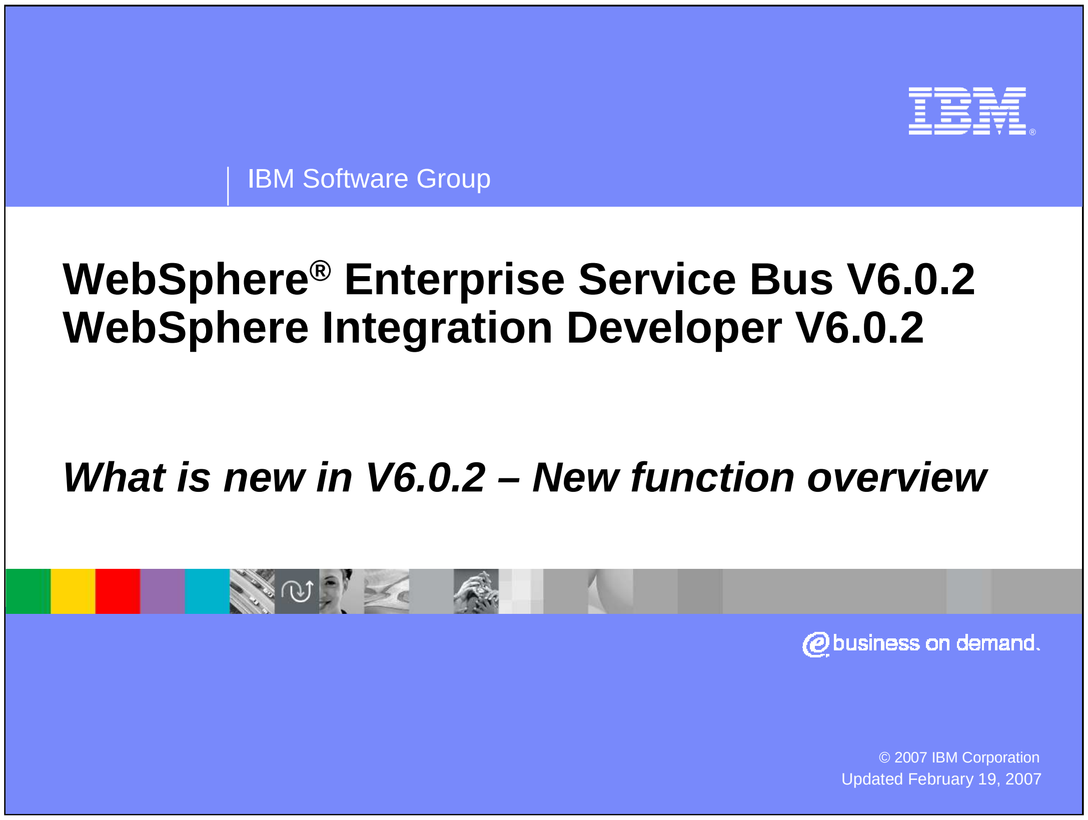


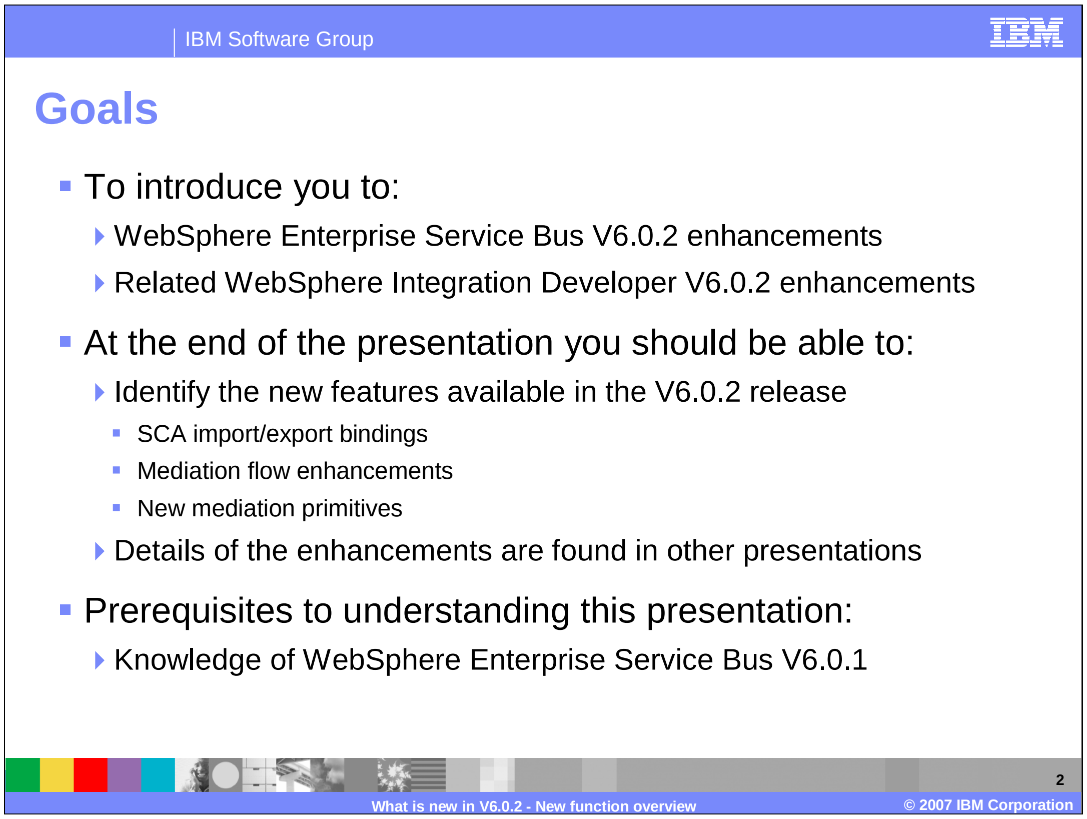
IBM Software Group

WebSphere® Enterprise Service Bus V6.0.2 WebSphere Integration Developer V6.0.2

What is new in V6.0.2 – New function overview

business on demand.

This presentation provides an introduction to the new function that has been added to WebSphere Enterprise Service Bus in version 6.0.2.



IBM Software Group

Goals

- To introduce you to:
  - WebSphere Enterprise Service Bus V6.0.2 enhancements
  - Related WebSphere Integration Developer V6.0.2 enhancements
- At the end of the presentation you should be able to:
  - Identify the new features available in the V6.0.2 release
    - SCA import/export bindings
    - Mediation flow enhancements
    - New mediation primitives
  - Details of the enhancements are found in other presentations
- Prerequisites to understanding this presentation:
  - Knowledge of WebSphere Enterprise Service Bus V6.0.1

2

What is new in V6.0.2 - New function overview

© 2007 IBM Corporation

The goal of this presentation is to introduce you to the enhancements that have been made to WebSphere Enterprise Service Bus in version 6.0.2. This includes new and changed capabilities in WebSphere Integration Developer version 6.0.2 which support the WebSphere Enterprise Service Bus enhancements from a development tool perspective.

At the end of the presentation you should be able to identify the new features in WebSphere Enterprise Service Bus version 6.0.2. These enhancements fall into three categories.

The SCA import and export binding enhancements include the support for all JMS body types, bindings for native WebSphere MQ and also bindings for JMS over WebSphere MQ.

The Mediation flow enhancements include dynamic endpoint selection, handling of unmodeled faults, runtime administration of mediation properties and improved configuration of custom mediation primitives.

The new mediation primitives which are being introduced are the event emitter primitive, the message element setter primitive and the endpoint lookup primitive.

These enhancements are just being introduced here and you will find more detailed information on these enhancements in other presentations.

In order to understand the material being presented in the following slides you should already have a knowledge of the capabilities of WebSphere Enterprise Service Bus version 6.0.1.

