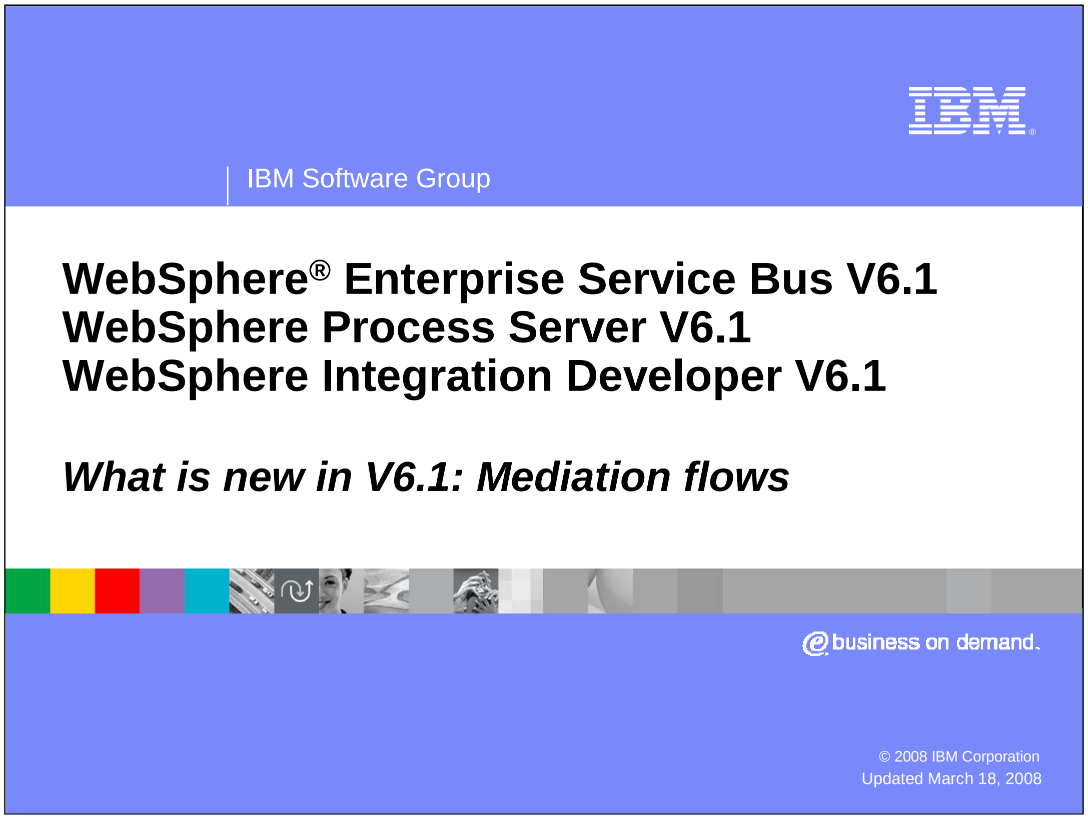


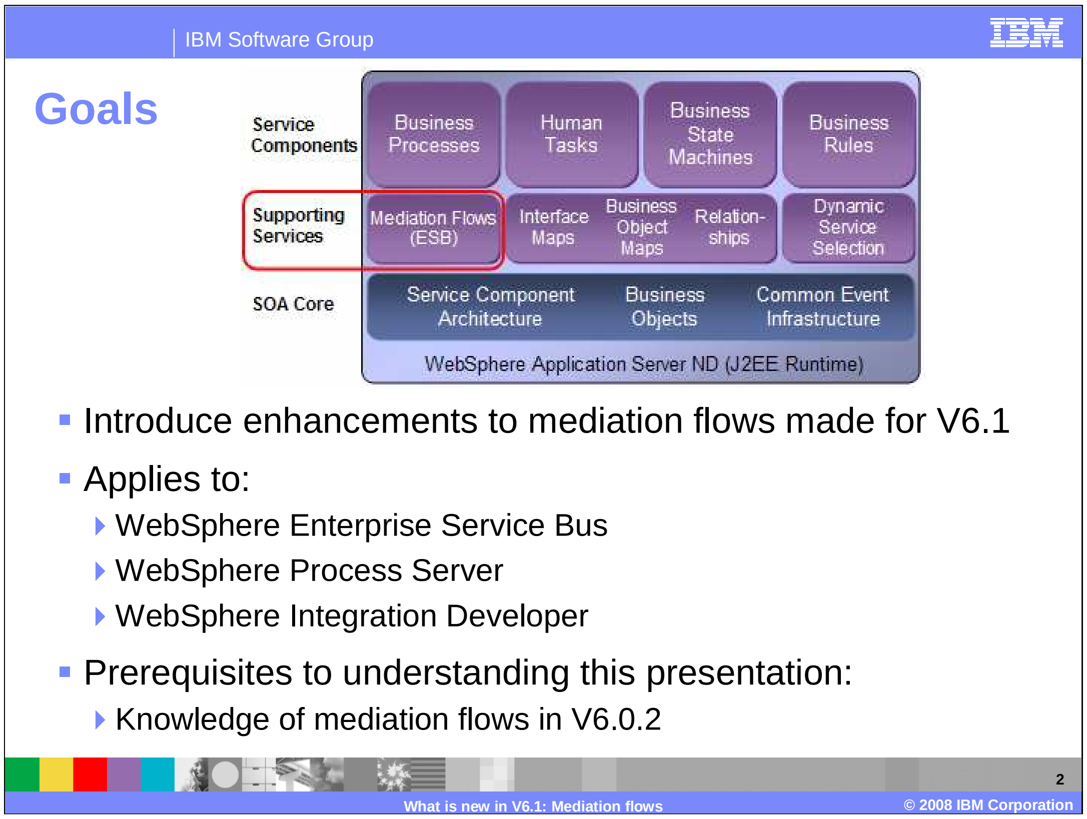
IBM Software Group

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

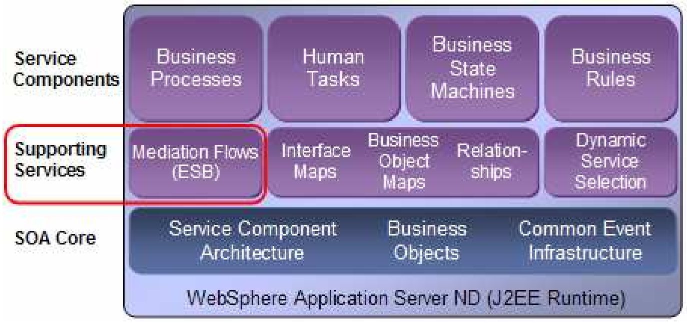
What is new in V6.1: Mediation flows

business on demand.

This presentation provides an introduction to the new function delivered in version 6.1 for mediation flows in WebSphere Enterprise Service Bus, WebSphere Process Server and WebSphere Integration Developer.



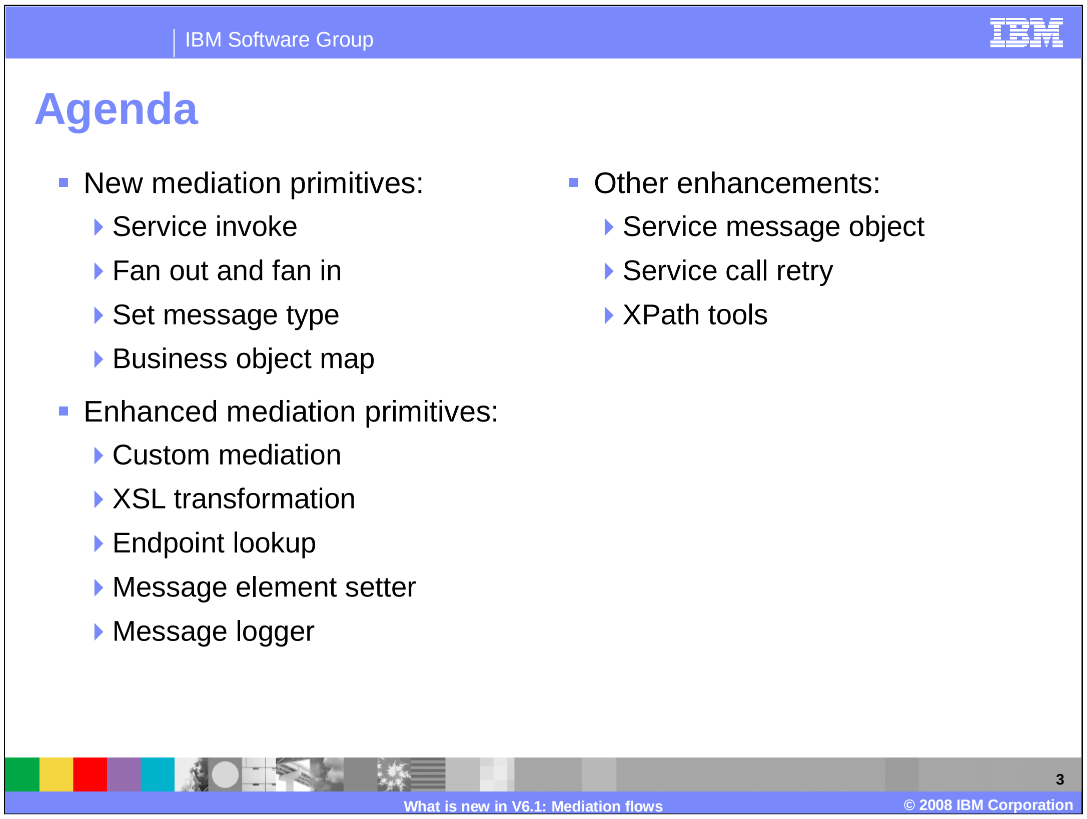
Goals



- Introduce enhancements to mediation flows made for V6.1
- Applies to:
  - WebSphere Enterprise Service Bus
  - WebSphere Process Server
  - WebSphere Integration Developer
- Prerequisites to understanding this presentation:
  - Knowledge of mediation flows in V6.0.2

The goal of this presentation is to introduce you to the enhancements that have been made to mediation flows for version 6.1. The architecture chart shows the place of mediation flows in the overall architecture for WebSphere Process Server. These enhancements apply equally to WebSphere Process Server and WebSphere Enterprise Service Bus, including new and changed capabilities in WebSphere Integration Developer which support these enhancements from a development tool perspective.

