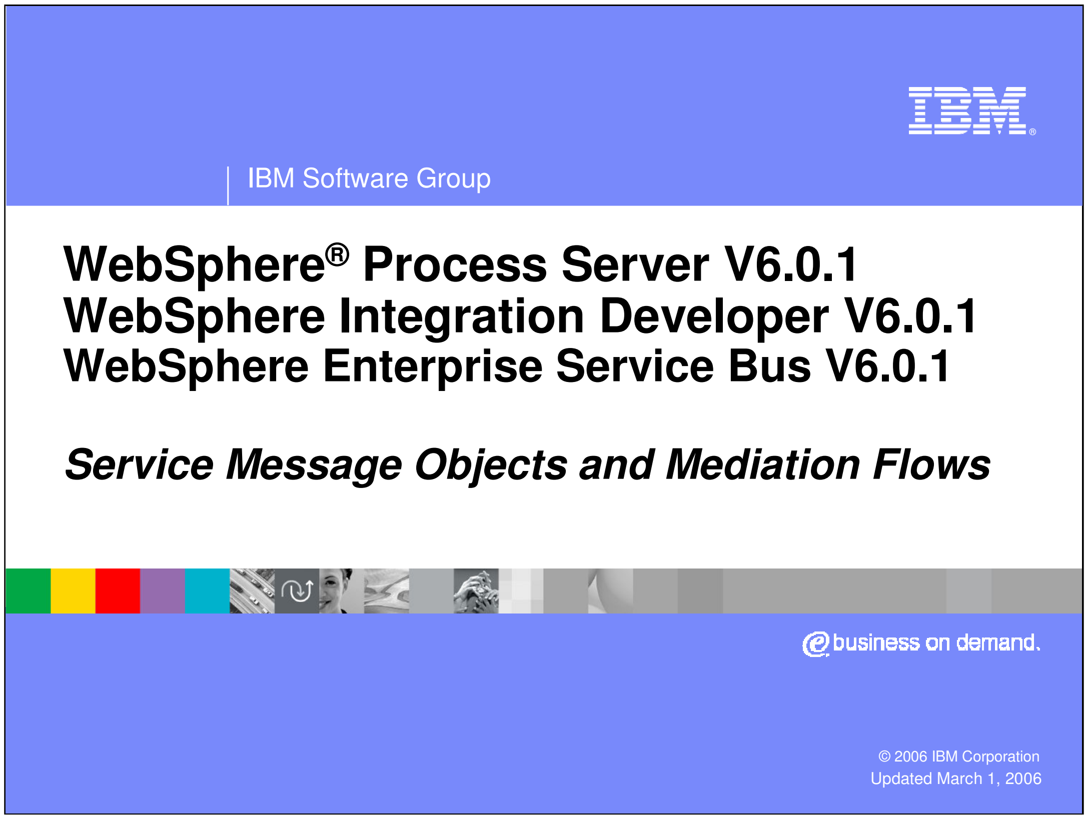


IBM Software Group

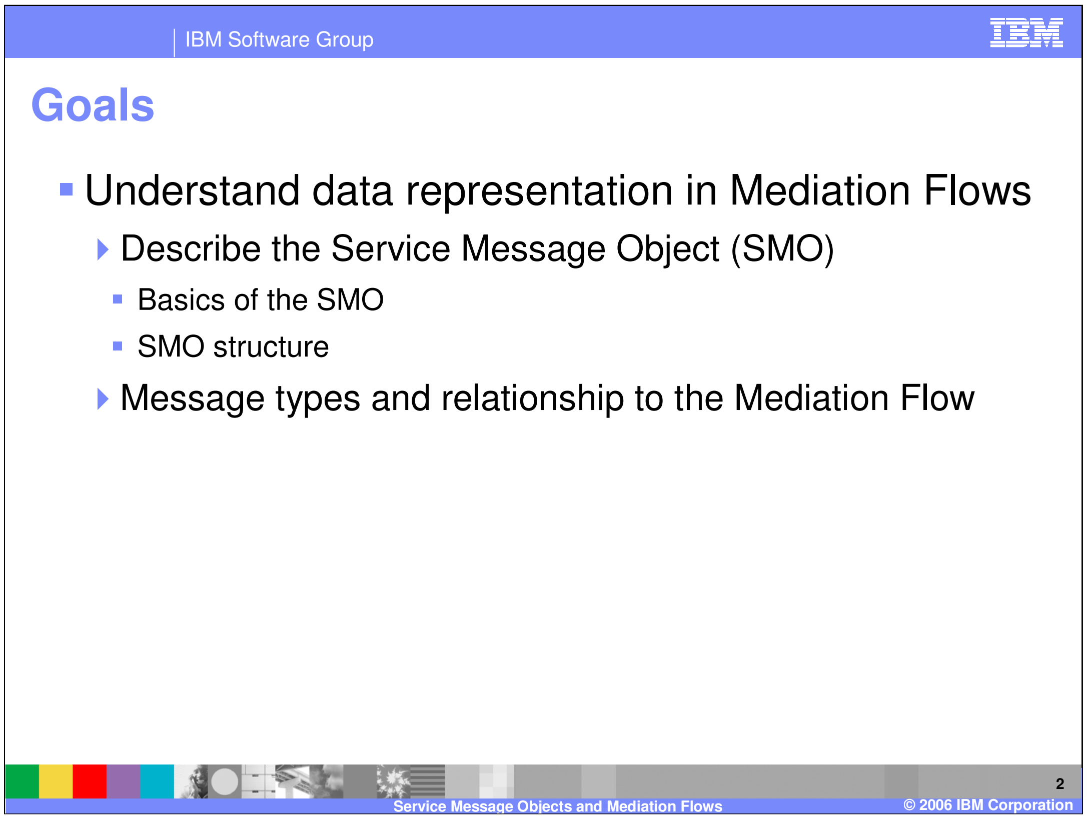
WebSphere® Process Server V6.0.1
WebSphere Integration Developer V6.0.1
WebSphere Enterprise Service Bus V6.0.1

Service Message Objects and Mediation Flows

business on demand.

Updated March 1, 2006

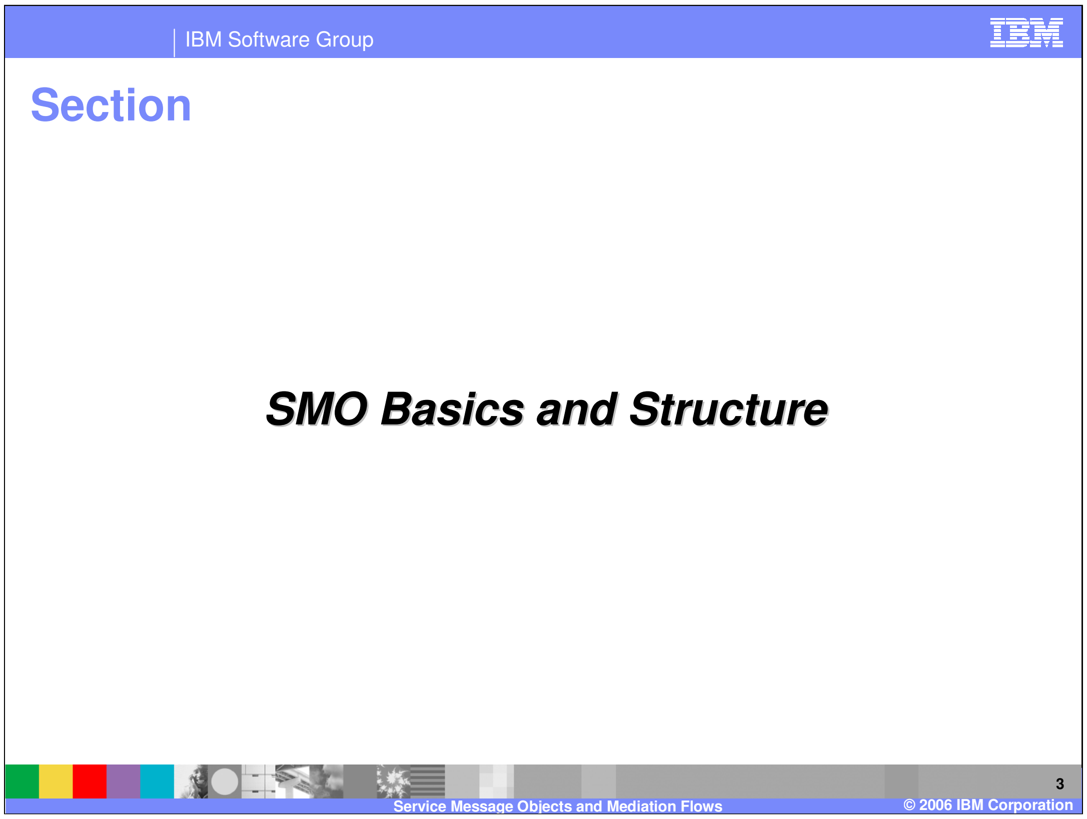
This presentation will describe Service Message Objects and how they relate to Mediation Flows.