IBM Software Group

Section

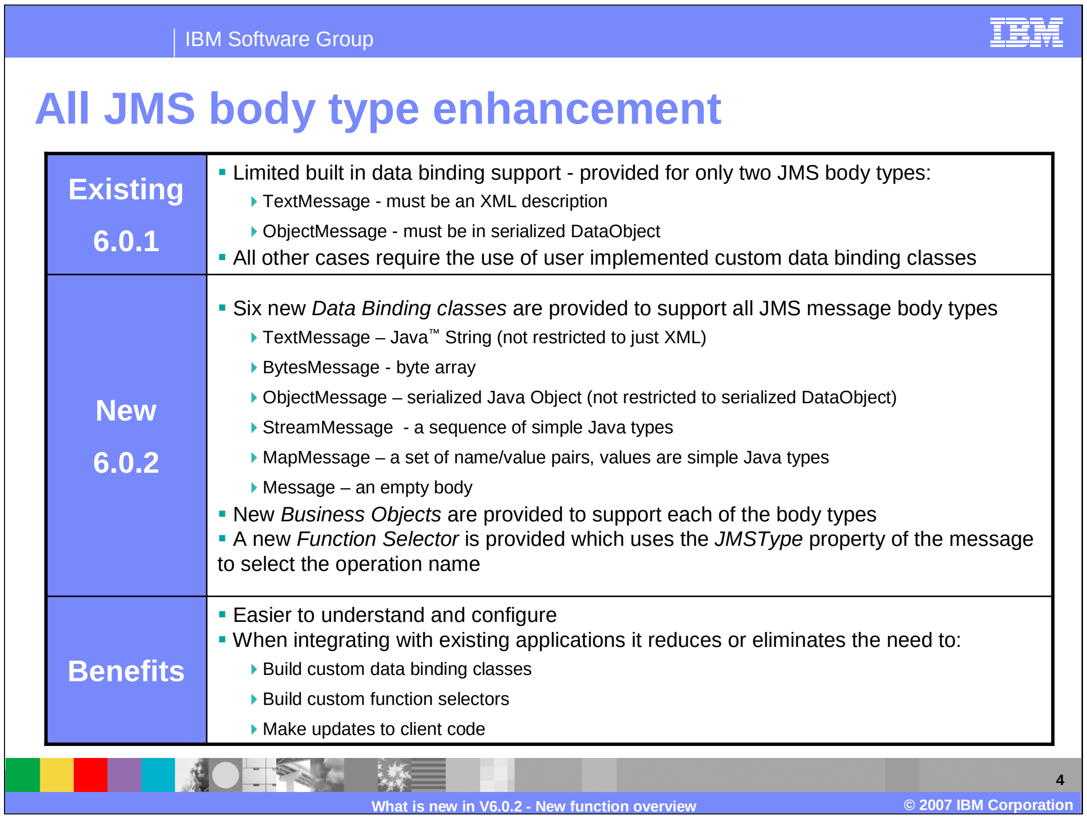
SCA import and export binding enhancements

3

What is new in V6.0.2 - New function overview

© 2007 IBM Corporation

This section will introduce you to the SCA import and export binding enhancements. These enhancements include improvements to the existing JMS bindings to provide built in support for all of the JMS body types. They also include two new binding types, one that works directly with WebSphere MQ queue managers and clients and the other that supports interfacing with JMS running over WebSphere MQ.


IBM Software Group

All JMS body type enhancement

Existing 6.0.1	▪ Limited built in data binding support - provided for only two JMS body types: ‣ TextMessage - must be an XML description ‣ ObjectMessage - must be in serialized DataObject ▪ All other cases require the use of user implemented custom data binding classes
New 6.0.2	▪ Six new *Data Binding classes* are provided to support all JMS message body types ‣ TextMessage – Java™ String (not restricted to just XML) ‣ BytesMessage - byte array ‣ ObjectMessage – serialized Java Object (not restricted to serialized DataObject) ‣ StreamMessage - a sequence of simple Java types ‣ MapMessage – a set of name/value pairs, values are simple Java types ‣ Message – an empty body ▪ New *Business Objects* are provided to support each of the body types ▪ A new *Function Selector* is provided which uses the *JMSType* property of the message to select the operation name
Benefits	▪ Easier to understand and configure ▪ When integrating with existing applications it reduces or eliminates the need to: ‣ Build custom data binding classes ‣ Build custom function selectors ‣ Make updates to client code

4

What is new in V6.0.2 - New function overview

© 2007 IBM Corporation

This enhancement provides support in SCA imports and exports for all of the message body types defined by the Java Message Service, also known as JMS.

In version 6.0.1 only two of the JMS body types had any built in support. Even the support that was provided was limited in nature. The TextMessage body type was supported only if the text it contained was XML and the ObjectMessage body type was supported only in the case where the serialized object it contained was a serialized DataObject. In all other cases you were required to implement your own custom data binding class.

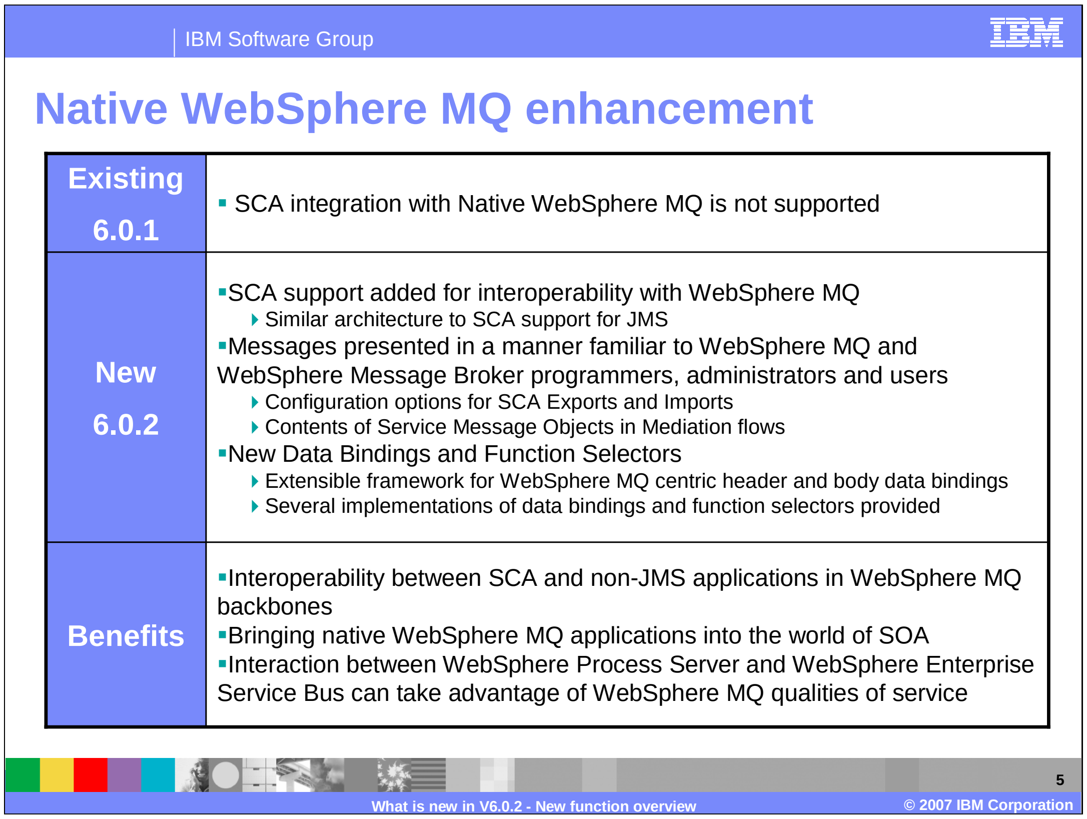
The enhancement in version 6.0.2 provides new data binding classes and new business objects in support of all the body types. It also provides a new function selector class which keys off of the body type of the message.

The purpose of a data binding with SCA JMS imports and exports is to convert between the JMS message encoding and the business object representation of the data. By providing data bindings and business objects for all of the body types, it is possible for you to handle any JMS message without coding any data bindings or defining any business objects.

The purpose of the function selector used with the export is to specify which operation on the interface should be called. The implementation of this function selector defines unique operations to be called for each of the body types. This is based on the JMSType field passed in the JMS message header which is one of the standard fields in any JMS message. You can define an interface which uses the unique operation names returned by this function selector and provide an implementation for each operation which is specific to the body type of the message.

These enhancements are independent of each other. In some situations you might only need to use the data binding, in other situations you might only need the function selector and in some cases you can use them together.

Among the benefits of these enhancements, you can reduce or eliminate the need to develop custom data binding classes and custom function selectors. Because the function selector is keying on a standard JMS header field it reduces the occasions where you might have to update the client which is sending the JMS message.