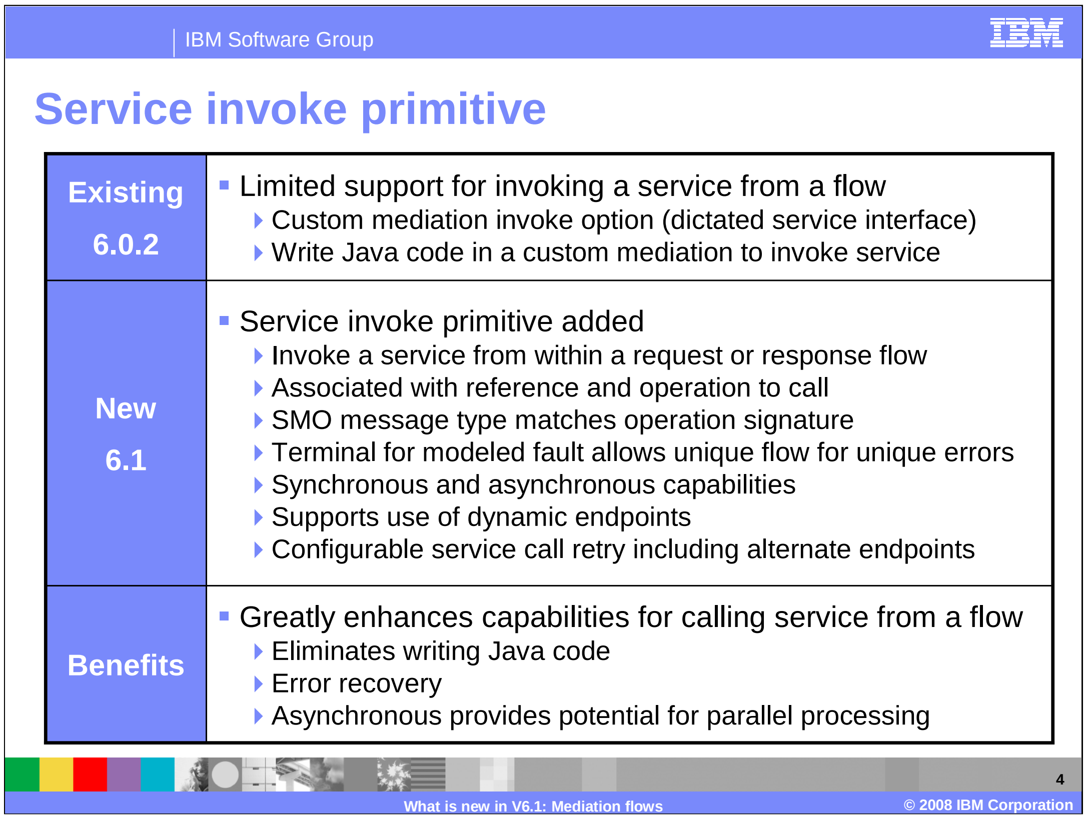
In order to understand the material in this presentation you should already have a knowledge of the capabilities of mediation flows in version 6.0.2 of the products.



Agenda

- New mediation primitives:
  - Service invoke
  - Fan out and fan in
  - Set message type
  - Business object map
- Enhanced mediation primitives:
  - Custom mediation
  - XSL transformation
  - Endpoint lookup
  - Message element setter
  - Message logger
- Other enhancements:
  - Service message object
  - Service call retry
  - XPath tools

The first new features for version 6.1 to be described are the new mediation primitives. These include the service invoke primitive, the fan out and fan in primitives, the set message type primitive and the business object map primitive. In addition to these, there are several of the existing version 6.0.2 primitives that have been enhanced. These include the custom mediation primitive, the XSL transformation primitive, the endpoint lookup primitive, the message element setter primitive and the message logger primitive. Other enhancements to be described are the modifications to the service message object, the ability to perform retry for failing service calls and enhanced tools for specifying XPath expressions.