Goals

- Understand data representation in Mediation Flows
  - Describe the Service Message Object (SMO)
    - Basics of the SMO
    - SMO structure
  - Message types and relationship to the Mediation Flow

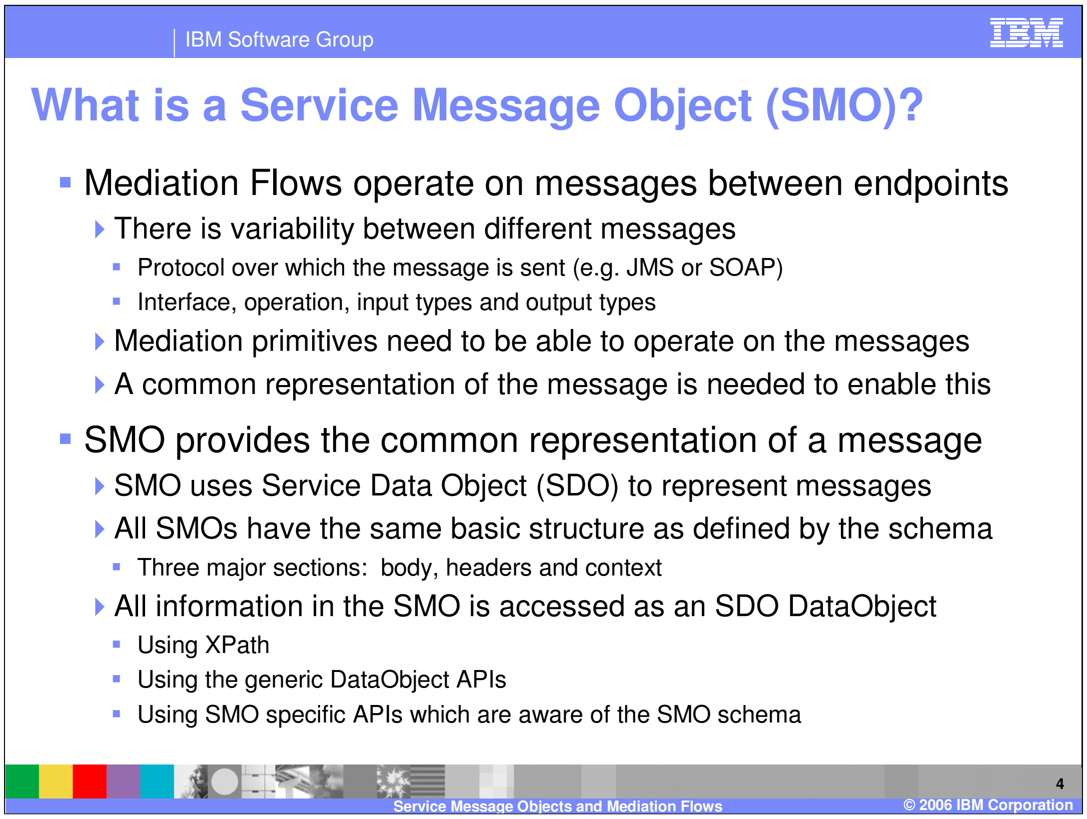
The goal of this presentation is to provide an understanding of how data is represented in a mediation flow. The data in a flow is described using a Service Message Object, also known as an SMO. The presentation explains some basic characteristics of an SMO and then looks at its overall structure. The structure of the application portion of the SMO, referred to as the body or payload, defines a message type. Message types are an important element when considering the logic and flow of a mediation. This presentation describes the relationship between message types and mediation flows.



Section

SMO Basics and Structure

This section of the presentation looks at the basic characteristics and structure of an SMO.



What is a Service Message Object (SMO)?

- Mediation Flows operate on messages between endpoints
  - There is variability between different messages
    - Protocol over which the message is sent (e.g. JMS or SOAP)
    - Interface, operation, input types and output types
  - Mediation primitives need to be able to operate on the messages
  - A common representation of the message is needed to enable this
- SMO provides the common representation of a message
  - SMO uses Service Data Object (SDO) to represent messages
  - All SMOs have the same basic structure as defined by the schema
    - Three major sections: body, headers and context
  - All information in the SMO is accessed as an SDO DataObject
    - Using XPath
    - Using the generic DataObject APIs
    - Using SMO specific APIs which are aware of the SMO schema

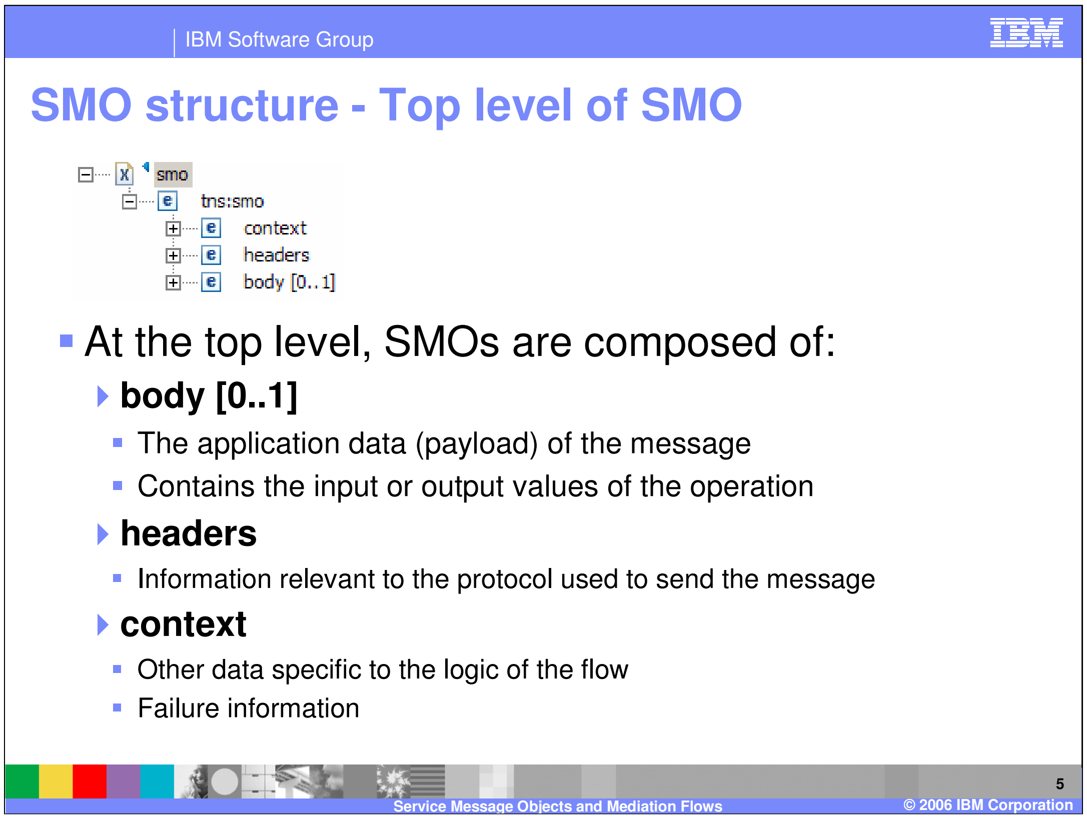
In order to answer the question "What is a Service Message Object?" you need to first understand some characteristics of a mediation flow. The primary function of a mediation flow is to operate on a message between endpoints where a service requestor and a service provider are those endpoints.

The first thing to understand is that a message may take on many different forms. This is because the protocol used to send a message can vary, such as using JMS or SOAP. Also, each message will be different depending upon the interface and operation the message is for and if this is the request side or response side of the interaction between the requestor and provider.

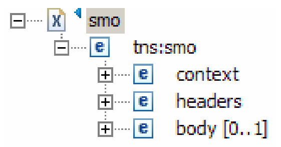
The next point to understand is that within the mediation flow, mediation primitives are used to operate on the message to examine and update the message's contents. Mediation primitives need some kind of a common representation of a message in order to do this.

So that is what a Service Message Object does, it provides a common representation of a message that accounts for differing protocols and differing interfaces, operations and parameters that the message represents.

SMOs are built upon the use of Service Data Object technology, also referred to as SDO. Using SDO there is a schema that describes the basic structure of an SMO which is composed of three major sections. The body of the message represents the specific interface, operation and parameters relevant to this message. The headers section of the message represents information about the protocol over which the message was sent. Lastly, the context section represents data that is important to the internal logic of the flow itself. Each of these major sections of the SMO will be examined in more detail in subsequent slides.



SMO structure - Top level of SMO



- At the top level, SMOs are composed of:
  - **body [0..1]**
    - The application data (payload) of the message
    - Contains the input or output values of the operation
  - **headers**
    - Information relevant to the protocol used to send the message
  - **context**
    - Other data specific to the logic of the flow
    - Failure information

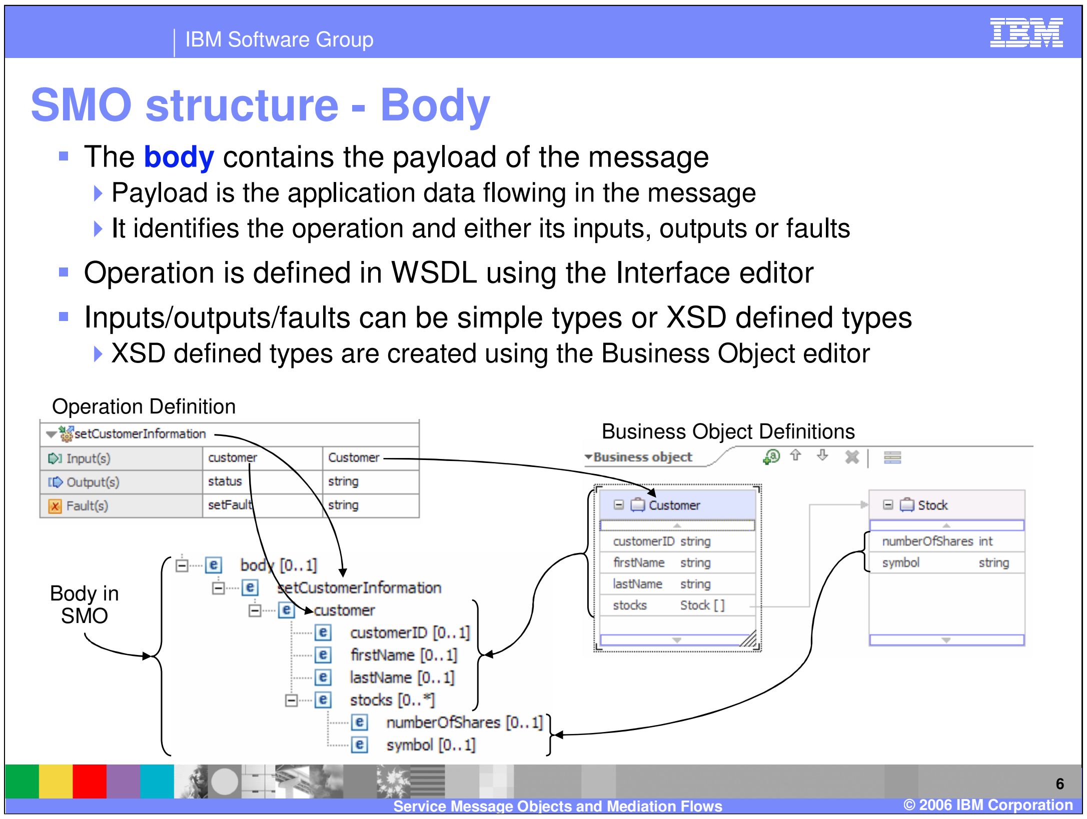
This is an illustration the three major sections of an SMO that were introduced on the previous slide.