IBM Software Group

Native WebSphere MQ enhancement

Existing 6.0.1	▪ SCA integration with Native WebSphere MQ is not supported
New 6.0.2	▪SCA support added for interoperability with WebSphere MQ ▸ Similar architecture to SCA support for JMS ▪Messages presented in a manner familiar to WebSphere MQ and WebSphere Message Broker programmers, administrators and users ▸ Configuration options for SCA Exports and Imports ▸ Contents of Service Message Objects in Mediation flows ▪New Data Bindings and Function Selectors ▸ Extensible framework for WebSphere MQ centric header and body data bindings ▸ Several implementations of data bindings and function selectors provided
Benefits	▪Interoperability between SCA and non-JMS applications in WebSphere MQ backbones ▪Bringing native WebSphere MQ applications into the world of SOA ▪Interaction between WebSphere Process Server and WebSphere Enterprise Service Bus can take advantage of WebSphere MQ qualities of service

5

What is new in V6.0.2 - New function overview © 2007 IBM Corporation

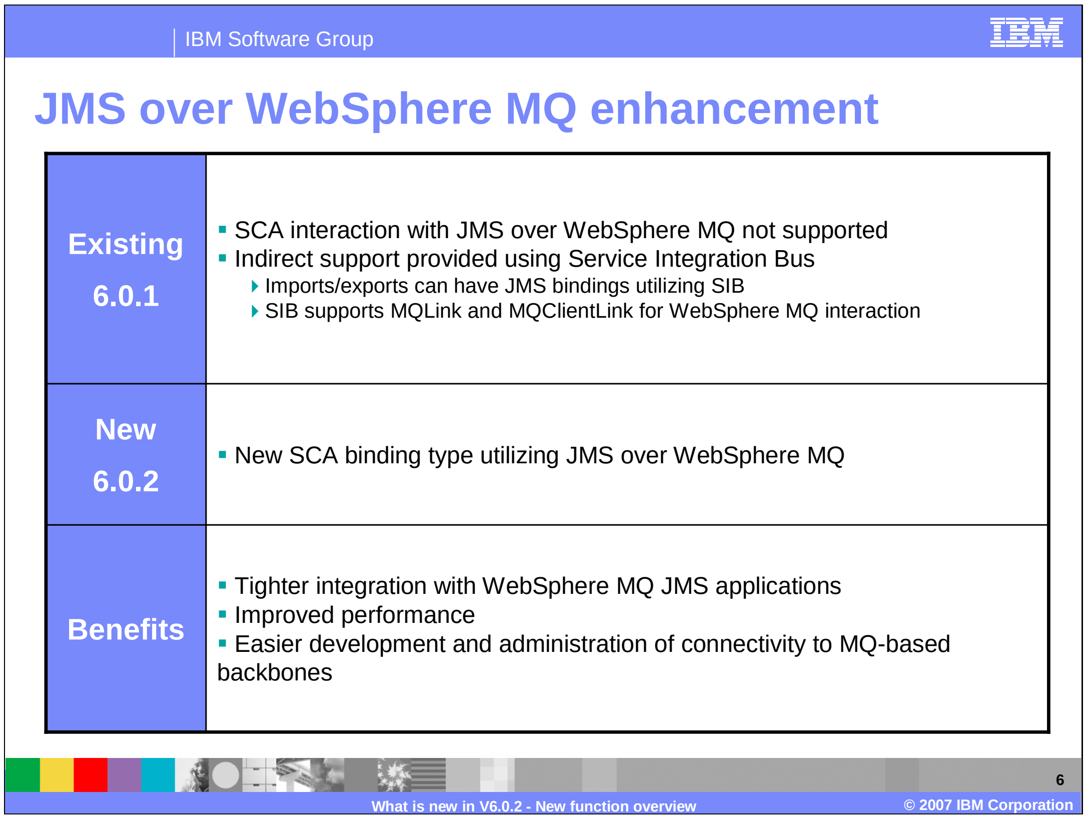
This enhancement provides support in SCA imports and exports to allow them to directly interact with WebSphere MQ queue managers and clients.

In version 6.0.1 the ability for SCA imports and exports to send and receive messages from WebSphere MQ was not supported.

In version 6.0.2 support has been added to allow SCA imports and exports to be configured to interact directly with WebSphere MQ queue managers and clients. The SCA architecture for doing this is similar to how the JMS support is implemented, using data bindings for conversion of the message data and function selectors to determine the operation to call for an incoming message.

There are characteristics of the WebSphere MQ architecture that are familiar to those who have programmed or administered WebSphere MQ environments. The structure and use of multiple headers and the conventions used for message correlation are a couple of examples. The design of this support in version 6.0.2 allows those characteristics to show through in a variety of ways, thus making the support understandable to you if you have a WebSphere MQ background. These characteristics show in the configuration options for the imports and exports. They also show in the WebSphere MQ header structure of the service message object used in mediation flows. The new data bindings and function selectors provided are based on an extensible framework which encapsulates much of the WebSphere MQ specific knowledge. This make it easier for you to develop your own data bindings and function selectors if the predefined ones do not meet your needs.

There are several benefits obtained from this support. It allows interoperability between SCA applications and non-JMS applications running in WebSphere MQ. Therefore, the door is opened for existing WebSphere MQ applications to become integrated into service oriented architecture environments. Another way this can be used is enabling WebSphere MQ to handle the messaging between WebSphere Process Server and WebSphere Enterprise Service Bus instances so that they can take advantage of the qualities of service WebSphere MQ provides.


IBM Software Group

JMS over WebSphere MQ enhancement

Existing 6.0.1	▪ SCA interaction with JMS over WebSphere MQ not supported ▪ Indirect support provided using Service Integration Bus ‣ Imports/exports can have JMS bindings utilizing SIB ‣ SIB supports MQLink and MQClientLink for WebSphere MQ interaction
New 6.0.2	▪ New SCA binding type utilizing JMS over WebSphere MQ
Benefits	▪ Tighter integration with WebSphere MQ JMS applications ▪ Improved performance ▪ Easier development and administration of connectivity to MQ-based backbones

6

What is new in V6.0.2 - New function overview

© 2007 IBM Corporation

This enhancement provides support in SCA imports and exports to allow them to directly interact with JMS running over WebSphere MQ.

In version 6.0.1 the capability to have an SCA import or export interact directly with WebSphere MQ JMS was not supported. It was possible to configure an SCA import or export to interact with the built in service integration bus and then configure the service integration bus to interact with WebSphere MQ using MQLink or MQClientLink. However, this approach had limitations, performance implications and complexities in configuration.

In version 6.0.2 a new SCA binding type which makes use of JMS running over WebSphere MQ has been added. This binding is similar in many ways to the JMS binding that utilizes the service integration bus. It makes use of the same data bindings, business objects and function selector discussed in a previous slide. However, the implementation uses the WebSphere MQ JMS provider based on JMS 1.1 whereas the JMS binding implementation is based on a J2EE Connector Architecture implementation. This leads to some differences you will see in how the imports and exports are configured for the two different types of JMS bindings.

In addition, there are some WebSphere MQ specific configuration options that can be specified, such as compression and connection pooling.

Among the benefits of this enhancement is the resulting tighter integration with WebSphere MQ JMS applications, along with better performance and easier configuration and administration when connecting to WebSphere MQ based backbones.

IBM Software Group

Section

Mediation flow enhancements

7

What is new in V6.0.2 - New function overview

© 2007 IBM Corporation

In this section you will be introduced to the mediation flow enhancements.