IBM Software Group

Service invoke primitive

Existing 6.0.2	▪ Limited support for invoking a service from a flow ▸ Custom mediation invoke option (dictated service interface) ▸ Write Java code in a custom mediation to invoke service
New 6.1	▪ Service invoke primitive added ▸ Invoke a service from within a request or response flow ▸ Associated with reference and operation to call ▸ SMO message type matches operation signature ▸ Terminal for modeled fault allows unique flow for unique errors ▸ Synchronous and asynchronous capabilities ▸ Supports use of dynamic endpoints ▸ Configurable service call retry including alternate endpoints
Benefits	▪ Greatly enhances capabilities for calling service from a flow ▸ Eliminates writing Java code ▸ Error recovery ▸ Asynchronous provides potential for parallel processing

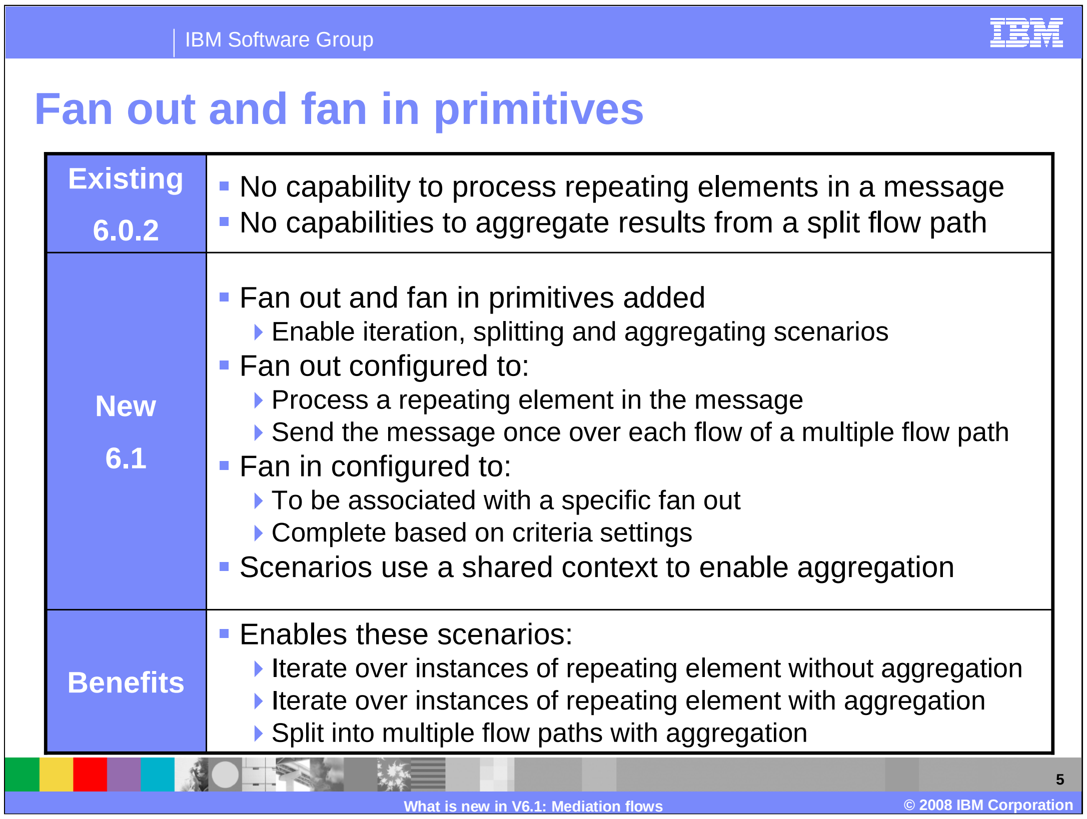
4

What is new in V6.1: Mediation flows © 2008 IBM Corporation

In version 6.0.2 there are two ways you can call a service from within a mediation flow. The first of these is the custom mediation with the invoke option. When this approach is used, the interface of the service is dictated by the custom mediation and therefore this is only practical when the service has been build expressly for this purpose. The other way to invoke a service requires you to write the Java code to do it within a custom mediation.

In version 6.1 the service invoke primitive has been added to provide a practical mechanism for calling a service from within a request or response mediation flow. The primitive is configured for a specific reference defined on the mediation flow component and for an operation defined on the interface the reference supports. The input terminal to the primitive has a message type corresponding to the operation's input definition and the output terminal has a message type corresponding to the operation's output definition. In addition, any modeled faults defined for the operation result in an additional output terminal with a message type matching the fault definition. The call to the service can be synchronous or asynchronous. Which it is depends upon a variety of contributing factors that are not discussed here. The use of dynamic endpoints is supported similar to how they work for callout nodes in version 6.0.2. Finally, whether to attempt to retry failing calls to the service can be configured. When the call makes use of dynamic endpoints the retry can make use of alternate endpoint addresses allowing the retry to be attempted to a different provider of the same service.

Clearly, the ability to make service calls from within a mediation flow has been greatly enhanced through the addition of the service invoke primitive. The need to write your own Java code has been eliminated, there is excellent support for error recovery and the use of asynchronous calls provides the potential for parallel processing.


