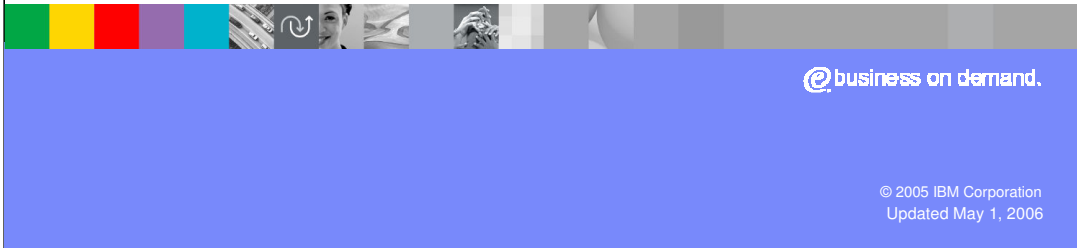




IBM Software Group

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

***ESB Concepts: Mediation Module, Flow  
Component and Primitives***



@business on demand.

© 2005 IBM Corporation  
Updated May 1, 2006

This presentation will provide an overview and detailed description of Mediation Module, containing the Mediation Flow component and the mediation primitives that process service messages as they pass through the Mediation Module.

## Agenda

- **Overview and ESB Concepts**
- Mediation Module
- Mediation Flow Component
- Mediation Primitives
- Unified Common Data Structure - SMO
- Mediation Module Deployment Package and Development to Deployment Life Cycle

The agenda of the presentation is shown here. The first section will provide an overview of the Mediation Module.

## Overview

- In a loosely coupled SOA architecture, Service requestors and providers connect with each other through an Enterprise Service Bus
- Loosely coupled services provide more flexibility and ability to introduce mediations and QOS that can then be applied uniformly to the services connecting through the bus
- Mediation services intercept and modify messages that are passed between existing services (providers) and clients (requesters) that want to use those services
- Mediation services are implemented using mediation modules that contain mediation flows.
- WebSphere ESB and Process Server provide the ESB capability through the use of Mediation Module deployed in the server
- Mediation Module uses the same Service component architecture (SCA) introduced in WebSphere Integration Developer V6.0.0 and WebSphere Process Server V6.0.0



In a service-oriented architecture, services represent business functions that can be reused and combined to create responsive business systems. These services can have loosely-coupled connections through an Enterprise Service Bus (ESB) rather than being connected directly to each other. Service requestors and providers connect to the ESB using the entry and exit end points

Introducing WebSphere ESB between services enables you to *mediate* between these services. Mediations intercept messages that are passed between services in order to provide a quality of service that can be uniformly applied to all the services using the bus.

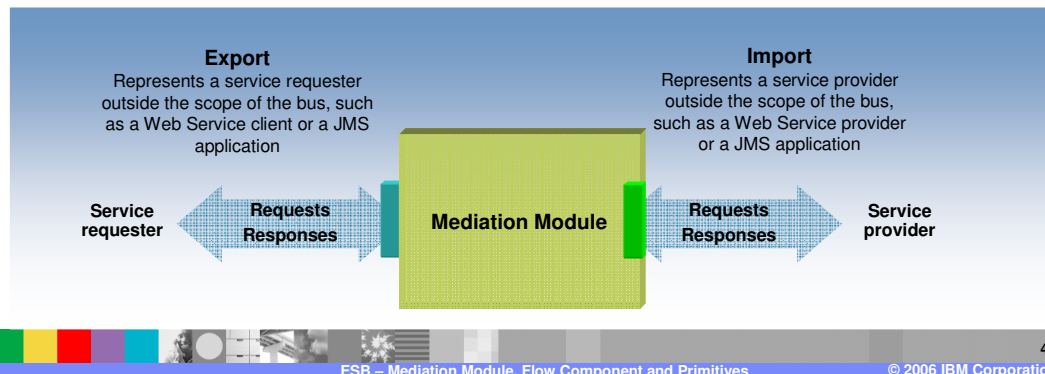
The new WebSphere ESB and upgraded WebSphere Process Server v6.0.1 includes the new capabilities of the ESB functionality through the use of Mediation Module, deployed in the server.

Mediation Module follows the same SCA concepts introduced in WebSphere Process Server v6.0.0.

Service requestors connect to the Mediation Module through the module exports. Calls to the external service provider through the bus are made using the module imports.

## ESB Concepts: Mediation Module

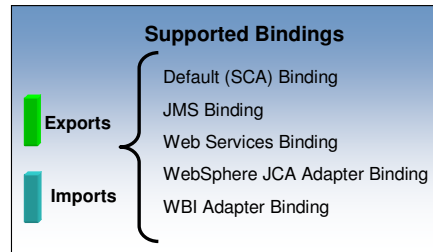
- WebSphere ESB and Process Server introduces a new type of module, called Mediation Module, that intercept and modify messages between service requester and the service provider
  - Mediation module provides the ESB functions of converting protocols, routing, transformation and other custom processing on the messages
- Mediation Module is the unit of deployment and runs within the WebSphere ESB or Process Server
- Interactions with external service requesters and providers defined by imports and exports, whose interfaces is defined using WSDL



Mediation Module is a new module introduced in WebSphere ESB and WebSphere Process Server to provide ESB functionality by allowing message processing between service requestors and providers. This module enables loosely coupled connectivity and mediation services between service requestors and providers connecting through the bus. The Mediation module allows protocol conversion, routing, transformation and other custom processing on messages typically required in an ESB environment. WebSphere Process Server supports business modules used for business processing and the new mediation modules, whereas WebSphere ESB supports mediation modules. Service requestors interact with the mediation module in the bus using module exports, and the module interacts with the service providers using module imports. Both the export and import interfaces are defined using WSDL.