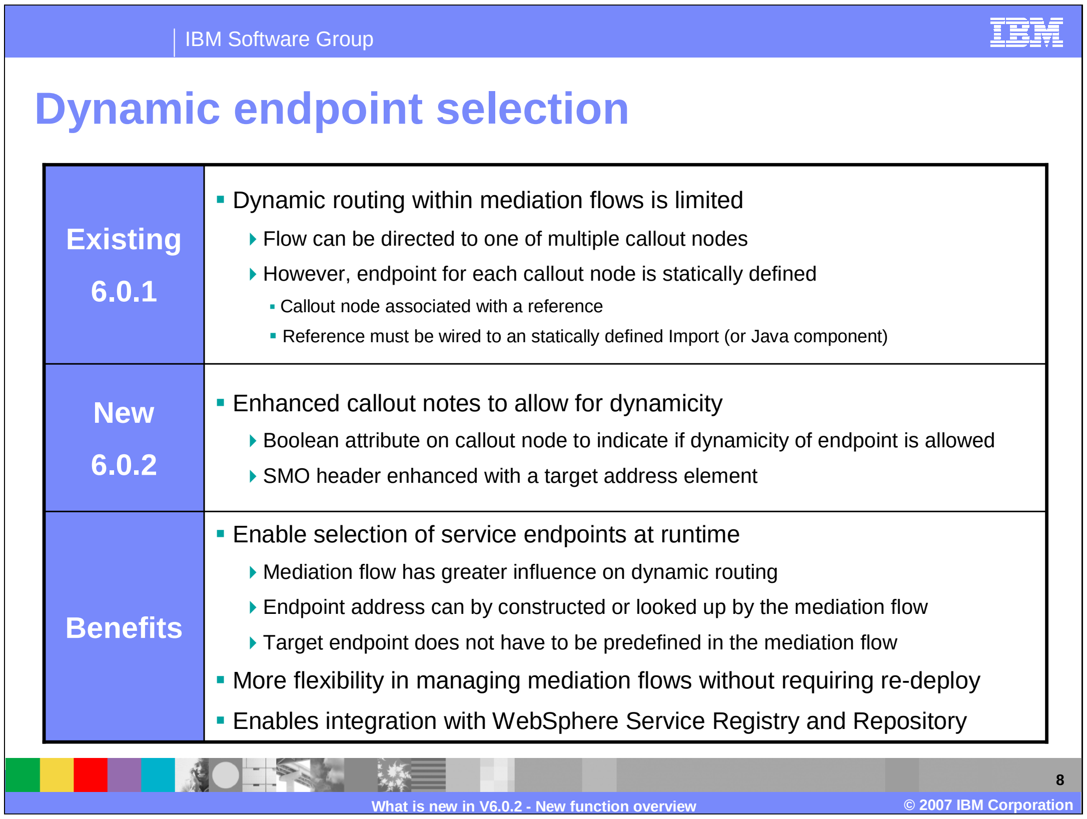
These enhancements include:

•Enabling of dynamic endpoint selection on the callout of a mediation flow

•The ability to handle unmodeled faults in the response flow

•The runtime administration of mediation properties

•Simplification in the configuration of custom mediation primitives


IBM Software Group

Dynamic endpoint selection

Existing 6.0.1	▪ Dynamic routing within mediation flows is limited ▸ Flow can be directed to one of multiple callout nodes ▸ However, endpoint for each callout node is statically defined ▪ Callout node associated with a reference ▪ Reference must be wired to an statically defined Import (or Java component)
New 6.0.2	▪ Enhanced callout notes to allow for dynamicity ▸ Boolean attribute on callout node to indicate if dynamicity of endpoint is allowed ▸ SMO header enhanced with a target address element
Benefits	▪ Enable selection of service endpoints at runtime ▸ Mediation flow has greater influence on dynamic routing ▸ Endpoint address can by constructed or looked up by the mediation flow ▸ Target endpoint does not have to be predefined in the mediation flow ▪ More flexibility in managing mediation flows without requiring re-deploy ▪ Enables integration with WebSphere Service Registry and Repository

8

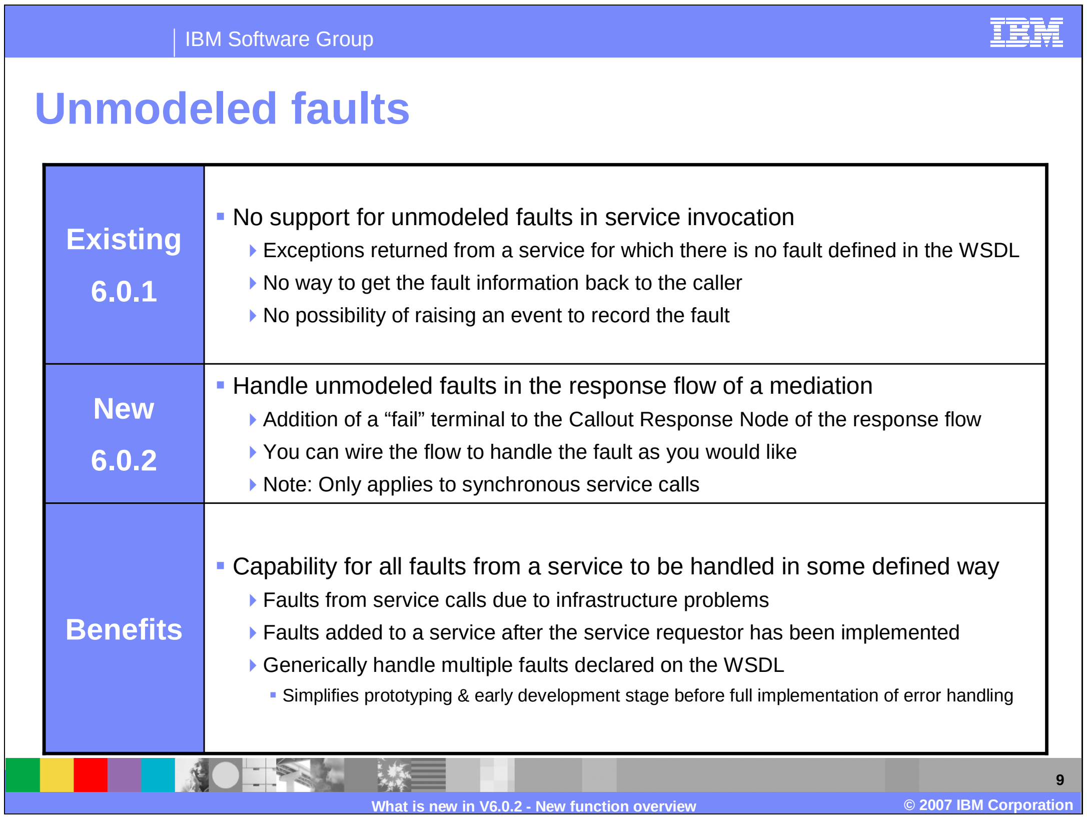
What is new in V6.0.2 - New function overview © 2007 IBM Corporation

One of the key characteristics of an enterprise service bus is the loose coupling of service requesters and service providers. The bus determines which provider to use to satisfy a request. This enhancement, dynamic endpoint selection, greatly increases the WebSphere Enterprise Service Bus capability and flexibility when making service provider routing decisions.

In version 6.0.1 it was possible to make service provider routing decisions within a mediation flow. The flow could be defined with multiple callout nodes. At runtime, decisions within the flow would determine which callout would be used for a particular request. However, each of the callout nodes contained within the flow had a static definition of the endpoint for the service provider. The callout node would be associated with a reference on the flow component, and the reference would be wired to either an SCA import or SCA Java component. In essence, the routing decision was made between different pre-configured endpoints.

In version 6.0.2 the callout node has been changed so that it can be configured to allow for the specification of a dynamic endpoint in the form of a URI identifying the target address of the service provider. The association of the callout node with a reference is then optional. The callout node obtains the URI from a field contained within the service message object header and calls that endpoint to satisfy the service request. The URI can identify the service as a Web Service with either SOAP over HTTP or SOAP over JMS. It can also identify the service as an export in another module which has either a Web Service binding or SCA binding. Finally, it can identify an import in the same module which can have any type of binding. It is up to the logic of the mediation flow to initialize the service message object header with a valid URI.

The benefit of this enhancement is that the endpoint of a service provider can be determined at runtime, giving the mediation flow a much greater influence over the routing to be done. The mediation flow can construct or look up a URI based on some logic, including the possibility of making a call to the WebSphere Service Registry and Repository. It is now also possible that when a service provider is moved and therefore has a new endpoint address, the change can be handled in the mediation flow without requiring the mediation module to be re-deployed.