IBM Software Group

Fan out and fan in primitives

Existing 6.0.2	▪ No capability to process repeating elements in a message ▪ No capabilities to aggregate results from a split flow path
New 6.1	▪ Fan out and fan in primitives added ▸ Enable iteration, splitting and aggregating scenarios ▪ Fan out configured to: ▸ Process a repeating element in the message ▸ Send the message once over each flow of a multiple flow path ▪ Fan in configured to: ▸ To be associated with a specific fan out ▸ Complete based on criteria settings ▪ Scenarios use a shared context to enable aggregation
Benefits	▪ Enables these scenarios: ▸ Iterate over instances of repeating element without aggregation ▸ Iterate over instances of repeating element with aggregation ▸ Split into multiple flow paths with aggregation

5

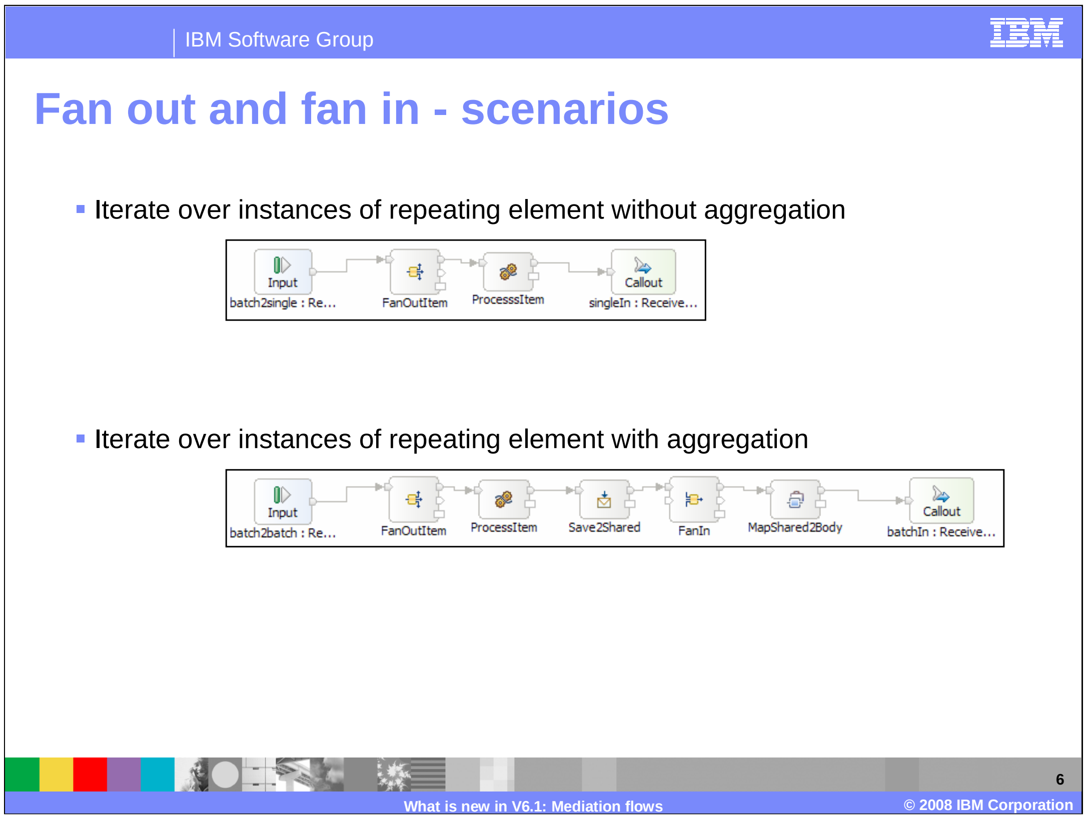
What is new in V6.1: Mediation flows

© 2008 IBM Corporation

Using the version 6.0.2 mediation flow capabilities it is not possible to process repeating elements within a message without writing custom code to do it. Also, when a flow has a split path there is no mechanism to combine the results of the processing done on each of the paths.

In version 6.1 two new primitives, fan out and fan in, are added to enable these scenarios that require iteration, splitting and aggregating of messages. The fan out can be configured in two different ways. One configuration option identifies a repeating element in the message that is iterated over by sending the message over the subsequent flow once for each element in the array. The other configuration option is used when there are multiple flows wired from the fan out and the message is sent once over each of the wired flows. A fan in is then configured such that it is associated with a specific fan out and has some completion criteria. When the flow reaches the fan in, if the completion criteria has not been met it returns to the fan out. If the completion criteria has been met the flow continues following the fan in. While the processing has been taking place between the fan out and fan in, the shared context, a shared memory area, is used to accumulate the results. When the fan in completes, the flow can take the aggregated results out of the shared context and use them to set up the message body.

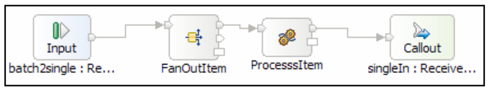
The benefit of these primitives is the processing scenarios they enable, of which there are basically three. There is iteration over repeating elements which can be done either with or without aggregation of results. The other scenario is multiple flow paths which is useful only when used with aggregation of results. Examples of these are shown on the next two slides.