## Mediation Module: Import and Export Bindings

- Different kinds of requester and provider types of interactions are made available via different *bindings* for the imports and exports
- WebSphere ESB provides support for
  - ▶ **JMS bindings** - JMS 1.1 provided by WebSphere Platform Messaging
    - Can exploit a variety of transports
      - TCP/IP, SSL, HTTP(S)
    - Allows interoperability with the WebSphere family
      - WAS, WebSphere MQ, WebSphere Message Broker
  - ▶ **Web Services binding**
    - SOAP/HTTP, SOAP/JMS, WSDL 1.1
    - Service Registry – UDDI 3.0
    - WS-Security, WS-Atomic Transactions
  - ▶ **WebSphere Adapter bindings**
    - JCA Adapters - SAP, PeopleSoft, Siebel, Files, JDBC
    - WBI Adapters for all the rest
  - ▶ **Built-in SCA (default) binding**
    - Used for module to module communication - supports both synchronous and asynchronous communication
    - WebSphere ESB supports update this binding via the admin console allowing module to module connectivity to be changed

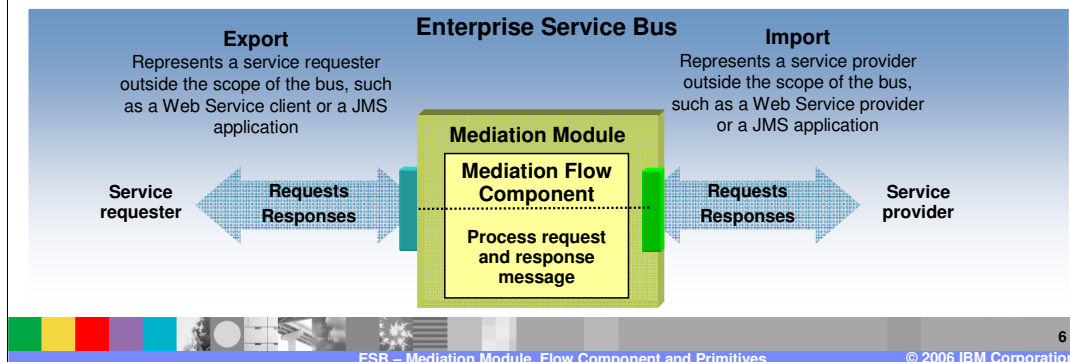


Interactions with external service requesters and providers are defined by imports and exports. Import and export interfaces are defined using Web Services Description Language (WSDL), which can contain several service *operations*. Various kinds of requesters and providers are made available using *bindings* for the imports and exports.

WebSphere ESB and WebSphere Process Server v6.0.1 support JMS binding, Web Services bindings, WebSphere Adapter bindings and the default, built-in SCA binding. These bindings allow maximum flexibility for requestors and providers to use the protocol of their choice. Use of different bindings also permits easy transformation of protocols between service requestors and providers. The import and export bindings are the same as those used for Business modules in WebSphere Process Server.

## Mediation Flow Component and Request-Response Interaction

- Mediation module contains a new type of SCA component, called Mediation Flow Component
- Mediation Flow Components act as 'service intermediaries' to
  - Pass a (potentially modified) request from a service requester to a service provider
  - Pass a (potentially modified) response from a service provider to a service requester
- Processing of requests is separated from processing of responses in the mediation flow component
- Request processing within a mediation flow component can send a response back to the requester without necessarily needing to contact a service provider



Mediation Module contains a new SCA component called Mediation flow component, which acts as a service intermediary for the processing of messages. The Mediation flow component provides a standard way of processing the message independent of the binding protocol used by the service requestors or providers, and supports a one way model where no response is expected, or a 2 way request and response model. It also supports an asynchronous or synchronous invocation model, similar to other SCA components.

Within the Mediation flow component, processing of the request message is performed separately from the response message, allowing different processing to occur on the request and the response side by having different mediation primitives on the request and response flows.

The mediation application developer can choose to create and handle the response within the mediation flow component without actually calling the service provider. The Mediation Module developer will need to construct the response message based on the interface definition of the module export.

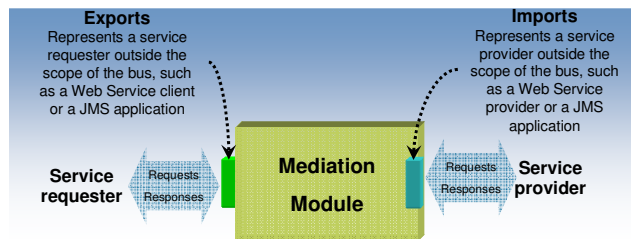
## Agenda

- Overview and ESB Concepts
- **Mediation Module**
- Mediation Flow Component
- Mediation Primitives
- Unified Common Data Structure - SMO
- Mediation Module Deployment Package and Development to Deployment Life Cycle

This section will provide more details concerning the Mediation Module.

## Mediation Module: Contents

- Mediation Module can have the following:
  - ▶ Exports, defined using WSDL, that expose the mediation module to external service requesters
  - ▶ Imports, defined using WSDL, that identify external service providers and their interfaces
  - ▶ A new type of SCA component called, Mediation flow component – this provides the *mediation function* on the messages between these services requestors and the providers
    - In cases, where the only need is to transform the message from one interaction protocol to another, there may not be any need for a mediation flow component in the module
  - ▶ Optional SCA Java components – this is used in conjunction with the custom mediation primitive or when there is a need to use Java interface



Mediation module contains exports, imports, a new type of SCA component called the Mediation flow component and optionally other SCA components.

Mediation Imports are like normal SCA imports with all the supported bindings, including the Default SCA, JMS, and Web Services, which serve as entry points into the Bus.