IBM Software Group

Unmoded faults

Existing 6.0.1	▪ No support for unmodeled faults in service invocation ▸ Exceptions returned from a service for which there is no fault defined in the WSDL ▸ No way to get the fault information back to the caller ▸ No possibility of raising an event to record the fault
New 6.0.2	▪ Handle unmodeled faults in the response flow of a mediation ▸ Addition of a “fail” terminal to the Callout Response Node of the response flow ▸ You can wire the flow to handle the fault as you would like ▸ Note: Only applies to synchronous service calls
Benefits	▪ Capability for all faults from a service to be handled in some defined way ▸ Faults from service calls due to infrastructure problems ▸ Faults added to a service after the service requestor has been implemented ▸ Generically handle multiple faults declared on the WSDL ▪ Simplifies prototyping & early development stage before full implementation of error handling

9

What is new in V6.0.2 - New function overview

© 2007 IBM Corporation

The enhancement for unmodeled faults enables better error handling logic in mediation flows.

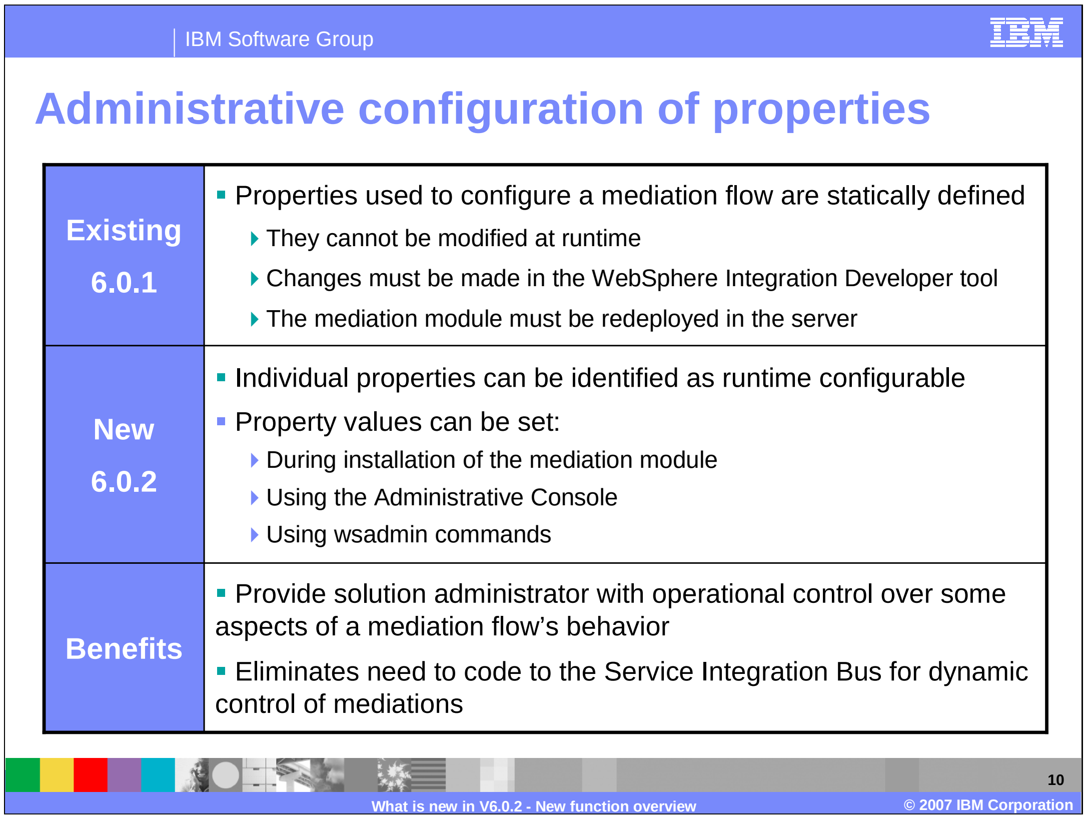
In version 6.0.1 the only way a mediation flow could handle an exception returned from a service provider was if it was a fault defined on the WSDL definition of the interface. Therefore, if an exception was returned for which there was no fault definition, the mediation flow would not be given control. There was no ability to perform additional processing such as logging the error, raising an event or returning the information to the service requestor.

In version 6.0.2 the ability to handle an unmodeled fault in the response flow has been added. This has been done by adding a fail terminal to the callout response node. When an unmodeled fault is returned, control passes through this fail terminal which can be wired to mediation primitives to perform appropriate processing logic for how you would like the fault to be handled.

Note that this capability only applies if the service request was synchronous. This is because the SCA exception handling architecture for asynchronous calls routes the exception to the failed event queue rather than returning it to the mediation.

The benefit of this enhancement is that unmodeled faults on synchronous calls can be handled in some defined way. It enables the mediation response flow to generate a log, raise an event or even to map the unmodeled fault to a defined fault so that it can be returned to the original requestor. Unmodeled faults might arise due to infrastructure problems while processing the call. They could also arise if new faults have been added to the interface of the service provider that the caller was not aware of.

It also enables the possibility of taking a generic approach to error handling. This can have benefits during prototyping and early stages of development before all the detailed error handling logic has been implemented.



Administrative configuration of properties

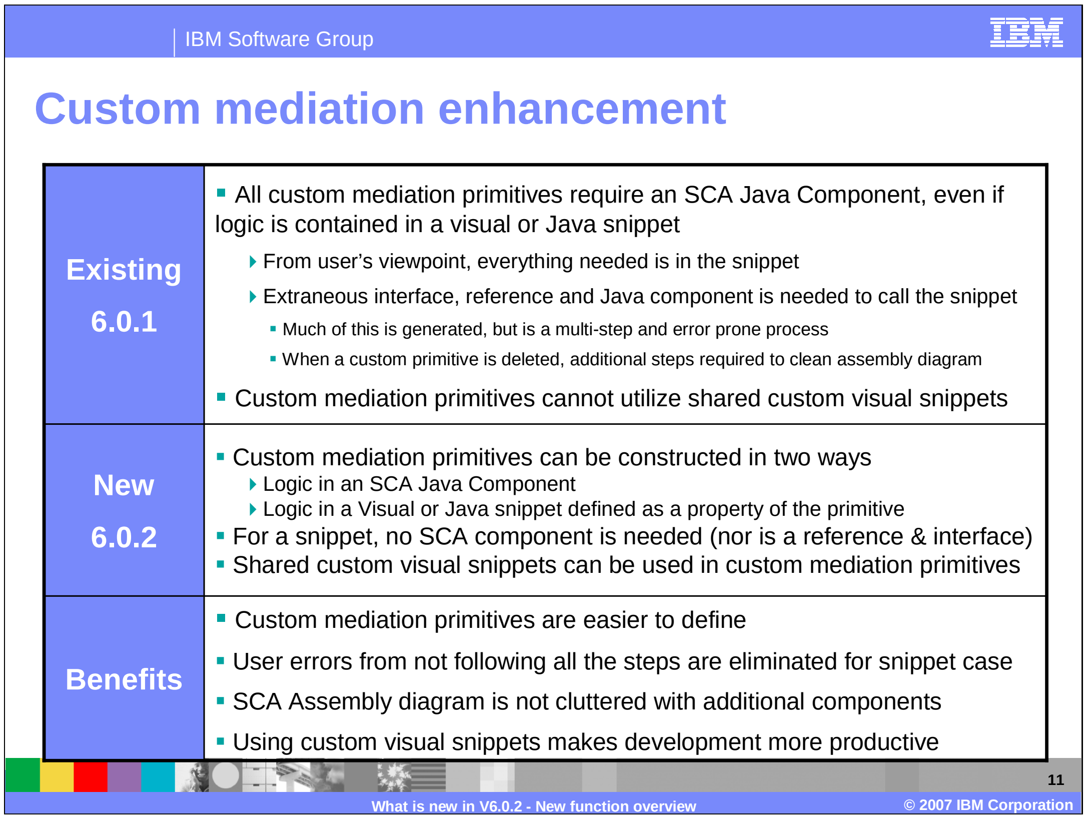
Existing 6.0.1	▪ Properties used to configure a mediation flow are statically defined ▸ They cannot be modified at runtime ▸ Changes must be made in the WebSphere Integration Developer tool ▸ The mediation module must be redeployed in the server
New 6.0.2	▪ Individual properties can be identified as runtime configurable ▪ Property values can be set: ▸ During installation of the mediation module ▸ Using the Administrative Console ▸ Using wsadmin commands
Benefits	▪ Provide solution administrator with operational control over some aspects of a mediation flow's behavior ▪ Eliminates need to code to the Service Integration Bus for dynamic control of mediations

This enhancement allows administrative control over selected properties of mediation primitives and mediation flows.