The body of the SMO contains the application data which is sometimes referred to as the payload. This is the data that is relevant to the endpoints, the service requestor and service provider. The body describes the operation being performed as will as the inputs or outputs of that operation. The schema definition shows that it is possible for the body not to be present, but in practice an SMO always contains a body.

The headers of the SMO contains protocol specific information associated with the protocol over which the message is being sent.

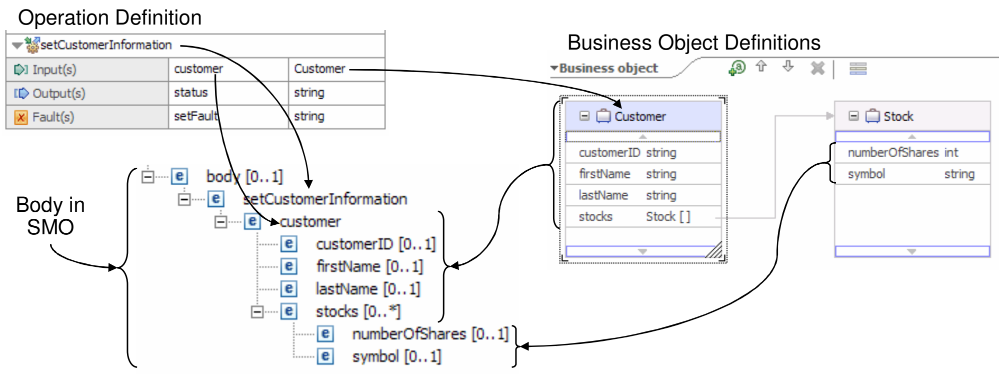
The context of the SMO contains data that is required by the logic of the flow. This data exists within the flow itself but is not passed to or from the requestor or provider. Under certain conditions error information is also added to the context.

Each of these sections of the SMO will be examined more closely in the following slides.



SMO structure - Body

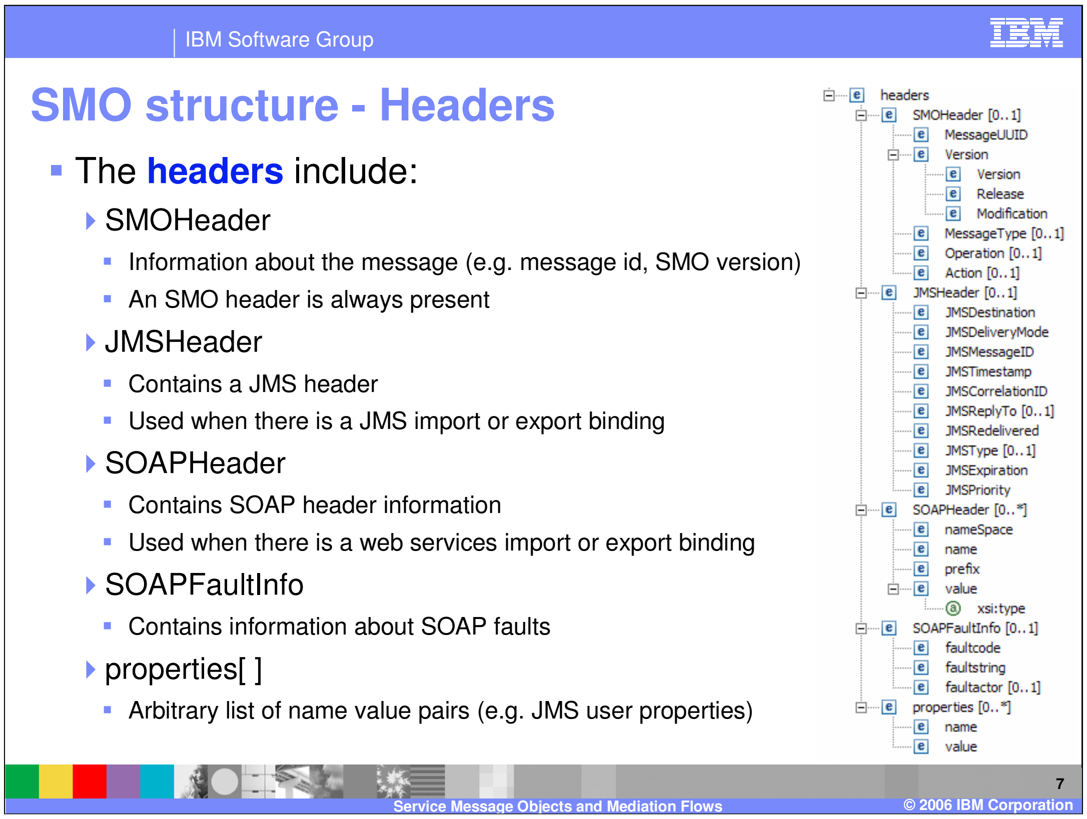
- The **body** contains the payload of the message
  - Payload is the application data flowing in the message
  - It identifies the operation and either its inputs, outputs or faults
- Operation is defined in WSDL using the Interface editor
- Inputs/outputs/faults can be simple types or XSD defined types
  - XSD defined types are created using the Business Object editor



The body of the SMO contains the payload, which is the application data that flows between a service requestor and service provider. The body represents a specific operation on a specific interface. The data associated with that operation is also contained in the body and will be either the inputs, the outputs or the faults defined for the operation. The interface is a WSDL defined interface. The Interface Editor in WebSphere Integration Developer can be used to define such an interface. The inputs, outputs and faults can be simple types or they can be XSD defined types. The Business Object Editor in WebSphere Integration Developer can be used to define these types.

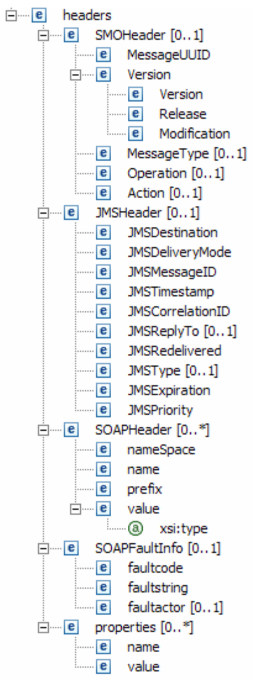
The illustration at the bottom of the slide shows the relationship between an interface defined in the Interface Editor, a business object defined in the Business Object Editor and the contents of the body of an SMO. In the lower left section you will see an SMO body expanded to see the individual elements. Starting at the upper left in the Interface Editor, you will see an operation definition for an operation called getCustomerInformation. Notice that the body contains as its top level a section called getCustomerInformation. Looking at this operation you see that it has an input called customer which is defined by a Customer Business Object. Since this SMO body represents the request flow it contains the inputs. You will see within the SMO that the getCustomerInformation section contains a customer section. To understand what is contained in the customer section, you need to look at the Business Object Editor in the upper right where the Customer business object is defined. It is composed of 4 fields, a customerID, firstName and lastName which are all strings and a stocks field which is an array of Stock business objects. A Stock business object is composed of two fields, numberOfShares which is an int and symbol which is a string. Now if you direct your attention to the SMO body you will see that it contains the same elements as the Customer business object defined in the Business Object Editor.

So you can see that the body of the SMO truly is a representation of an operation and the data associated with that operation.



SMO structure - Headers

- The **headers** include:
  - SMOHeader
    - Information about the message (e.g. message id, SMO version)
    - An SMO header is always present
  - JMSHeader
    - Contains a JMS header
    - Used when there is a JMS import or export binding
  - SOAPHeader
    - Contains SOAP header information
    - Used when there is a web services import or export binding
  - SOAPFaultInfo
    - Contains information about SOAP faults
  - properties[ ]
    - Arbitrary list of name value pairs (e.g. JMS user properties)



The headers section of the SMO contains protocol information related to the mediation flow and is divided into several subsections.

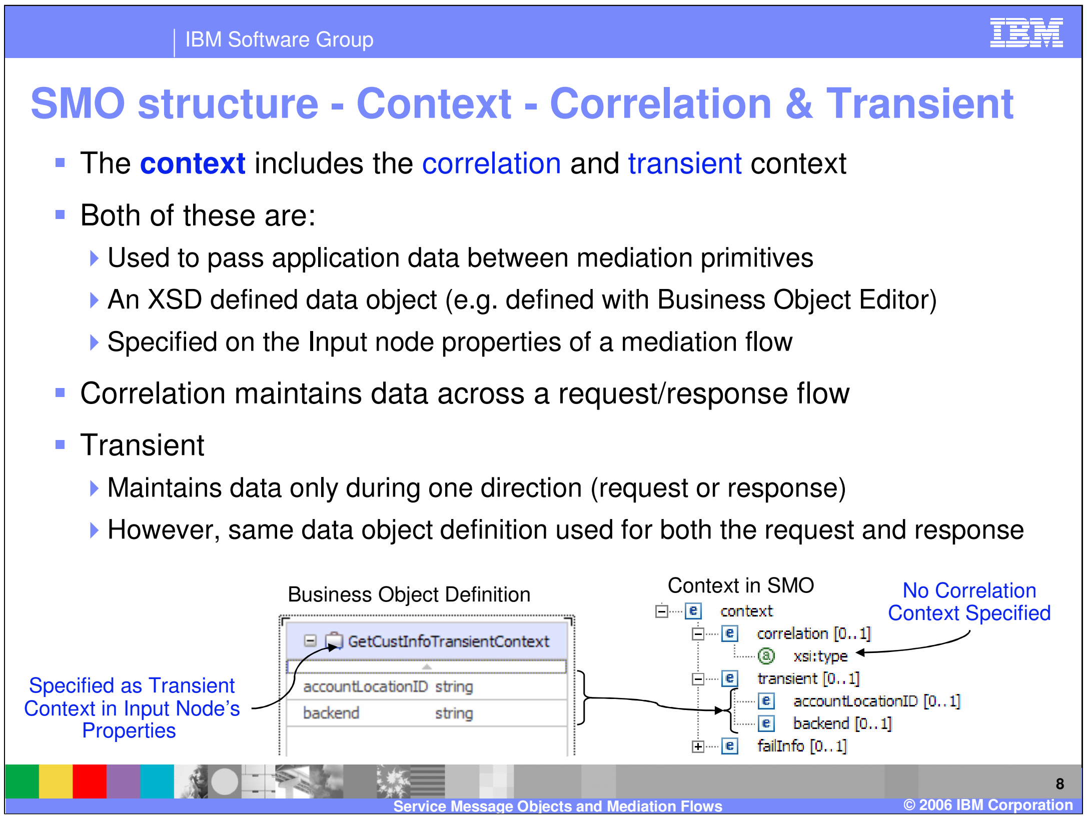
The **SMOHeader** contains information that defines the message, such as a unique message ID, the version number and message type. The SMO header is always present in a service message object.