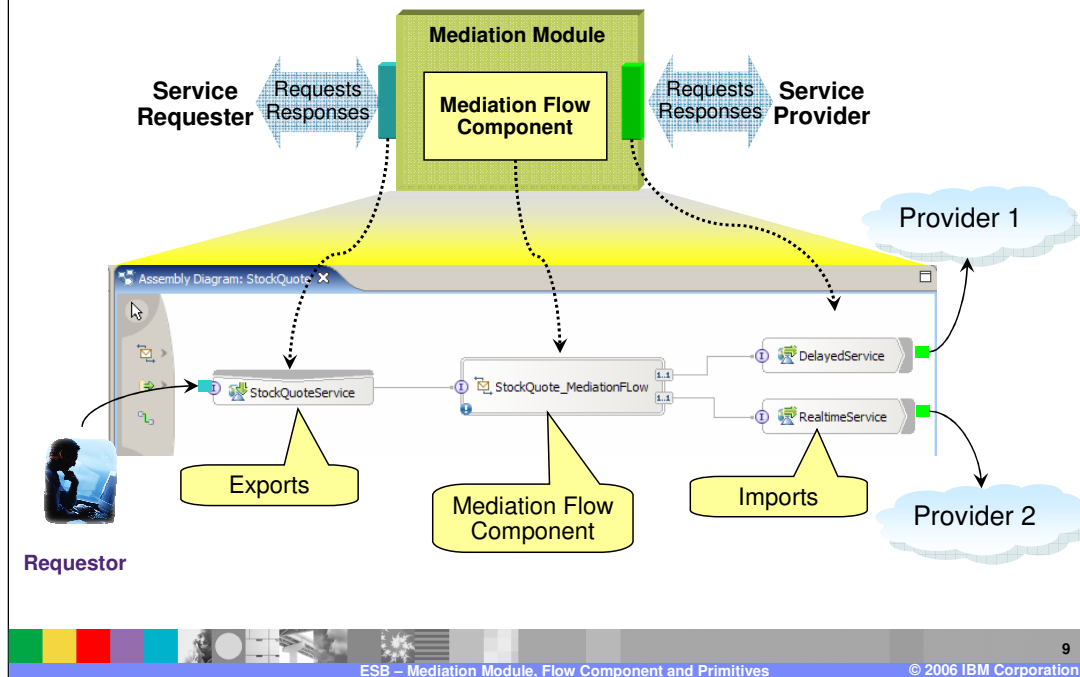
Similarly, Mediation Exports are like normal SCA exports with all the supported bindings, including the Default SCA, JMS, and Web Services, which serve as the exit points from the Bus.

A new type of SCA component, called the Mediation Flow component, contains logic for how the message is processed between the input and output of the flow. Functions like message routing, transformation, augmentation, logging or any other custom processing are performed on the message within the Mediation Flow component.

Lastly, the module can optionally contain SCA Java™ components, which are used to implement custom mediation primitives. This will be covered in greater detail later in the presentation.



## Mediation Module: Assembly Diagram Editor



WebSphere Integration Developer includes the Assembly Diagram editor, which is used to build the Mediation Module. The components shown in this diagram form the Mediation Module that runs within the Enterprise service bus of WebSphere ESB or WebSphere Process Server. It also contains the exports used by service requestors to send the service message to the bus and the Mediation Flow component, which is used to process the service message through the use of mediation primitives. Lastly, it shows the imports through which service providers are called. Each export and import are for a specific binding protocol and having multiple import or exports allows support of multiple bindings.

## Agenda

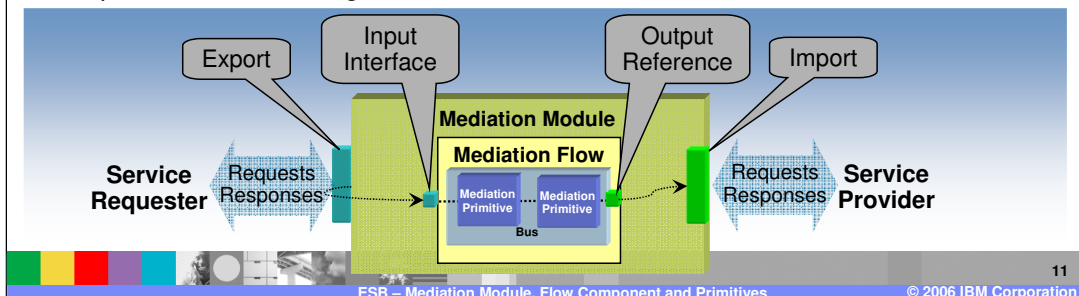
- Overview and ESB Concepts
- Mediation Module
- **Mediation Flow Component**
- Mediation Primitives
- Unified Common Data Structure - SMO
- Mediation Module Deployment Package and Development to Deployment Life Cycle



This section will provide more details related to the Mediation Flow component.

## Mediation Flow Component

- Mediation Flow component is where the request and response message is processed between the services
- Mediation flow component contains the following:
  - Connections between the message flow input (entry) and output (exit) operation
    - The message flow is created by the Integration Developer by visually wiring the input with the appropriate output operation
    - For N input operations, and M output operations, there are a total of NxM possible operation connections
  - For each connected flow from an input, zero or more Mediation primitives can be defined to act on the message, for the request flow as well as for the response flow
    - Primitives are wired together to create the message flow between the incoming message and the outgoing message
- Mediation Flow Editor allows creating visual connections of input and output operations, and adding Mediation Primitives



ESB – Mediation Module, Flow Component and Primitives

© 2006 IBM Corporation

The Mediation flow component is the main component of the Mediation Module, where the service message between the requestor and provider is processed. The inputs of the Mediation flow components receive the message that is passed by the requestor through the mediation module exports. Between the mediation flow component input and output are the mediation primitives, which process the message and perform routing, transformation, augmentation, logging and any other custom processing on the message.