In version 6.0.1 the properties used to configure a mediation flow and the mediation primitives were all statically defined during development. In order to change any property value the change needed to be made in the WebSphere Integration Developer and the mediation module then needed to be redeployed to the server.

In version 6.0.2 the ability to administratively modify selected property values within the runtime environment has been added. The selection of which properties can be modified, referred to as being promoted, is based on two things. First, only selected properties are eligible to be promoted while other properties are restricted from being promoted. This is because some properties define values which could not be modified at runtime without requiring some other changes to the mediation. The remaining properties are not restricted and these properties are referred to as being promotable. It is the mediation flow designer who then determines for a specific mediation flow which of the promotable properties will actually be promoted to allow administrative control. Those properties that are promoted contain a value specified in the mediation flow itself, and the value can be modified during installation of the module into the server or by using the Administrative Console or wsadmin commands.

The benefits of this enhancement is that the administrator now can have some control over selected aspects of the mediation flow. For example, a flow could be defined where the administrator had control over whether or not log messages would be written, allowing him to turn on logging if trying to debug a problem. It also eliminates the need to code to the service integration bus when dynamic control of mediations is a requirement.



Custom mediation enhancement

Existing 6.0.1	▪ All custom mediation primitives require an SCA Java Component, even if logic is contained in a visual or Java snippet ▸ From user's viewpoint, everything needed is in the snippet ▸ Extraneous interface, reference and Java component is needed to call the snippet ▪ Much of this is generated, but is a multi-step and error prone process ▪ When a custom primitive is deleted, additional steps required to clean assembly diagram ▪ Custom mediation primitives cannot utilize shared custom visual snippets
New 6.0.2	▪ Custom mediation primitives can be constructed in two ways ▸ Logic in an SCA Java Component ▸ Logic in a Visual or Java snippet defined as a property of the primitive ▪ For a snippet, no SCA component is needed (nor is a reference & interface) ▪ Shared custom visual snippets can be used in custom mediation primitives
Benefits	▪ Custom mediation primitives are easier to define ▪ User errors from not following all the steps are eliminated for snippet case ▪ SCA Assembly diagram is not cluttered with additional components ▪ Using custom visual snippets makes development more productive

This enhancement makes it easier for you to use custom mediation primitives with a Java or visual snippet.

In version 6.0.1 all custom mediation primitives defined in the mediation flow required an SCA Java component to be wired in the assembly diagram. In some cases the logic would be encoded in the Java component itself. However, in the case where the custom mediation primitive was a visual or Java snippet, the Java component only contained generated code that called the snippet. So from your point of view, all the logic needed was in the snippet but it carried along with it an extra interface, reference and Java component as part of the infrastructure. This resulted in extra steps when creating a custom mediation primitive and additional cleanup that was needed when a custom mediation primitive was deleted from a flow. The need for these extra steps made the process error prone.

Additionally, in version 6.0.1, a custom mediation primitive visual snippet could not utilize any shared custom visual snippets that were defined for the project.

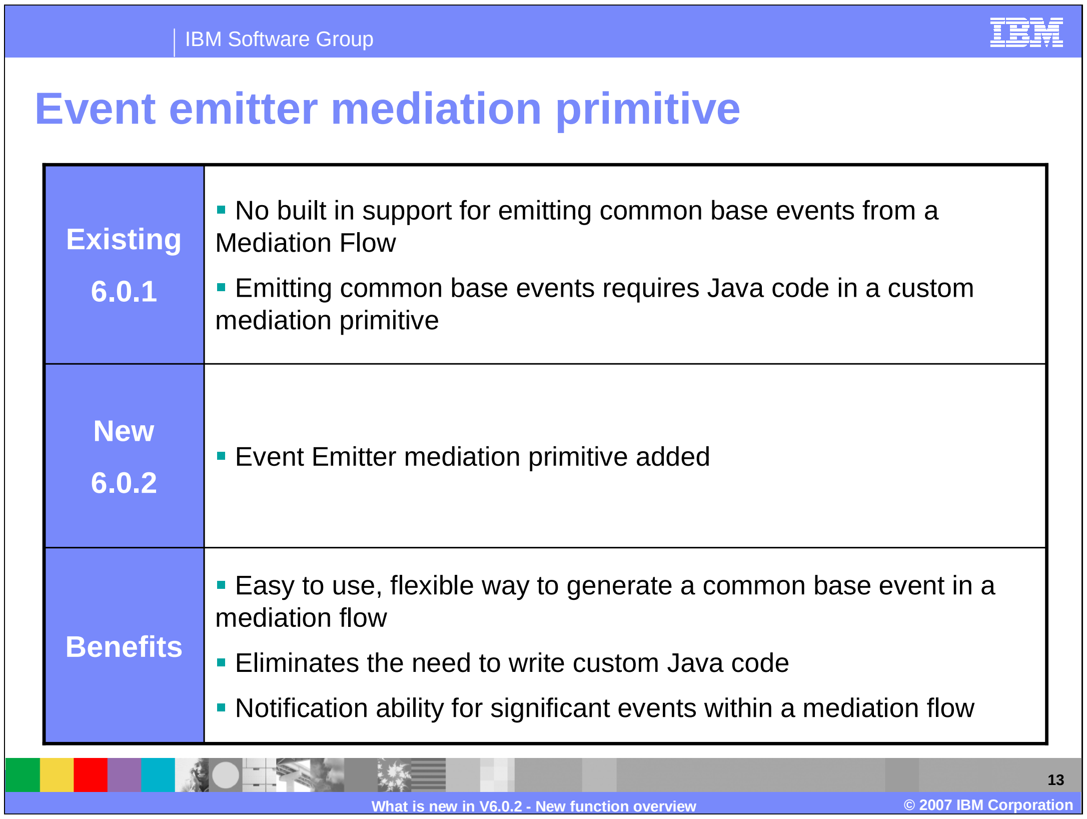
In version 6.0.2, the handling of custom mediation primitives was greatly simplified. When creating the custom mediation primitive you specify if the logic will be a visual snippet, a Java snippet or an invoke of an SCA Java component. If either of the snippets is selected, then the additional interface, reference and Java component are not needed and the snippet logic is coded as a property of the primitive. Additionally, if it is a visual snippet, any custom visual snippets available in your project can be used in building the logic.

The benefit of this enhancement is that custom mediation primitives using a visual or Java snippet are easier to define, modify and maintain and the process of creating them is less error prone. The SCA assembly diagram is no longer cluttered with references wired to Java components which contained no real logic other than to call a snippet. Finally, the ability to use custom visual snippets make the overall development process easier and more productive.

Section

New mediation primitives

You will be introduced to three new mediation primitives in this section. They are the Event Emitter mediation primitive, the Message Element Setter mediation primitive and the Endpoint Lookup mediation primitive.


IBM Software Group

Event emitter mediation primitive

Existing 6.0.1	▪ No built in support for emitting common base events from a Mediation Flow ▪ Emitting common base events requires Java code in a custom mediation primitive
New 6.0.2	▪ Event Emitter mediation primitive added
Benefits	▪ Easy to use, flexible way to generate a common base event in a mediation flow ▪ Eliminates the need to write custom Java code ▪ Notification ability for significant events within a mediation flow