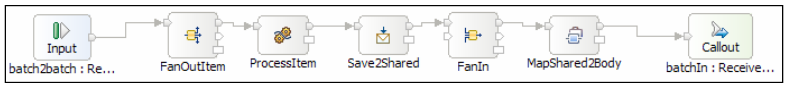
IBM Software Group

Fan out and fan in - scenarios

- Iterate over instances of repeating element without aggregation



- Iterate over instances of repeating element with aggregation



6

What is new in V6.1: Mediation flows

© 2008 IBM Corporation

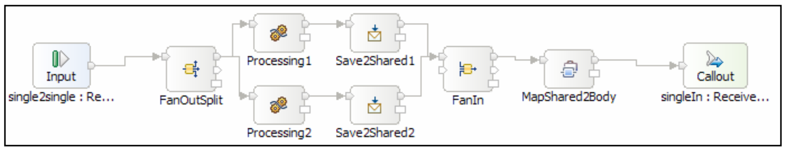
These flows illustrate the basic scenarios that will typically be used with the fan out and fan in primitives. For simplicity of explanation, these scenarios are all using one way operations on input and output, but that does not imply these patterns can not be used with request response operations.

The first of these scenarios is using fan out to iterate over a repeating element in the message but without aggregating the results. Because there is no aggregation of results, there is no fan in primitive in the flow. In this scenario the incoming one way message has a list of item IDs sent as a batch. The output is a series of messages, one for each item, that contains the item ID and additional information about that item. Looking at the flow, you will see the input node is wired to a fan out primitive, FanOutItem, which is configured to iterate over the item IDs in the message. The ProcessItem primitive adds the additional item information for a single item and passes it to the callout node, which forwards the message to the service. After the callout node sends the message, the flow returns to the FanOutItem which propagates the message again to ProcessItem for the next item in the batch. The flow completes when there are no more items in the batch.

The second scenario also iterates over a repeating element in the message. The input is a batch of item IDs and the output is a single message with a batch of item IDs plus additional information for each item. In this case, the FanOutItem primitive is configured exactly the same as in the previous example. It propagates the message to ProcessItem which adds additional information for an individual item. The message is then propagated to the Save2Share primitive which saves the item ID and information in the shared context. The flow then proceeds to the FanIn primitive which is configured to complete when all items have been processed. If all items have not been processed, the flow returns to the FanOutItem, with this loop between FanOutItem and FanIn continuing until all items are processed. The flow then proceeds to the MapShared2Body primitive which takes the aggregated item ID and information for all items and maps them from the shared context to the message body for the output message. This is then passed to the callout node which forwards the message to the service.

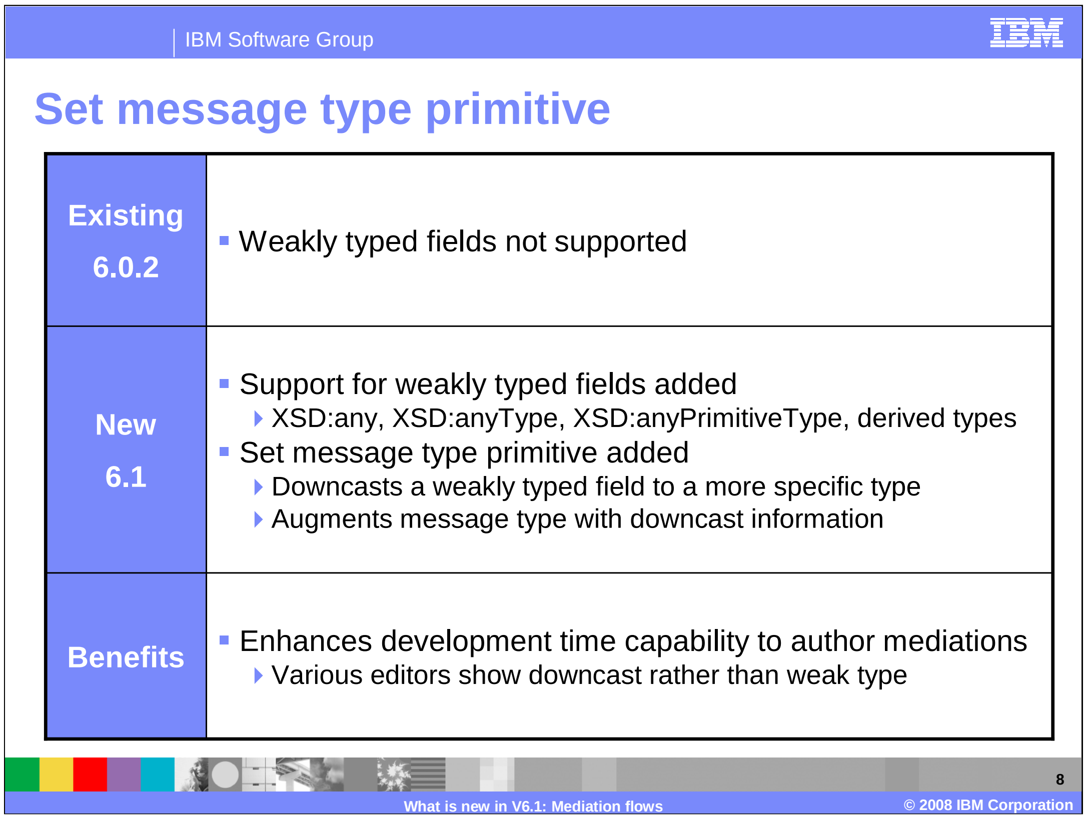
Fan out and fan in – scenarios (continued)