## Mediation Flow Editor

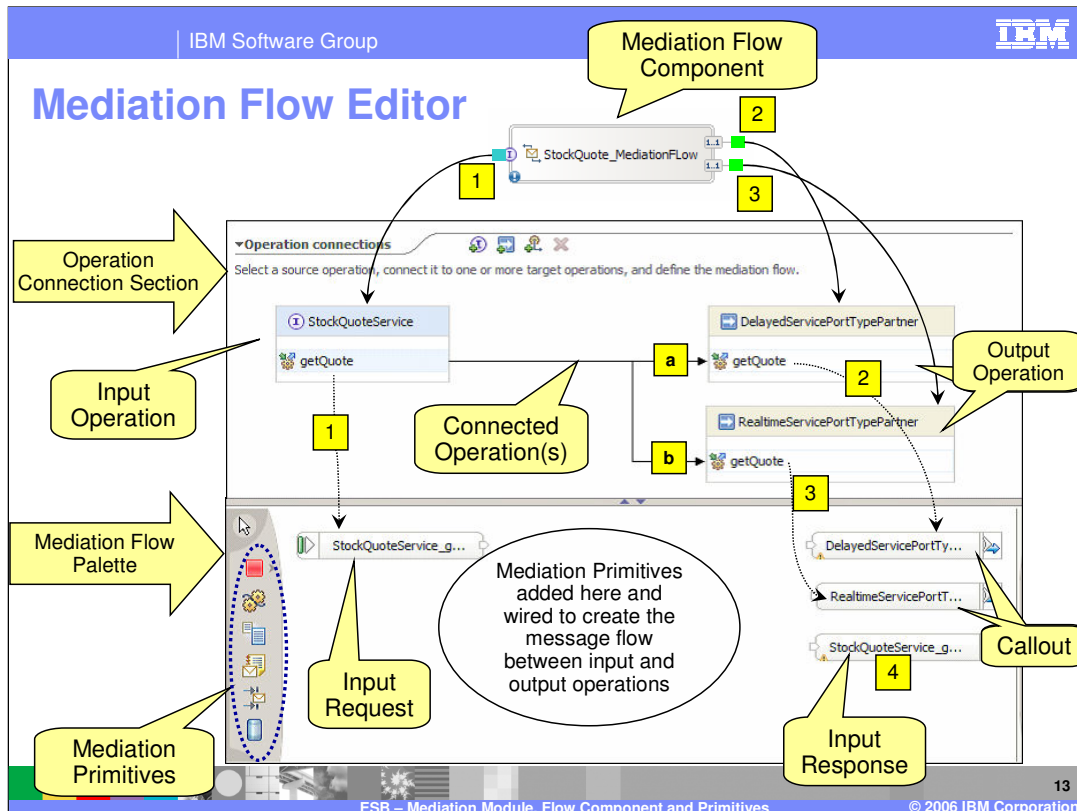
- Mediation Flow Component is implemented in the Mediation Flow Editor in WebSphere Integration Developer tool
- The Editor is divided into the 3 sections:
  - ▶ Operation Mediation Section – Top Section
    - Used to define the mapping of a source operation to one or more target operations
  - ▶ Message Flow Section – Middle section
    - Displays a flow diagram for a selected source operation per flow direction (Request and Response)
    - The user drag-n-drops mediation elements from the palette groups onto the canvas and wire them together
  - ▶ Mediation Properties Section – Bottom section
    - Displays a property sheet that shows the properties for each selected mediation primitives in the Message Flow section



The Mediation Flow editor is used to provide the implementation of the mediation components that are used to process the message flow as it flows from the service requestor to the service provider through the Enterprise Service bus. The editor contains the following 3 sections:

- The top section, the Operation Mediation section, is used to define the mapping of the source input operation to one or more target output operations. The map is created by visually wiring the input operation to the appropriate target out operation.
- The middle section, the Mediation Flow section, is used to create the message processing flow once the connection is made between a source and target operation. Mediation Primitives are added here and wired to create the message flow between the input and output operation.
- The bottom section of the editor, the Mediation Properties section, is used to view or modify the properties of the connection and primitives that are highlighted in the mediation flow section.

The next few slides provide diagrams of the different sections of the editor.



Examples of the top and middle sections of the Mediation Flow editor are shown here.

The top most box is the mediation flow component in the assembly diagram of the service module.

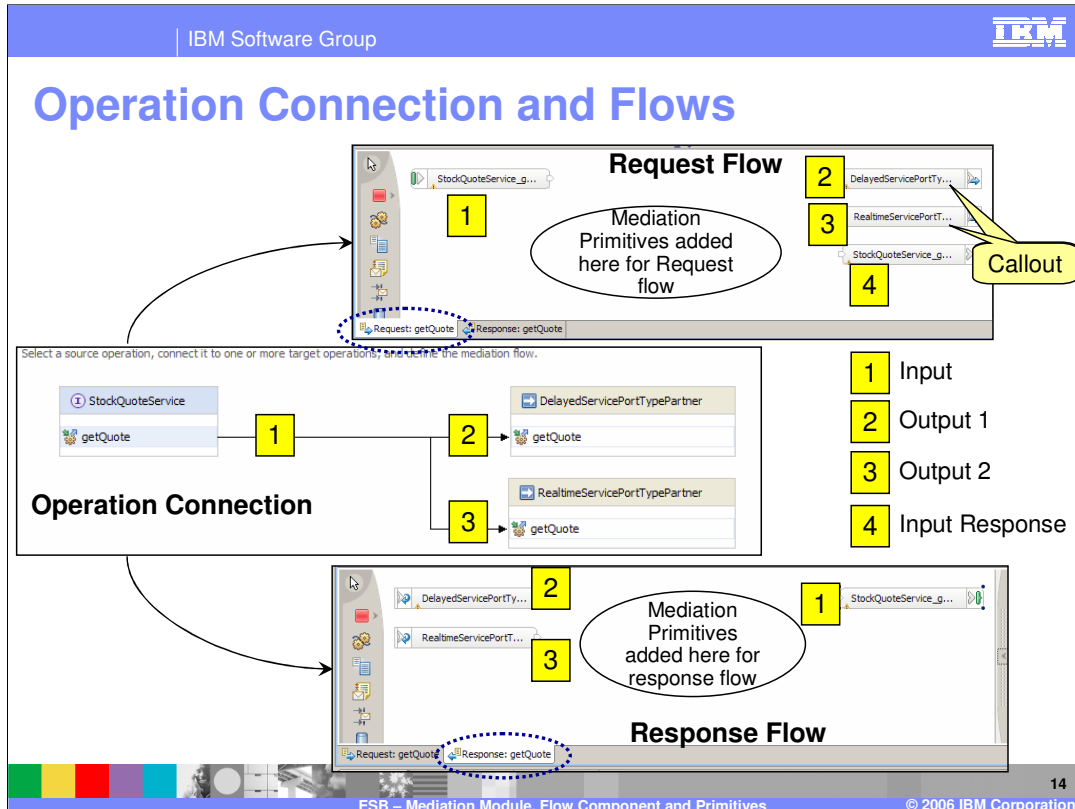
The import is represented by box 1 in the example. The example has 2 exports and 2 outputs shown by boxes 2 and 3.