13

What is new in V6.0.2 - New function overview

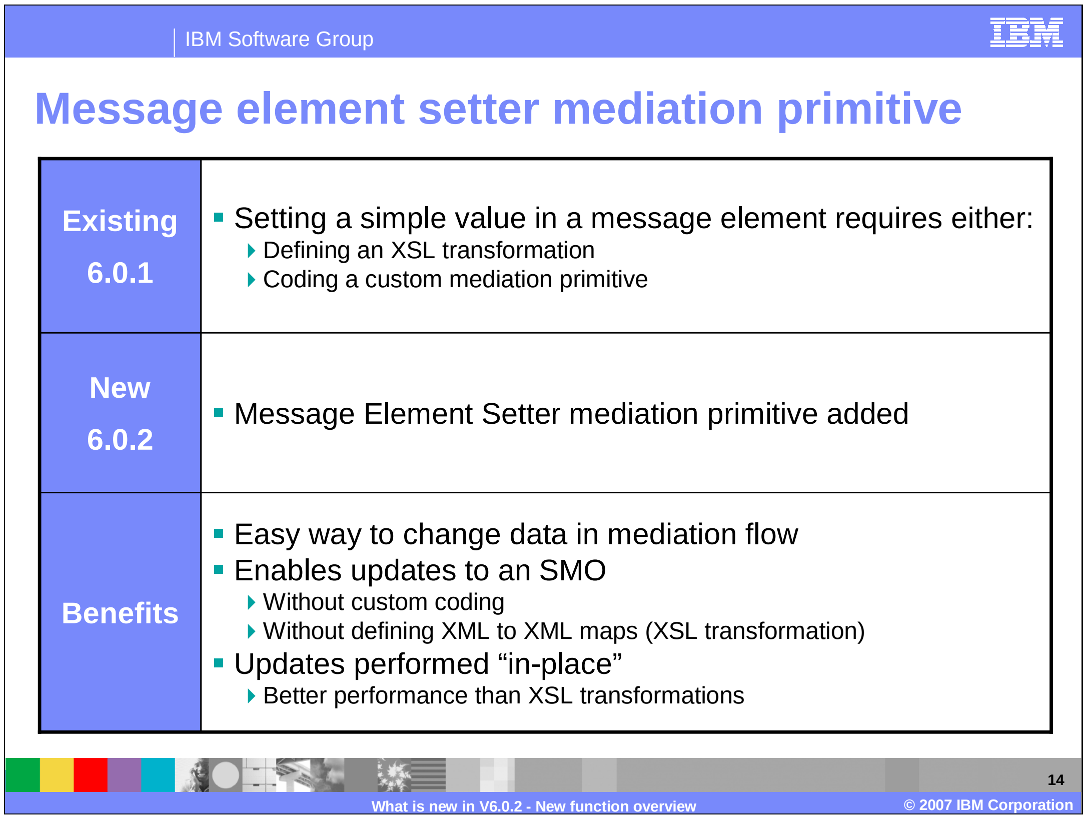
© 2007 IBM Corporation

This enhancement provides a mechanism to raise an event within a mediation flow without having to write your own Java code.

In version 6.0.1 there was no built in mechanism within a mediation flow which enabled you to emit a Common Base Event, commonly called a CBE. In order to raise a common base event you needed to use a custom mediation primitive containing Java code which used the event emitter APIs to raise the event.

In version 6.0.2 a new Event Emitter mediation primitive has been added. You configure the Event Emitter primitive to emit a common base event that will be handled by the common event infrastructure. The event can be configured to contain any portion of the service message object that is relevant to the event being raised. The event participates in all aspects of the common event infrastructure and is therefore viewable using the Common Base Event Browser and can also be consumed by the WebSphere Business Monitor or your own custom event processing application.

The benefit provided is the ease in which events can now be emitted from within a mediation flow. This enables easily providing notification of significant events that occur within the flow.



Message element setter mediation primitive

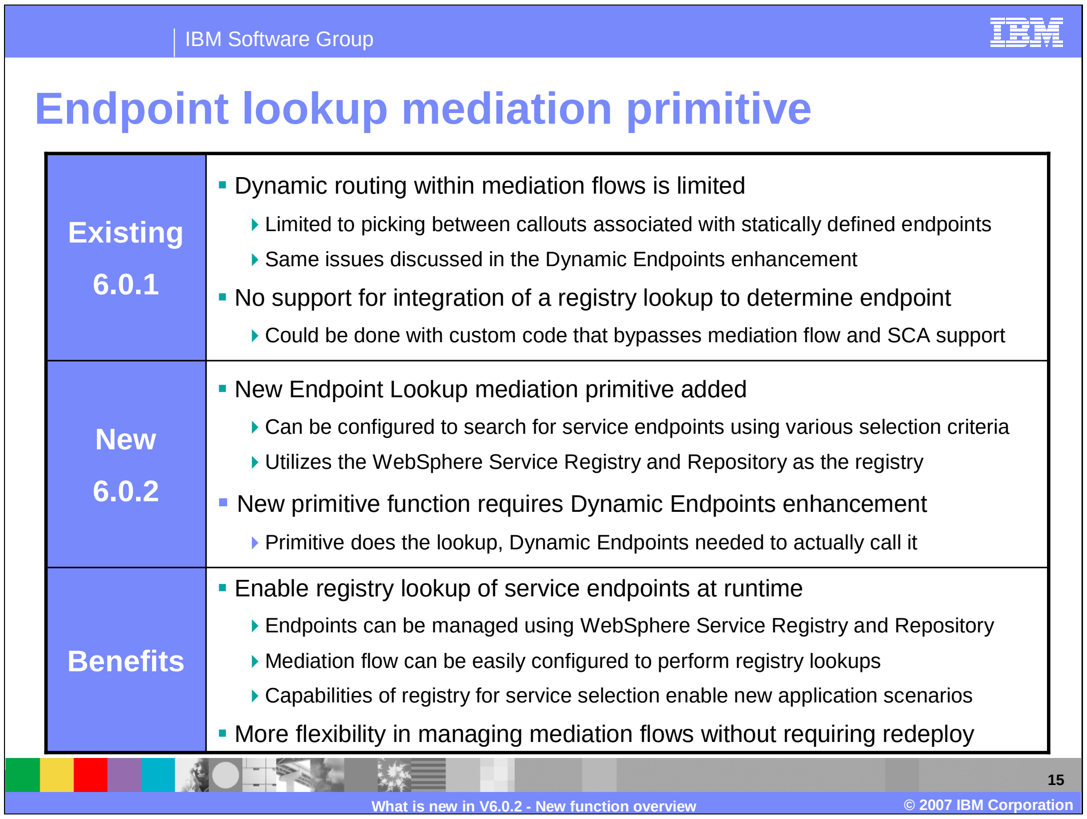
Existing 6.0.1	▪ Setting a simple value in a message element requires either: ▸ Defining an XSL transformation ▸ Coding a custom mediation primitive
New 6.0.2	▪ Message Element Setter mediation primitive added
Benefits	▪ Easy way to change data in mediation flow ▪ Enables updates to an SMO ▸ Without custom coding ▸ Without defining XML to XML maps (XSL transformation) ▪ Updates performed “in-place” ▸ Better performance than XSL transformations

This enhancement provides a new and simpler mechanism for setting a value into the service message object.

In version 6.0.1 there were two mechanisms by which you could set a value into an element of the service message object. One approach was using an XSL Transformation primitive and the other was by coding a custom mediation primitive. When just needing to set values for individual elements of a service message object these two more powerful approaches are more difficult to implement than what is really required for the simple setting of an element.

In version 6.0.2 the Message Element Setter primitive has been added. This provides a much more straight forward way to set individual elements of the service message object. An element can be set to a constant value or a value copied from another element of the service message object. Copy gives the capability to copy leaf node elements and also sub-trees provided the source and target sub-trees match.

The benefits of this enhancement include ease of development and also better performance. Ease of development comes when the Message Element Setter can be used rather than having to write custom Java code. It is also easier to configure the Message Element Setter primitive than it is to configure an XSL Transformation primitive. Additionally, the XSL Transformation primitive makes a new copy of the service message object whereas the Message Element Setter primitive updates the service message object in place and therefore provides better performance.