The **JMSHeader** contains the JMS header information when a JMS import or export binding is used to initiate this flow.

The **SOAPHeader** contains SOAP header information when a web services import or export binding is used to initiate this flow.

The **SOAPFaultInfo** contains information about SOAP faults and will be present when a web services import or export binding is used and a SOAP fault has occurred.

The **properties** provide the ability to include an arbitrary list of name/value pairs that can be use to represent any information. An example of the use of properties would be to contain JMS user properties that were included with a JMS message.



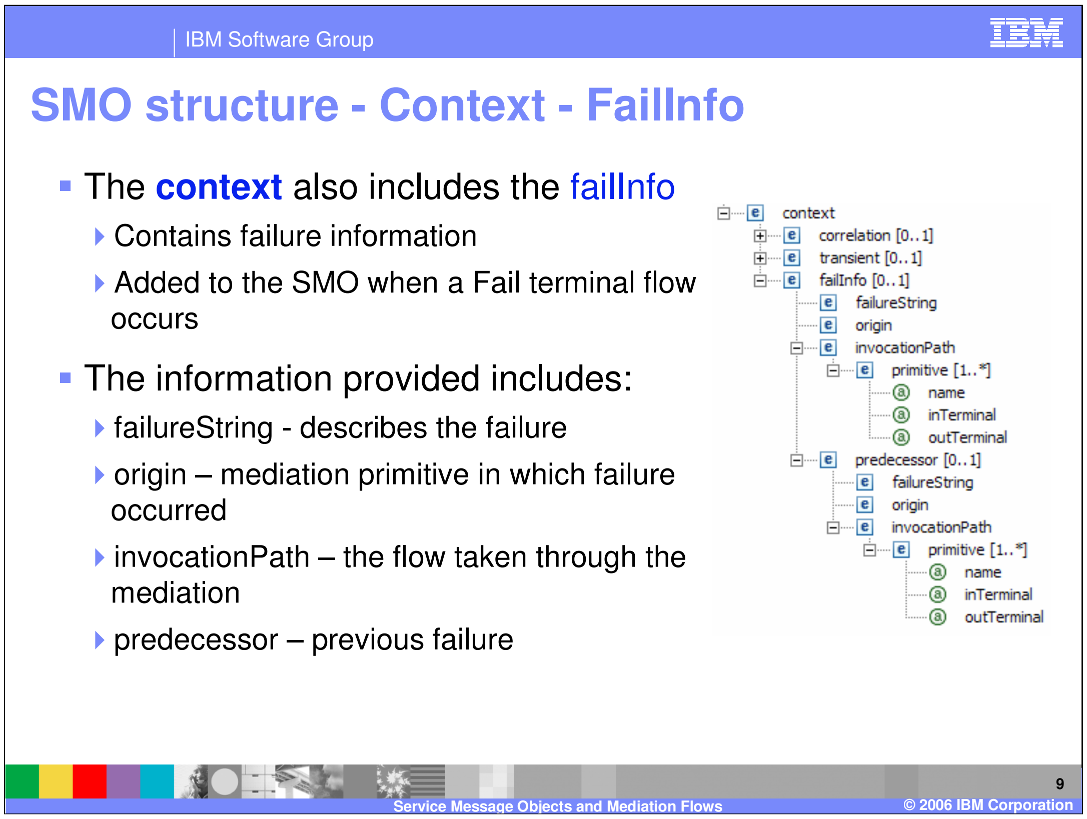
SMO structure - Context - Correlation & Transient

- The **context** includes the correlation and transient context
- Both of these are:
  - Used to pass application data between mediation primitives
  - An XSD defined data object (e.g. defined with Business Object Editor)
  - Specified on the Input node properties of a mediation flow
- Correlation maintains data across a request/response flow
- Transient
  - Maintains data only during one direction (request or response)
  - However, same data object definition used for both the request and response

The context section of an SMO contains the correlation and the transient context. There are several things that are common about the correlation and transient context. They are both used to pass flow specific information between mediation primitives. An XSD defined data object, such as one created using the Business Object Editor, is used to define a correlation context or a transient context. Each of these is associated with a flow by specifying the appropriate business object on the input node of the mediation flow. Where a correlation context and transient context differ is in the scope over which they maintain their data. A correlation context will retain its data across a request/response flow and therefore can be used to pass data from a mediation primitive on the request flow side to a mediation primitive on the response flow side. A transient context can be used during either the request or response flow but does not retain the data set in the request for access by the response. Only one business object is used to define the transient context. Therefore, if you want to use it on both the request and response flows, the business object definition must contain the fields required for both sides of the flow. This is true even though the values set in the request flow will not be available for the response flow.

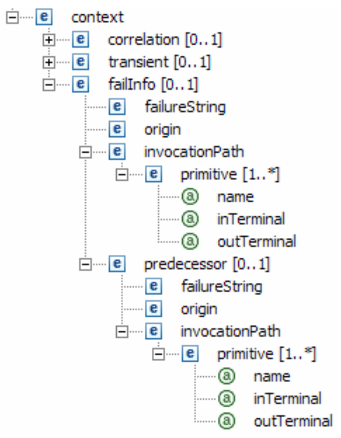
The bottom of the slide on the right side shows an expanded context section of an SMO. In this particular example there has been no correlation context specified but there is a transient context. The left side shows the definition of the transient context in the Business Object Editor and you can see that the fields defined in the business object are the same as the fields that appear in the SMO.

The context section of the SMO can also contains failure information, which will be examined on the next slide.

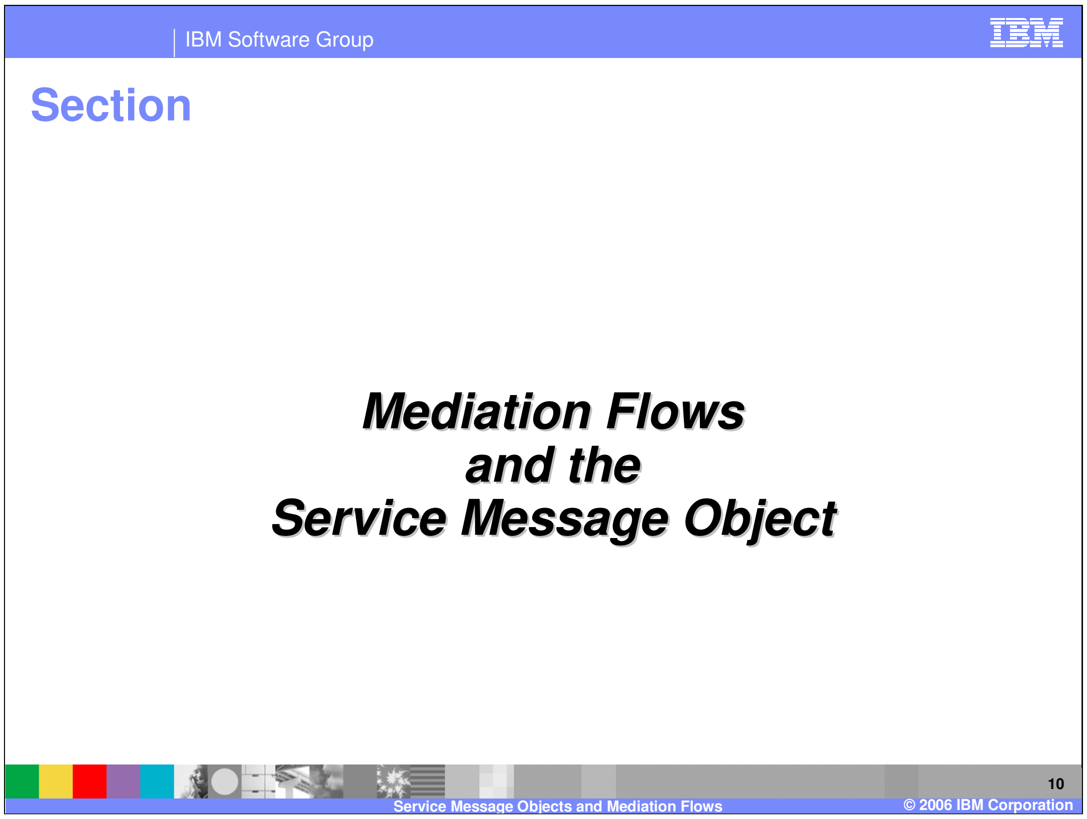


SMO structure - Context - FailInfo

- The **context** also includes the failInfo
  - Contains failure information
  - Added to the SMO when a Fail terminal flow occurs
- The information provided includes:
  - failureString - describes the failure
  - origin – mediation primitive in which failure occurred
  - invocationPath – the flow taken through the mediation
  - predecessor – previous failure