The matching source input and target output operations are shown in the operation connection section and the mediation flow palette.

The 2 wired connections between the input and output operations are represented by boxes 'a' and 'b' in the operation connection section.

The mediation flow palette is where the necessary mediation primitives are added and wired to create the message flow and its processing between the input and output operations.

The Input response component, represented by box 4 in the diagram, is used to send back the response without going to any external provider. For example, the mediation primitives might create their own response on a user exception and send it back to the requestor. The developer can perform this action by wiring the appropriate output of the primitive to the Input response component. In this case, the message will not get forwarded to external providers, but processed by the primitives in the bus and sent back to the requestor. As previously noted, an appropriate response message will need to be



This page shows more details of the mediation flow editor. Specifically, the separate request and response flow, which can be selected as the tabs on the bottom of the mediation flow palette. The operation connection section is shown in the middle of the page. The top right box shows the request flow palette and the bottom right box shows the response flow palette.

## Mediation Flow Component Design Methodologies

- Two types of design methodology:
  - ▶ Top-down design
    - Developer creates with Mediation Flow component with the required interfaces and references
    - Developer generates an implementation (empty) for the Flow component
      - This will open the Mediation Flow component editor
    - Using the Mediation Flow Editor, the developer create mappings from a source operation to one or more target operations
  - ▶ Bottom-up design
    - User starts with actual implementation of the flow component – does not yet have the Mediation Flow component
    - The mediation flow component is then used to assemble the module
    - This approach can be used to modify any existing design and then merging the implementation of the flow component

Two design methodologies can be used when designing the mediation module within WebSphere Integration Developer.

•Using the top down design method, the developer creates the mediation flow component with the required interfaces and references, and then creates the implementation of the mediation flow component. The implementation is done by creating the mappings from the source operation to the target operation within the mediation flow component editor and then building the message flow between the source and the target operations using mediation primitives. This is done for the request flow as well as the response flow.

•Using the bottom-up design, the developer starts with the actual implementation of the mediation flow component and then uses it to build the mediation module. In this case, the developer must know the interfaces of their service requester (source) and service provider (target). The mediation flow component is then used to assemble the module, where the appropriate imports and exports are added to the module and connected to the flow component interfaces and references.

## Agenda

- Overview and ESB Concepts
- Mediation Module
- Mediation Flow Component
- **Mediation Primitives**
- Unified Common Data Structure - SMO
- Mediation Module Deployment Package and Development to Deployment Life Cycle

This section will provide details related to the Mediation primitives.



## Mediation Primitives

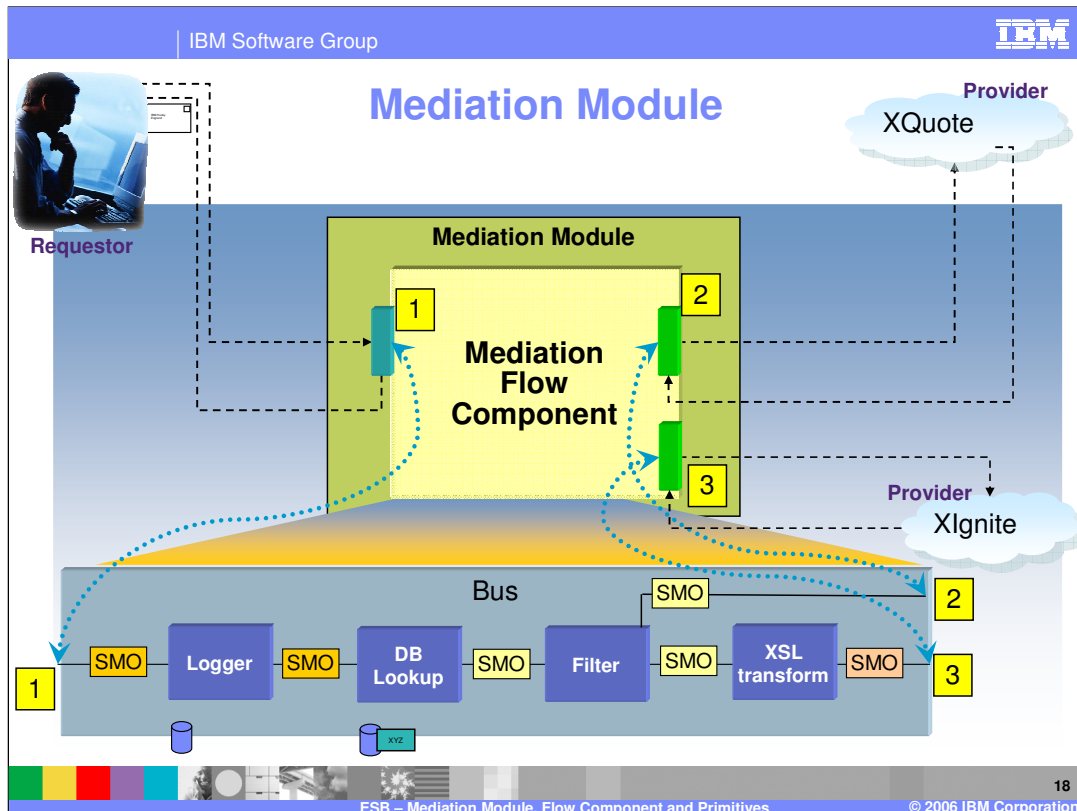
- Mediation primitives can perform routing, transformation, augmentation, logging or any other custom processing on the message
  - ▶ Several built-in mediation primitives are provided within the WebSphere Integration Developer tooling
  - ▶ Users can create Custom mediation primitives, implemented by Java class
- Mediation primitives have input/output terminals to accept or propagate messages
  - ▶ All primitives have one Input terminal
  - ▶ Most mediation primitives have one or more output terminals
    - There are 2 kinds of output terminals: Successful flows and Fail flows
    - There can be one or more successful output terminal, depending on the primitive
    - There is up to 1 fail output terminal used when exception occurs – the fail terminal propagates the original message, together with any exception information
- Input and Output messages to the Mediation primitives are represented as Service Message Objects (SMO)

Mediation Primitives are the core building blocks used to process the request and the response message.