- Split into multiple flow paths with aggregation



The third scenario receives an incoming message with a single item ID which requires multiple processing paths to complete the information needed for the outbound message. In this case, the fan out primitive, FanOutSplit, is configured to send the message once and the fan in primitive, FanIn, is configured to complete after receiving two messages. The message is propagated from the input node to the FanOutSplit which propagates it on one of the wires to Processing1 which provides the additional information it adds. The message then goes to Save2Shared1 which places the information into the shared context and the flow proceeds to the FanIn. Since this is the first message received at the FanIn, the flow returns to the FanOutSplit which propagates the message to Processing2 which adds the additional information it provides. The message then flows to Save2Shared2 which places the information into the shared context and the flow proceeds to the FanIn. Since this is the second message received at FanIn, the flow proceeds to MapShared2Body which takes the information from the shared context and creates the body of the output message which it then propagated to the output node.

In all three of these cases, the processing for the item is shown as a single primitive ProcessItem, Processing1 or Processing2, which was done for simplicity's sake. The reality is that, depending upon the application requirements, there can be several primitives involved in the flow to accomplish what needs to be done.


IBM Software Group

Set message type primitive

Existing 6.0.2	▪ Weakly typed fields not supported
New 6.1	▪ Support for weakly typed fields added ▸ XSD:any, XSD:anyType, XSD:anyPrimitiveType, derived types ▪ Set message type primitive added ▸ Downcasts a weakly typed field to a more specific type ▸ Augments message type with downcast information
Benefits	▪ Enhances development time capability to author mediations ▸ Various editors show downcast rather than weak type

8

What is new in V6.1: Mediation flows

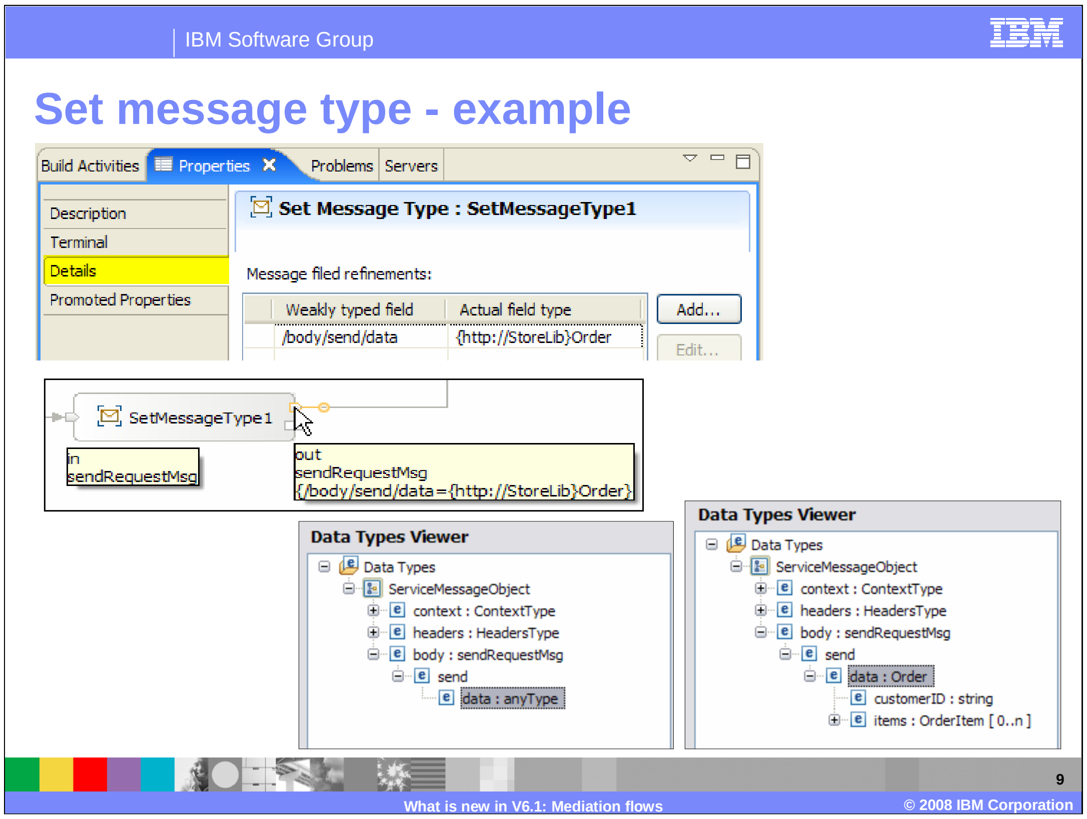
© 2008 IBM Corporation

In version 6.0.2 weakly typed fields were not supported well in some areas and not supported at all in other areas.