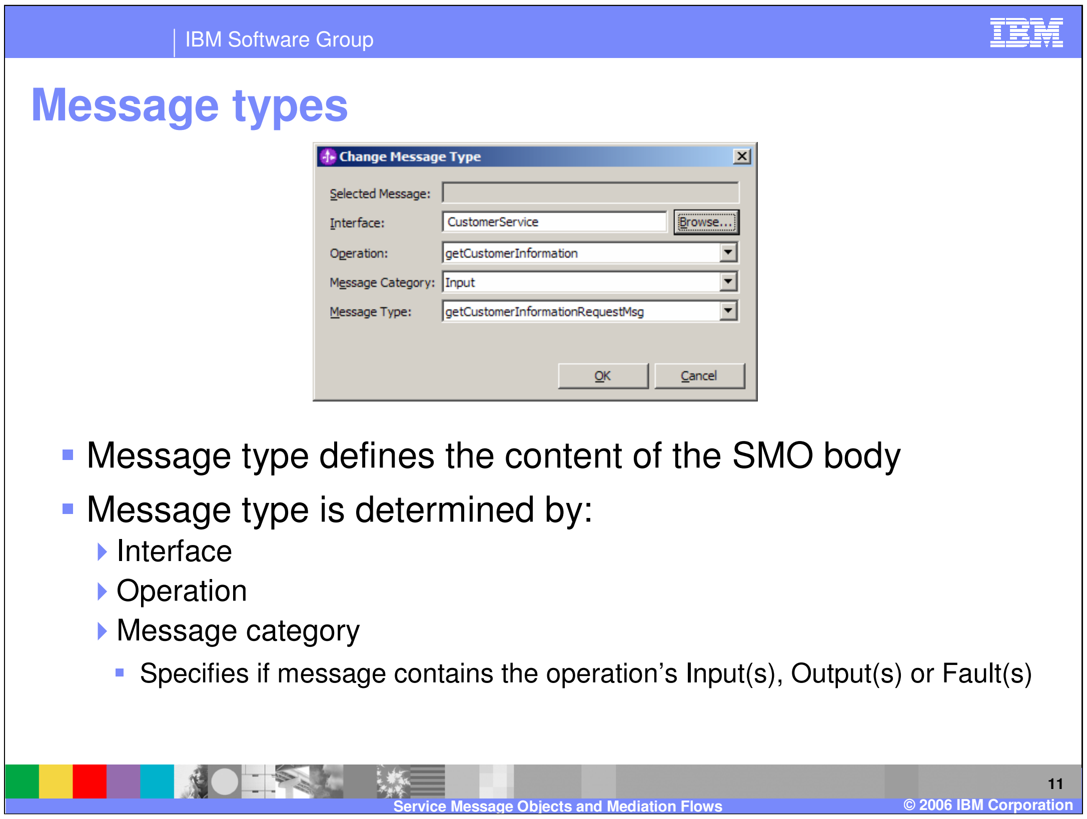
On the right you see an expanded view of the failInfo portion of the context section. This is used to contain information about a failure that occurred during the flow. It is only populated when a failure occurs in a mediation primitive and the mediation primitive has its fail terminal wired to another primitive or node. This allows a mediation flow to examine a failure and determine how the failure should be handled. The failInfo contains a string which describes the failure, the name of the mediation primitive in which the failure occurred and information about the path taken through the flow prior to the failure. In the event that a second failure occurs while processing the first failure the predecessor section is used to retain the information about the original failure.



Section

Mediation Flows and the Service Message Object

This section looks at the relationship between SMOs and mediation flows, in particular how message type plays a major role when defining a mediation flow.



Message types

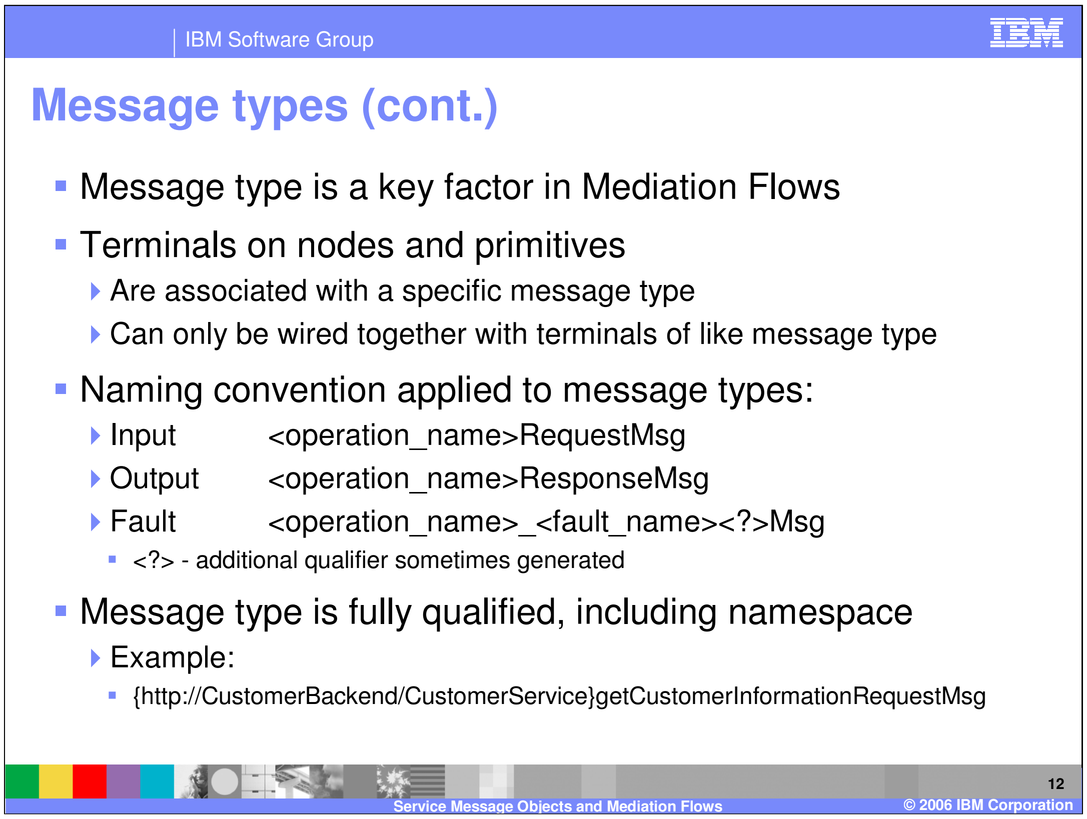
Change Message Type

Selected Message:	
Interface:	CustomerService
Operation:	getCustomerInformation
Message Category:	Input
Message Type:	getCustomerInformationRequestMsg

Browse...
OK
Cancel

- Message type defines the content of the SMO body
- Message type is determined by:
  - Interface
  - Operation
  - Message category
    - Specifies if message contains the operation's Input(s), Output(s) or Fault(s)

A message type defines what the structure of an SMO body will be. The message type is defined by the interface and the operation associated with the message and the message category. The message category indicates if the message contains the operations inputs, outputs or faults. The screen capture in the slide shows the Change Message Type dialog. It is used by first browsing for and selecting an interface. Once that is done, the Operation dropdown box is used to select an operation from the list all of the operations defined on that interface. Finally the Message Category is set indicating if it is the operation's inputs, outputs or faults that will be included. From these three settings, the Message Type field will be set to some specific type.



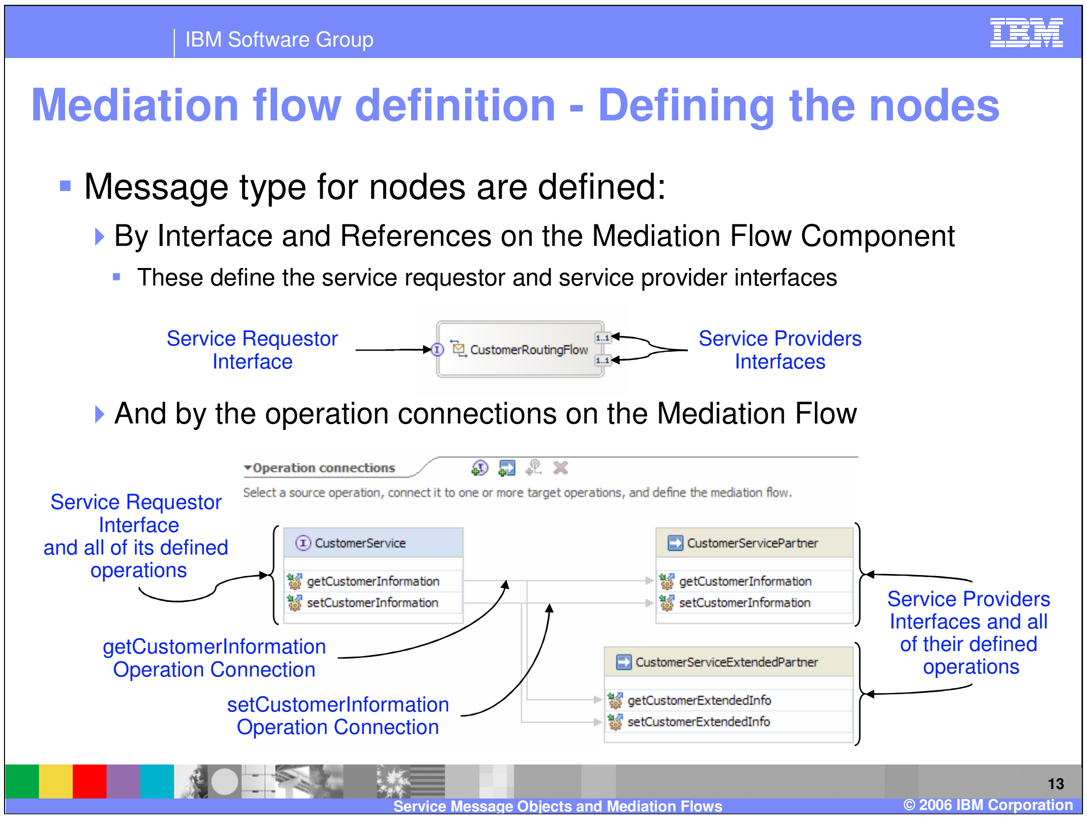
Message types (cont.)

- Message type is a key factor in Mediation Flows
- Terminals on nodes and primitives
  - Are associated with a specific message type
  - Can only be wired together with terminals of like message type
- Naming convention applied to message types:
  - Input <operation_name>RequestMsg
  - Output <operation_name>ResponseMsg
  - Fault <operation_name>_<fault_name><?>Msg
    - <?> - additional qualifier sometimes generated
- Message type is fully qualified, including namespace
  - Example:
    - {http://CustomerBackend/CustomerService}getCustomerInformationRequestMsg

Message types are a key factor when defining a mediation flow. In a mediation flow the nodes and mediation primitives have terminals and each terminal is associated with a specific message type. When wiring a flow, only terminals of like message type can be wired together.