WebSphere ESB and the tooling provide several function built-in primitives, including logger, message filter and others. Some details of the built-in primitives are discussed later in this presentation. Support for a custom mediation primitive is also provided for use by developers where the built-in primitives do not provide the required functionality. Custom mediation logic is implemented using the SCA Java™ component and more detail concerning this is provided in a separate presentation.

Mediation primitives have input and outputs, known as terminals and one input where the input message enters the mediation primitive. Mediation primitives also have 2 kinds of output terminals. The first type of output terminal is for a successful flow with no exceptions. Depending on the mediation primitive, there can be more than one output success terminal, as in the case of the built-in Message filter mediation primitive. The other type of output flow is for flows that fail due to exceptions.

The input and output messages for all the mediation primitives are represented as Service Message Objects or SMO. The SMO interface extends the DataObject interface, which is defined by Service Data Object (SDO). More on SMO later in this presentation.



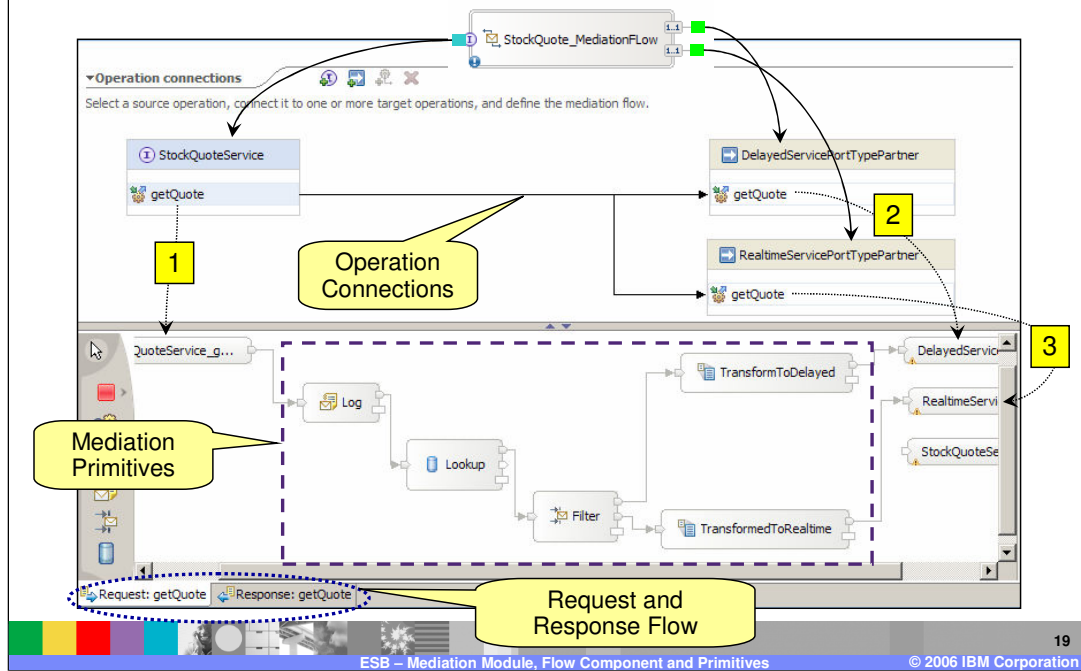
Shown here is an example of the different mediation primitives used in a message flow from a requestor to one or more providers.

The 1<sup>st</sup> primitive shown is the built-in message logger used to log the message in a database.

When the request passes through the Mediation Module, a DataBaseLookup Primitive is used to look up the userid in a table of “gold” users, then a Filter is applied.








If the user is a gold customer, the request will be passed to a different external Web Service (offered by Xignite), otherwise it will continue as before. Because the message formats differ, the request is first passed through an XSLT Primitive that will transform the message using an XPath expression. The response from Xignite must also be transformed as part of the Mediation so that the format is as expected.

## Mediation Flow Editor With Primitives



Shown here is a snapshot of the mediation flow editor with the primitives for the example shown on the previous page. The example shows only the request message flow.

## Mediation Primitives – Overview

Mediation Primitives	Symbol	Description
<b>Message Logger</b>		To log/store message information to a database
<b>Message Filter</b>		To filter messages selectively forwarding them on to output terminals, based on simple condition expression
<b>Database Lookup</b>		To access information in a database and store it in the message
<b>XSLT</b>		To manipulate or transform messages using XSL transformation
<b>Stop</b>		To stop a path in the flow, without generating an exception
<b>Fail</b>		To stop a path in the flow, and generate an exception
<b>Custom</b>		For custom processing of message. It uses a SCA Java component for custom message processing

WebSphere ESB provides several built-in mediation primitives and allows the capability of adding your own custom mediation for cases that are not covered by the built-in mediation primitives.

The following built-in mediation primitives are provided:

- Message Logger is used to log and store message information to a database.
- Message Filter is used to filter messages and selectively forward them on to output terminals, based on simple condition expressions.
- Database lookup is used to access information in a database and insert it in the message. The mediation primitive is supplied with the key id to look for, and where the value of the key is located within the message. Using these two pieces of information, the value of the specified column for the matching key is inserted in the specified location within the message.
- XSL Transformation mediation primitive is used to transform messages using XSL transformation. This is mainly used when the target provider has a different interface than the incoming message interface. Using the mapping within the XSLT, input values can be mapped to the appropriate output fields.
- Stop mediation primitive is used to stop the flow execution.
- Fail mediation primitive is used for error conditions where the flow execution is stopped and an exception is generated.