IBM Software Group

Endpoint lookup mediation primitive

Existing 6.0.1	▪ Dynamic routing within mediation flows is limited ▸ Limited to picking between callouts associated with statically defined endpoints ▸ Same issues discussed in the Dynamic Endpoints enhancement ▪ No support for integration of a registry lookup to determine endpoint ▸ Could be done with custom code that bypasses mediation flow and SCA support
New 6.0.2	▪ New Endpoint Lookup mediation primitive added ▸ Can be configured to search for service endpoints using various selection criteria ▸ Utilizes the WebSphere Service Registry and Repository as the registry ▪ New primitive function requires Dynamic Endpoints enhancement ▸ Primitive does the lookup, Dynamic Endpoints needed to actually call it
Benefits	▪ Enable registry lookup of service endpoints at runtime ▸ Endpoints can be managed using WebSphere Service Registry and Repository ▸ Mediation flow can be easily configured to perform registry lookups ▸ Capabilities of registry for service selection enable new application scenarios ▪ More flexibility in managing mediation flows without requiring redeploy

15

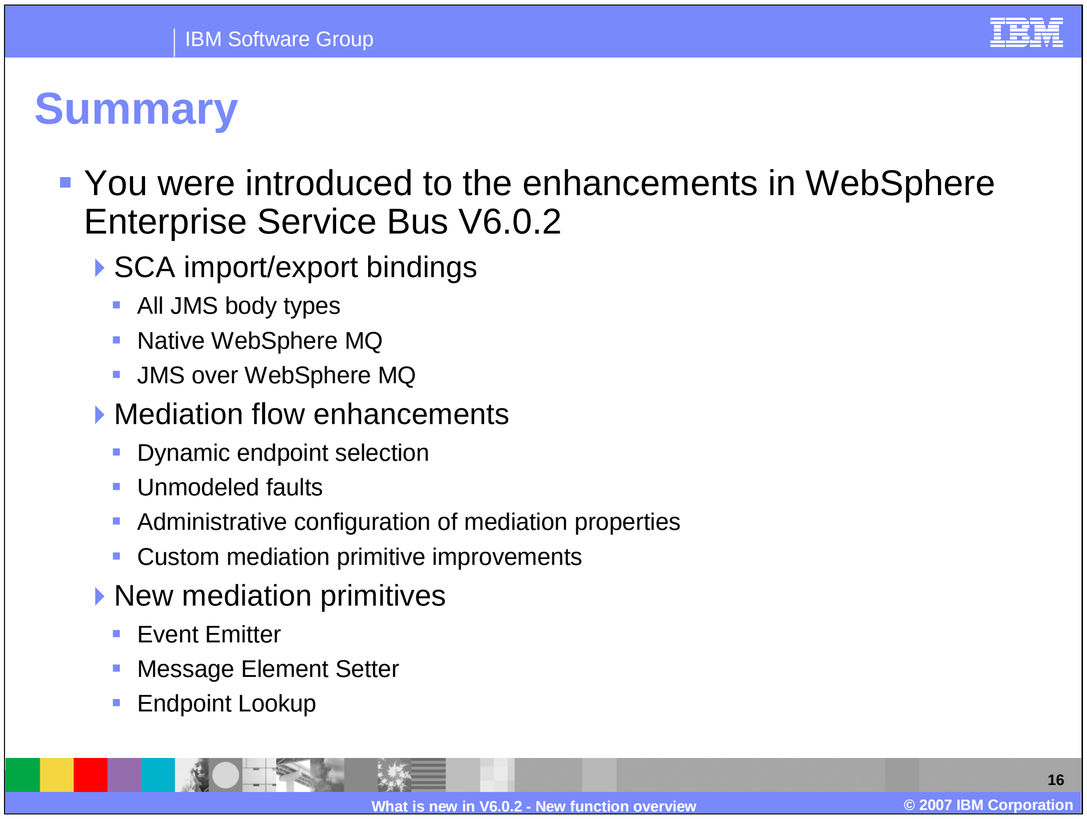
What is new in V6.0.2 - New function overview © 2007 IBM Corporation

This enhancement provides a mechanism to integrate the use of WebSphere Service Registry and Repository within a mediation flow.

In version 6.0.1 a mediation flow was limited in its dynamic routing capabilities. It could only select between callouts which had statically defined endpoints associated with them. In order to make use of a registry for the lookup of service endpoints in 6.0.1 you would have to write custom code. This code would need to make calls to the service providers directly, bypassing the mediation flow and SCA support all together.

In version 6.0.2 the Endpoint Lookup mediation primitive has been added. The new primitive can be configured to make calls to a WebSphere Service Registry and Repository. You configure various selection criteria used by the registry to lookup possible service endpoints. This new primitive works in conjunction with the dynamic endpoint selection enhancements, setting the URI of the selected endpoint in the service message object which is then used by the dynamic callout.

The benefits of this enhancement include the ability to have the WebSphere Service Registry and Repository manage the endpoints for services in your environment. The primitive can be easily configured to perform the lookups. This leads to an environment that can be more easily managed and enables scenarios where service endpoints can be moved within the environment without having to re-deploy the applications and mediations which use those services.

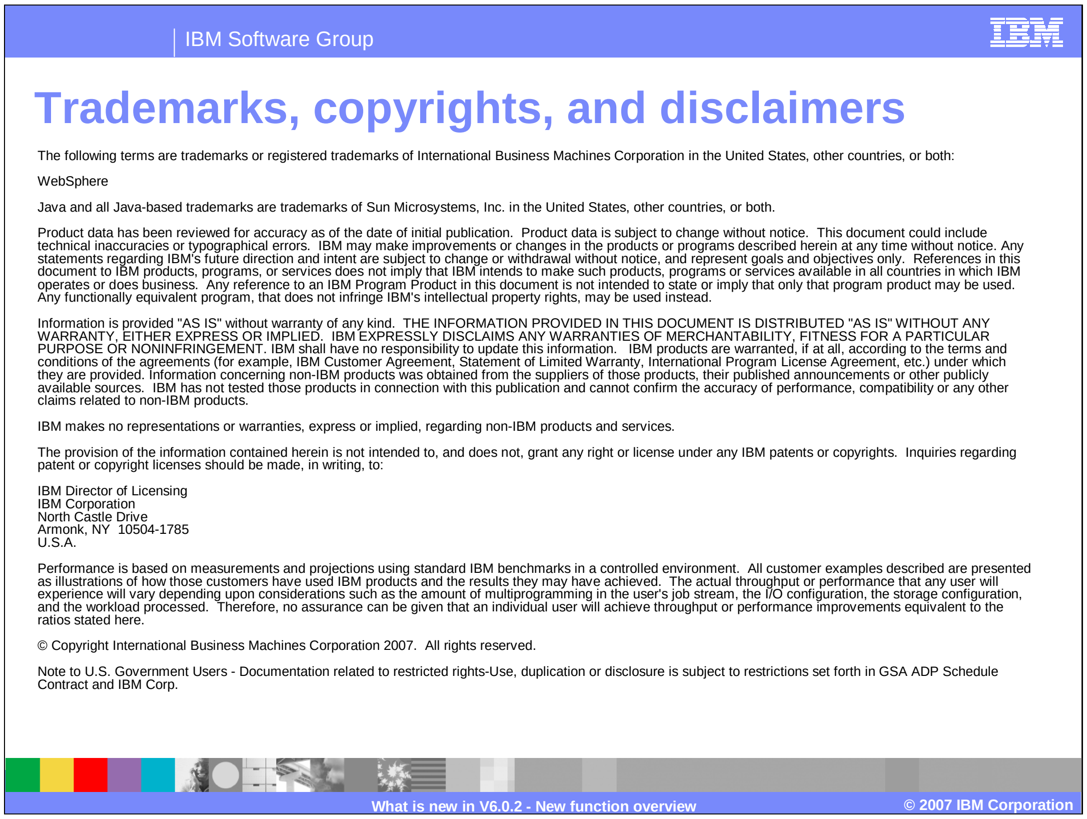


Summary

- You were introduced to the enhancements in WebSphere Enterprise Service Bus V6.0.2
  - SCA import/export bindings
    - All JMS body types
    - Native WebSphere MQ
    - JMS over WebSphere MQ
  - Mediation flow enhancements
    - Dynamic endpoint selection
    - Unmodeled faults
    - Administrative configuration of mediation properties
    - Custom mediation primitive improvements
  - New mediation primitives
    - Event Emitter
    - Message Element Setter
    - Endpoint Lookup

In this presentation you were introduced to the new features in WebSphere Enterprise Service Bus version 6.0.2. These enhancements fall into three categories. The SCA import and export binding enhancements include the support for all JMS body types, bindings for native WebSphere MQ and also bindings for JMS over WebSphere MQ. The Mediation flow enhancements include dynamic endpoint selection, handling of unmodeled faults, runtime administration of mediation properties and improved configuration of custom mediation primitives. The new mediation primitives are the event emitter primitive, the message element setter primitive and the endpoint lookup primitive.

See other presentations for more detailed information on these enhancements.



Trademarks, copyrights, and disclaimers