There is a naming convention that is used for the definition of message types. For an input message the convention is operation name request message. For an output message, it is operation name response message and for a fault it is operation name underbar fault name message, with a possible qualifier used between fault name and message.

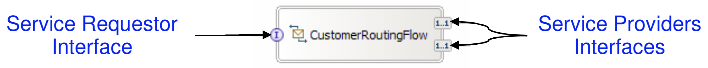
These naming conventions are actually the shortened form of the message type which appears in the mediation flow editor whereas the real message type appears in the properties view. The real message type is a fully qualified name and includes both the namespace and interface as is shown in the example above. This example shows a namespace of http://CustomerBackend, an interface of CustomerService, an operation of getCustomerInformation and it ends in request message to indicate this is for a request flow and will contain the inputs.



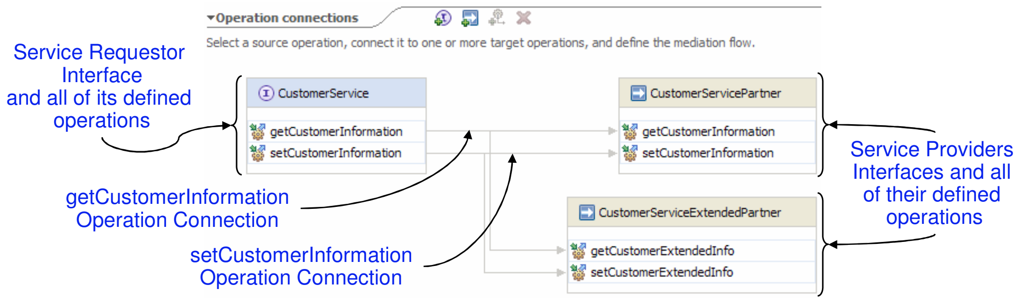
IBM Software Group

Mediation flow definition - Defining the nodes

- Message type for nodes are defined:
  - By Interface and References on the Mediation Flow Component
    - These define the service requestor and service provider interfaces



  - And by the operation connections on the Mediation Flow


Operation connections
Select a source operation, connect it to one or more target operations, and define the mediation flow.
Service Requestor Interface and all of its defined operations
CustomerService
getCustomerInformation
setCustomerInformation
CustomerServicePartner
getCustomerInformation
setCustomerInformation
CustomerServiceExtendedPartner
getCustomerExtendedInfo
setCustomerExtendedInfo
Service Providers Interfaces and all of their defined operations
getCustomerInformation Operation Connection
setCustomerInformation Operation Connection

13

Service Message Objects and Mediation Flows

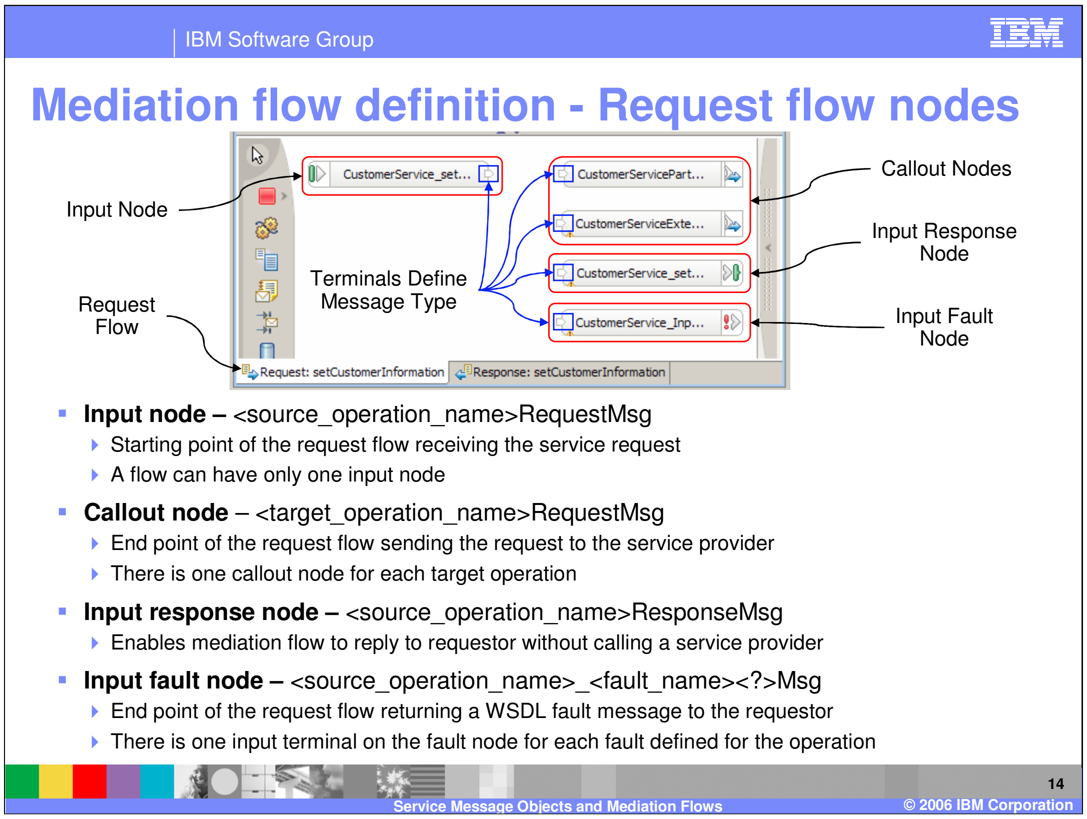
© 2006 IBM Corporation

The next several slides will be used to show how a mediation flow is defined. The specific focus is on the message types associated with the terminals of the nodes and mediation primitives that make up the flow.

Every mediation flow has nodes which represent the entry and exit points for the flow. The nodes have terminals which have fixed message types. The interfaces and operations associated with the flow determine which nodes are present in the flow and the message types associated with their terminals.

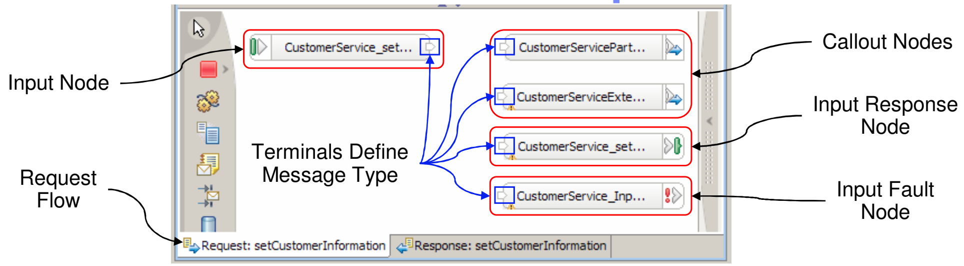
It starts with the definition of the Mediation Flow Component in the assembly diagram as is shown in the top of the slide. The Mediation Flow Component has an interface which defines the interface used by a requestor and also has references defining the interfaces to use for calling to providers. Looking at the lower portion of the slide you see that the Operation Connections panel of the Mediation Flow Editor shows all of the operations associated with the defined interfaces. Using this panel the operations on the input interface are connected to operations on the interfaces used to call providers.

Having done this there is now sufficient information for any input operation to define the nodes for the flow, including the message types associated with the terminals for the nodes.



IBM Software Group

Mediation flow definition - Request flow nodes



- **Input node –** <source_operation_name>RequestMsg
  - Starting point of the request flow receiving the service request
  - A flow can have only one input node
- **Callout node** – <target_operation_name>RequestMsg
  - End point of the request flow sending the request to the service provider
  - There is one callout node for each target operation
- **Input response node –** <source_operation_name>ResponseMsg
  - Enables mediation flow to reply to requestor without calling a service provider
- **Input fault node –** <source_operation_name>_<fault_name><?>Msg
  - End point of the request flow returning a WSDL fault message to the requestor
  - There is one input terminal on the fault node for each fault defined for the operation

14

Service Message Objects and Mediation Flows

© 2006 IBM Corporation

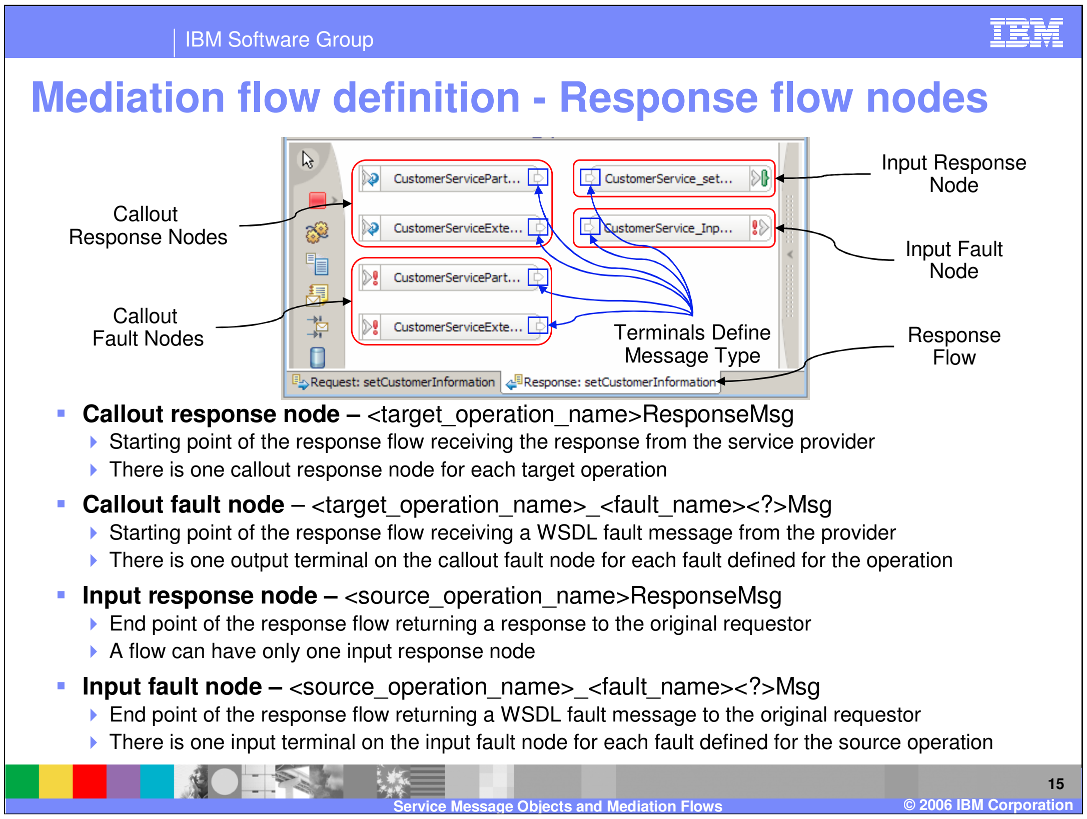
This slide shows the canvas of the Mediation Flow Editor for the request flow.