Custom mediation primitive is used to perform message processing that is not covered by other mediation primitives by executing custom logic. Custom Mediation Primitive calls an SCA Java component that you create or provide. The SCA Java component must be within the same Mediation module.

Details of each of the Mediation Module primitives and the custom primitive are covered in another presentation focused on the details of mediation primitives.

## Agenda

- Overview and ESB Concepts
- Mediation Module
- Mediation Flow Component
- Mediation Primitives
- **Unified Common Data Structure - SMO**
- Mediation Module Deployment Package and Development to Deployment Life Cycle

This section will cover the unified data structure, namely the Service Message Object or SMO used within the Mediation primitives in the Mediation Flow component.

## Use of Common Data Structure in the Flow - SMO

- The Mediation module's import and export binding interacts with some specific service provider type with its own data representation (e.g. Web Services, JMS and so on)
- The data from this binding-specific interaction is turned into a common data structure called the Service Message Object (SMO)
- SMO provides a common representation of message data flowing through the ESB for the Mediation programming environment
- SMO is implemented as a SDO (Service Data Object), similar to the common data representation for all data objects in WebSphere Process Server
- SMO provides a unified view of the message payload (a Business Object), message headers, and context information
- SMO provides interface to access/modify the SMO content
  - ▶ SMO and its embedded contents can be accessed using the SDO XPath reference mechanism

22

ESB – Mediation Module, Flow Component and Primitives

© 2006 IBM Corporation

The service requestor and service provider interact with the bus through the bindings for the exports and imports of the mediation service module. The data representing the message depends on the binding used for exports and imports. If the primitives in the mediation flow component had to support the data representation for all the various bindings, it would be difficult to implement primitives.

For this reason, the first thing the runtime does is convert the binding-specific data into common data structure, called Service Message Objects (SMO).

SMO interface extends the DataObject interface, which is defined by Service Data Object (SDO), similar to other business objects used in WebSphere Process Server.

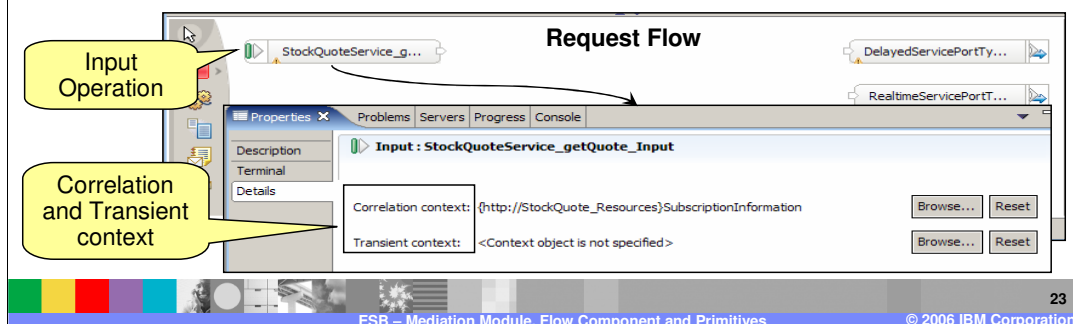
SMO includes the message headers, message data (payload) and context information and provides an interface to access and modify the SMO data, including headers, payload and context information. In addition, the SDO data can be accessed using the XPath reference mechanism.

The Mediation module input contains binding specific data representation. The mediation module output import binding dictates the data representation that must be sent to the output message. As a result, there is a lack of consistency in data representation and if the mediation flow primitives had to handle all the different data representation, it would become a huge challenge. To solve this problem, the data from binding-specific interaction is converted to a common data structure called the Service Message Object (SMO). At the Mediation Flow input boundary, the input binding specific interaction data is converted to SMO. At the Mediation Flow output boundary, the SMO is converted to the output binding-specific interaction

The SMO is represented as a Service Data Object (SDO) and provides the Mediation API to access or modify the SMO content. The XPath reference mechanism can be used to access the SMO content.

## Correlation and Transient Context

- Correlation and Transient context are like scratch pad used by mediation primitives to store data to be used later by different mediation primitives within a mediation flow
- Transient data is available only on the request or response flow but not both
  - ▶ Stored in memory
- Correlation data is available for the duration of the complete request/response flow
  - ▶ For Synch request/response, data is stored in memory
  - ▶ For Asynch request/response, data is stored in a Queue where different QOS can be applied
- Correlation and Transient context data are represented as a Data object and are defined at the start of the mediation flow
  - ▶ Defined at the input operation property panel for the Request flow or response flow, as shown below



If the mediation primitives in a message flow require a temporary area to save data for other primitives down the message flow or need data set in the request flow to be available during the response flow, context information is used as a scratch pad.

There are 2 types of context information:

Transient context, as the name suggests, is temporary and available only on the specific request flow or the response flow but is not carried from the request to the response flow, and is therefore stored in memory.

Correlation context data is available for the duration of the complete request/response flow. Data set in the request flow is available for all the primitives in the response flow.

Any primitive can modify the context information and downstream primitives in the message flow will have access to that information.

The context data are represented as data objects and the structure is inserted in the SMO structure at the start of the mediation flow.

In the mediation flow editor, the context information is defined at the input operation of the flow within the property panel, as shown in the diagram.

## Agenda

- Overview and ESB Concepts
- Mediation Module
- Mediation Flow Component
- Mediation Primitives
- Unified Common Data Structure - SMO
- Mediation Module Deployment Packaging and Development to Deployment Life Cycle

This section covers the Mediation Module deployment package and application development to deployment life cycle.



## Mediation Module: Deployment Packaging

- Mediation Module deployment package is similar to Business Module Deployment package
- WebSphere Integration Developer generates the deployment artifacts for the Mediation Module and creates a J2EE EAR file
- Mediation Module J2EE EAR file is installed in the WebSphere ESB or Process Server using Administrative Console or command line wsadmin tool

The packaging of the mediation Module is similar to the business application used in WebSphere Process Server.