Starting at the upper left of the canvas, you will see the Input Node which is the starting point for the request flow. There will be only one input node for a mediation request flow. It has an output terminal with a message type of source operation name request message.

On the right side of the canvas, the top two nodes are the Callout Nodes. These are the end points for the request flow where a call is made to a service provider. There will be one callout node for every target operation defined in the Operations Connections panel. The callout nodes each have an input terminal with a message type of target operation name request message.

The third node down on the right side is the Input Response Node. This nodes enables the mediation flow to return directly to the requestor without calling a service provider and can be used where the mediation flow knows how to satisfy the request. The input response node has an input terminal with a message type of source operation name response message.

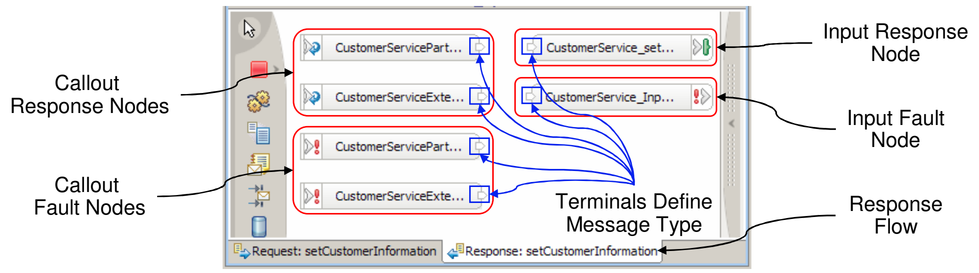
The bottom node on the right is the Input Fault Node. This node enables the mediation flow to return a WSDL fault to the requestor and can be used when some error has been detected within the mediation flow. This node may have multiple input terminals, one for each of the faults defined on the source operation. The message type associated with each terminal is source operation name underbar fault name optional qualifier message. If there are no faults defined for the source operation, the input fault node will not be present on the canvas.



IBM Software Group

IBM

Mediation flow definition - Response flow nodes



- **Callout response node –** <target_operation_name>ResponseMsg
  - Starting point of the response flow receiving the response from the service provider
  - There is one callout response node for each target operation
- **Callout fault node** – <target_operation_name>_<fault_name><?>Msg
  - Starting point of the response flow receiving a WSDL fault message from the provider
  - There is one output terminal on the callout fault node for each fault defined for the operation
- **Input response node –** <source_operation_name>ResponseMsg
  - End point of the response flow returning a response to the original requestor
  - A flow can have only one input response node
- **Input fault node –** <source_operation_name>_<fault_name><?>Msg
  - End point of the response flow returning a WSDL fault message to the original requestor
  - There is one input terminal on the input fault node for each fault defined for the source operation

15

Service Message Objects and Mediation Flows

© 2006 IBM Corporation

This slide shows the canvas of the Mediation Flow Editor for the response flow.

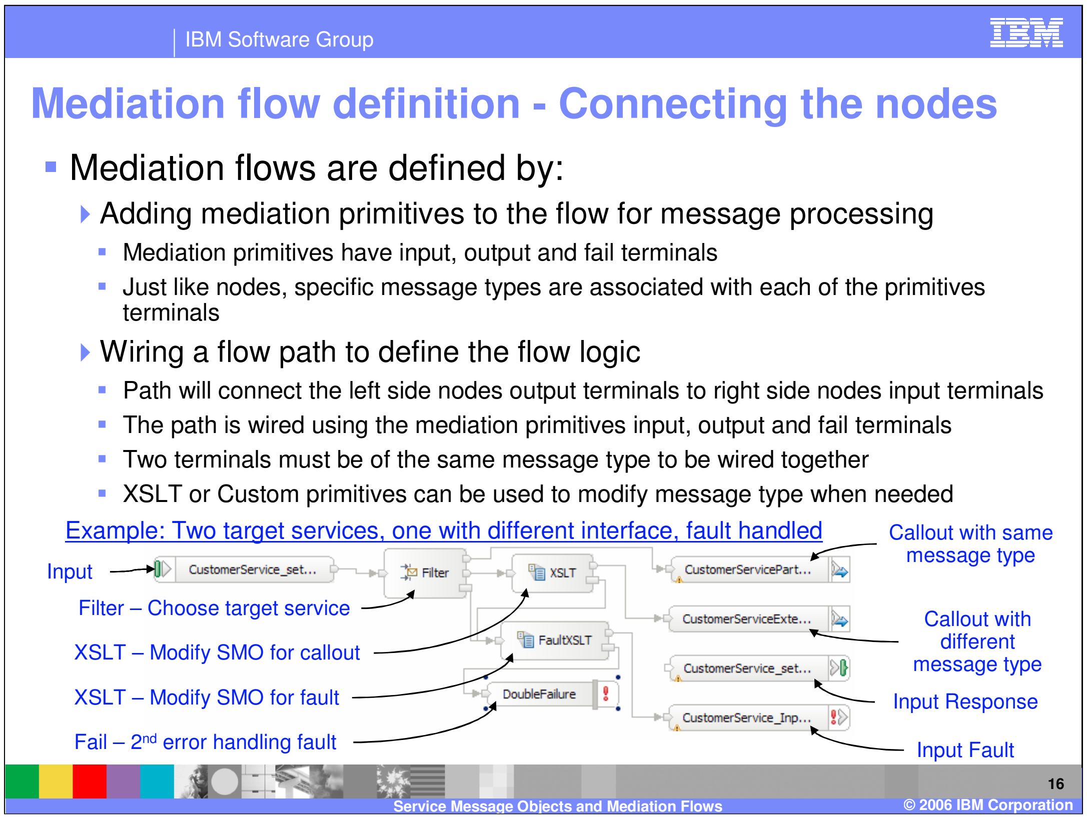
Starting at the upper left of the canvas, you will see that there are two Callout Response Nodes. These are the starting points for the response flow where the return from the service provider is received. There will be one callout response node for every target operation defined in the Operations Connections panel. They each have one output terminal with a message type of target operation name response message.

The lower two nodes on the left side are the Callout Fault Nodes. These are the starting points for the response flow when a service provider returns a fault. There will be one callout fault node for every target operation which has one or more faults defined. These nodes may have multiple output terminals, one for each defined fault on the target operation. The message type for the terminals will be target operation name underbar fault name optional qualifier message.

On the right side of the canvas, the top node is the Input Response Node. This is the end point for the response flow returning to the original service requestor. There will be only one input response in a response flow. The input terminal of this node has a message type of source operation name response message.

The bottom node on the right side is the Input Fault Node. This node is used to return a WSDL fault to the original service requestor. There may be multiple input terminals on this node, one for each of the faults defined on the source operation. The message type associated with each terminal is source operation name underbar fault name optional qualifier message. If there are no faults defined for the source operation, the input fault node will not be present on the canvas.

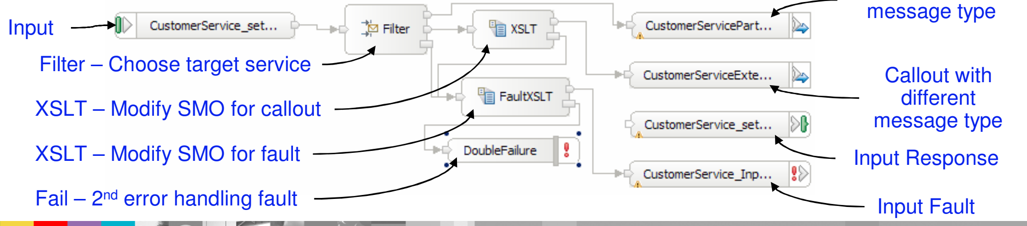
That completes an examination of the nodes, their terminals and associated message types.



Mediation flow definition - Connecting the nodes

- Mediation flows are defined by:
  - Adding mediation primitives to the flow for message processing
    - Mediation primitives have input, output and fail terminals
    - Just like nodes, specific message types are associated with each of the primitives terminals
  - Wiring a flow path to define the flow logic
    - Path will connect the left side nodes output terminals to right side nodes input terminals
    - The path is wired using the mediation primitives input, output and fail terminals
    - Two terminals must be of the same message type to be wired together
    - XSLT or Custom primitives can be used to modify message type when needed

Example: Two target services, one with different interface, fault handled

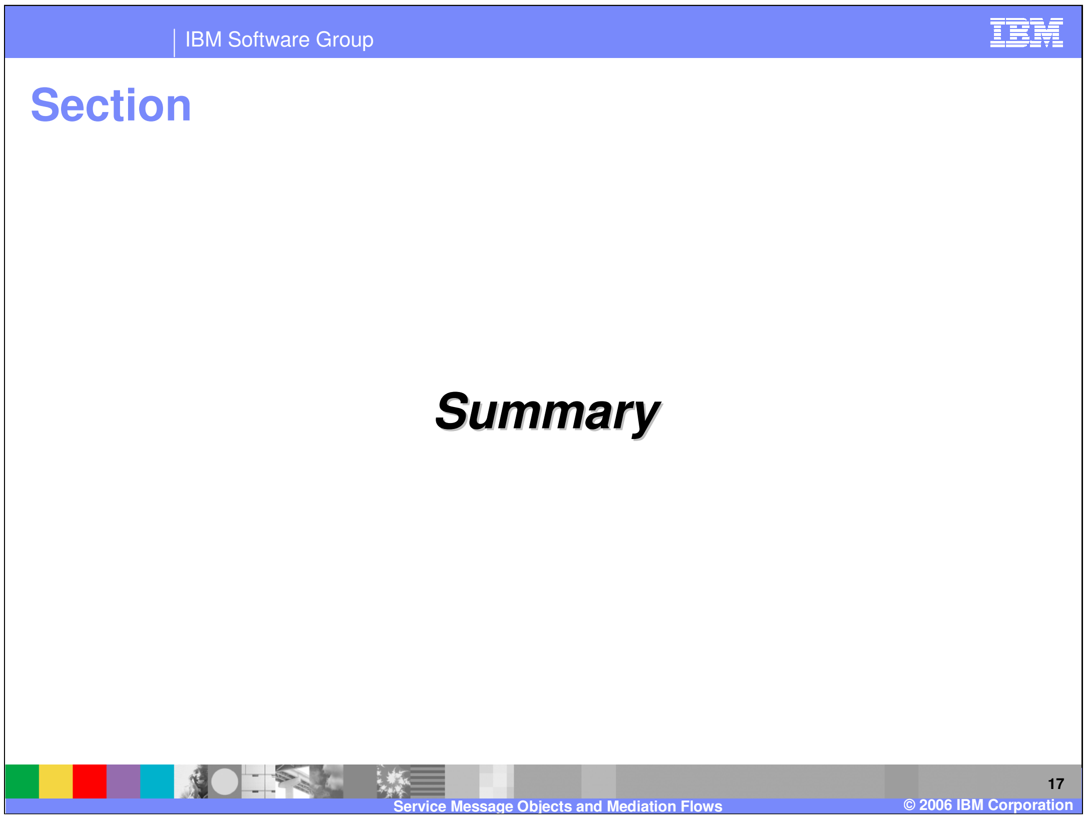

Input
CustomerService_set...
Filter
XSLT
FaultXSLT
DoubleFailure
CustomerServicePart...
CustomerServiceExte...
CustomerService_set...
CustomerService_Inp...
Filter – Choose target service
XSLT – Modify SMO for callout
XSLT – Modify SMO for fault
Fail – 2nd error handling fault
Callout with same message type
Callout with different message type
Input Response
Input Fault

The next step is to connect the nodes on the left side of the canvas to the nodes on the right side of the canvas by adding mediation primitives and wiring together a logical flow that performs the required mediation function. The mediation primitives have input, output and fail terminals. These terminals also must have a specific message type associated with them. As mentioned previously, when two terminals are wired together they must be of matching message type. Where there is a need to modify the type of message the XSLT or Custom Mediation primitives can be used to accomplish this.

The bottom of the slide contains a realistic example of a mediation flow. It illustrates the wiring of nodes and mediation primitives together while taking into account the constraint of only wiring terminals of like message type. This example shows two possible target service providers, one of which as a different interface than the service requestor. The flow also handles errors and returns a fault to the requestor if there is a failure in the flow.

Assume the input operation is called setVal1 and the output operation for the provider with the different interface is called setVal2. Starting with the input node on the left there is a wire to the Filter primitive. This wire connects terminals with a message type of setVal1RequestMsg. If the Filter determines the request should go to the callout node that supports the same interface and operation as the input there is no need to transform the message type. The wire connecting the Filter to the callout node connects terminals that also support a message type of setVal1RequestMsg. However, if the Filter determines the request should go to the callout node with the different interface and operation the message type needs to be changed using an XSL Transformation. The wire from the Filter to the XSLT would connect terminal types of setVal1RequestMsg and the wire from the XSLT to the callout would connect terminals with a message type of setVal2RequestMsg.

Assuming that an error occurs in either the Filter or the XSLT the flow would need to return a fault to the requestor using the input fault node. The input fault node's terminal is


IBM Software Group

Section

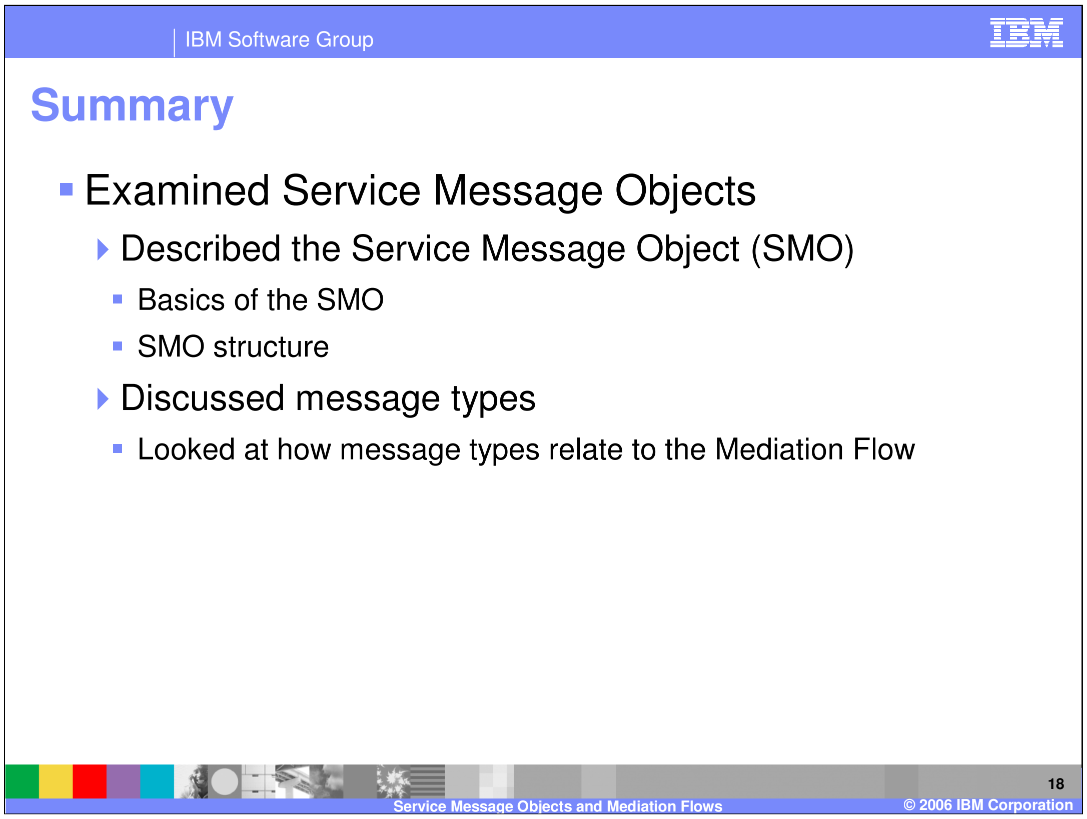
Summary

17

Service Message Objects and Mediation Flows

© 2006 IBM Corporation

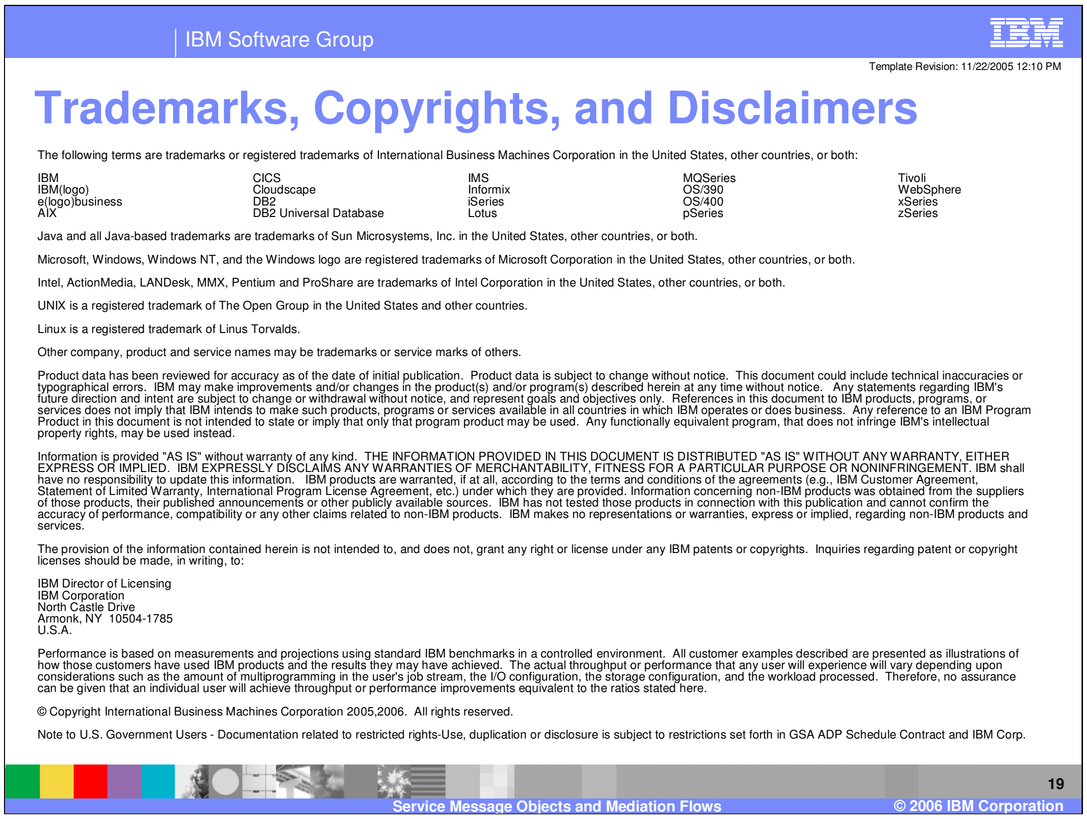
The presentation will conclude with a summary of what has been covered.



Summary

- Examined Service Message Objects
  - Described the Service Message Object (SMO)
    - Basics of the SMO
    - SMO structure
  - Discussed message types
    - Looked at how message types relate to the Mediation Flow

This presentation examined the use of Service Message Objects by first describing what an SMO is and how it is structured. The concept of message types was explained and a detailed description of how message types affect the construction of a mediation flow was given. Finally you were shown an example of a mediation flow illustrating how the message type affects the wiring of the flow.



Template Revision: 11/22/2005 12:10 PM

Trademarks, Copyrights, and Disclaimers