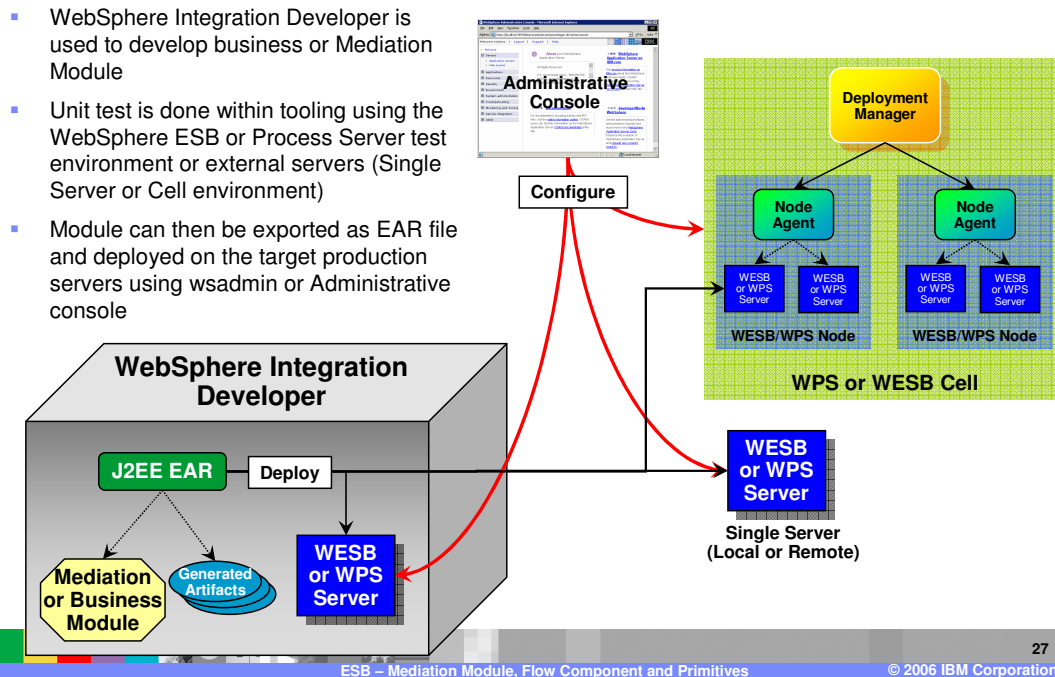
The WebSphere Integration Developer tooling generates all the J2EE™ artifacts for the mediation application and creates the J2EE EAR file that is then deployed in the WebSphere ESB or WebSphere Process Server using the normal install process in the Administrative console or using the command line wsadmin tool.

## Bus Resources Created for Mediation Module

- Service integration resources, such as bus destinations, are created when installing a mediation module - the bus destinations are used to hold the messages for the service components
- Following bus destinations are created on the bus called `SCA.SYSTEM.cell_name.Bus`
  - ▶ Queue `sca/module_name`
  - ▶ Queue `sca/module_name/export/export_name`
  - ▶ Queue `sca/module_name/exportlink/export_name`
  - ▶ Queue `sca/module_name/import/import_name`
  - ▶ Queue `sca/module_name/importlink/import_name`
  - ▶ Queue `sca/module_name/import/sca/dynamic/import/scaimport` [for SCA binding]
  - ▶ Queue `sca/module_name/import/sca/dynamic/import/wsimport` [for Web service binding]
  - ▶ Queue `sca/module_name/component/component_name`
  - ▶ Queue `sca/module_name/component/component_name/source/source_name`
  - ▶ Queue `sca/module_name/component/component_name/target/target_name`

Several service integration bus resources and destinations are created when deploying a mediation module in WebSphere ESB or WebSphere Process Server. The names used for the bus destination resources follow a naming convention that uses the mediation module name. Some of these resources are shown here. These destinations are used by the mediation module to pass asynchronous messages. When the mediation module is uninstalled, these destination resources are removed.

## Development, Testing and Deployment Cycle



This diagram shows the end to end development, testing and deployment lifecycle of both the business and mediation modules.

WebSphere Integration Developer is used for development of the applications. Testing can be done using the WebSphere ESB or WebSphere Process Server internal test server, or external WebSphere ESB or WebSphere Process Server on the same or a remote system. Additionally, the external servers could be single server or a Network Deployment cell environment. The administration of the internal test or the external server is done using the normal WebSphere ESB or WebSphere Process Server administration tools, namely, the Administrative console or the wsadmin command line.

The applications are exported as regular EAR files containing SCA module files, which can then be handed to the system administrator for deployment on the production server.

## Section

# ***Summary and References***

This section will provide a summary of this presentation.

## Summary

- WebSphere ESB and Process Server both provide the ESB capability
- Mediation Modules provide the loosely coupled connectivity for the service requestor and provider, along with mediation and other QOS functionality
- Mediation Module follows the same SCA model architecture introduced in WebSphere Integration Developer V6.0.0 and WebSphere Process Server V6.0.0 – It contains a new SCA component, called Mediation Flow component
- A Mediation Flow contains zero or more mediation primitives used to process the service message to perform routing, transformation, augmentation or any other custom processing on the message

In summary, the mediation module provides ESB functionality for both WebSphere ESB and WebSphere Process Server and follows the same SCA model architecture introduced in WebSphere Integration Developer and WebSphere process Server V6.0.0. Mediation Module contains mediation flow components that process the request and response message using mediation primitives. This presentation provided some detail concerning the mediation module, flow components and primitives in order to give you an idea of how to build the mediation module.

## Trademarks, Copyrights, and Disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM	CICS	IMS	MQSeries	Tivoli
IBM (logo)	Cloudscape	Informix	OS/390	WebSphere
e(logo)/business	DB2	iSeries	OS/400	xSeries
AIK	DB2 Universal Database	Lotus	pSeries	zSeries

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

Microsoft, Windows, Windows NT, and the Windows logo are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds.

Other company, product and 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 and/or changes in the product(s) and/or program(s) 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.

Information is provided "AS IS" without warranty of any kind. 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 NON-INFRINGEMENT. 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 (e.g., 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.

© Copyright International Business Machines Corporation 2004,2005. All rights reserved.

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.