In version 6.1 there were many improvements made throughout the products to support weakly typed fields. Weakly typed fields include XSD:any, XSD:anyType, XSD:anyPrimitiveType and derived types.

When you are developing a mediation flow that has a weakly typed field in the SMO, it is possible that you know what type the field will actually contain at that point in the flow. When that is the case, you can use the new set message type primitive to declare what the type is. From that point on in the flow, the additional type information provided by this primitive is used to further define the message type. Conceptually, this is providing a function very similar to a cast operation in a programming language.

The benefit of this is the ability for the tools to provide you with a view of the service message object that understands the downcast type. For example, the XPath expression builder dialog is able to show you the downcast type when building an XPath expression rather than only showing the weaker type.


IBM Software Group

Set message type - example

Set Message Type : SetMessageType1

Message filed refinements:

Weakly typed field	Actual field type
/body/send/data	{http://StoreLib}Order

in
sendRequestMsg

out
sendRequestMsg
{/body/send/data={http://StoreLib}Order}

Data Types Viewer

```
Data Types
  ServiceMessageObject
    context : ContextType
    headers : HeadersType
    body : sendRequestMsg
      send
        data : anyType
```

Data Types Viewer

```
Data Types
  ServiceMessageObject
    context : ContextType
    headers : HeadersType
    body : sendRequestMsg
      send
        data : Order
          customerID : string
          items : OrderItem [ 0..n ]
```

9

What is new in V6.1: Mediation flows

© 2008 IBM Corporation

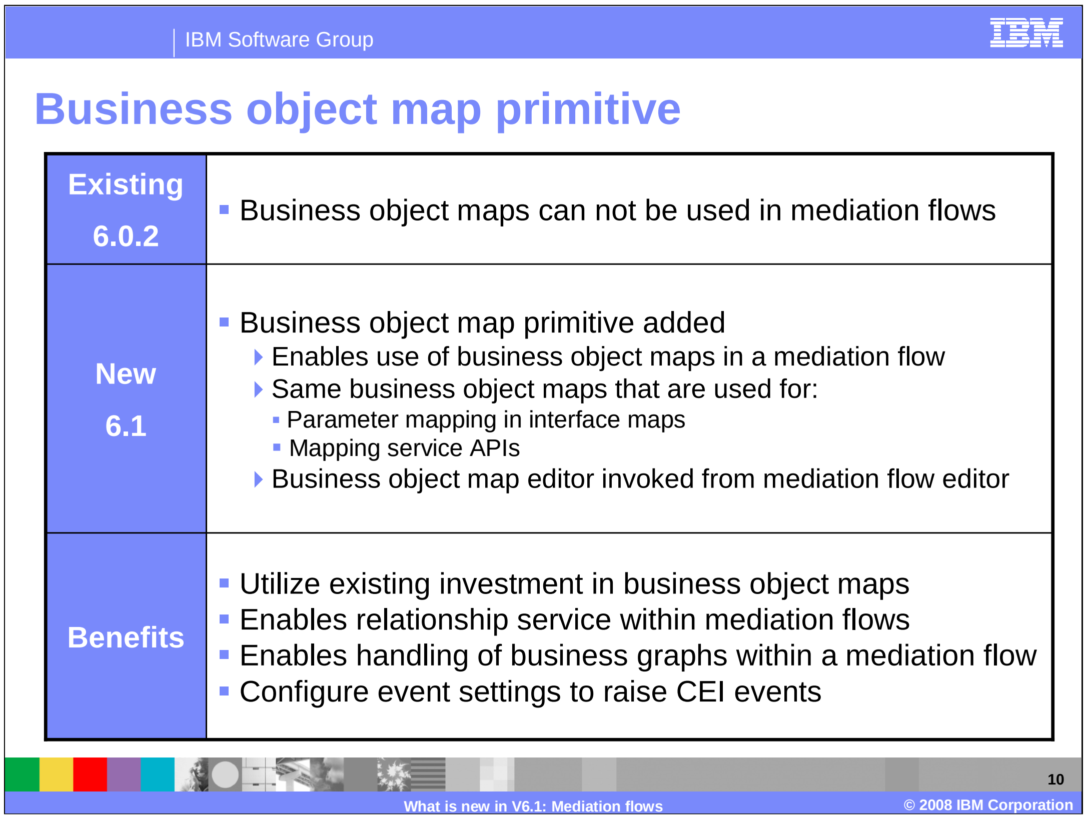
In this slide there is an example of a set message type primitive and the results of casting a weak type to a strong type.

In the top screen capture you see the configuration of a set message type primitive. The SMO contains a weakly typed XSD:anyType field at /body/send/data which is cast to an actual type Order defined in the StoreLib library.

Looking at the screen capture in the middle of the slide you see the set message type primitive from a mediation flow diagram. The message type of the input terminal is sendRequestMsg. The message type for the output terminal is also sendRequestMsg, but it has been augmented with the additional information about /body/send/data that it is actually of type Order.

Moving your attention to the screen captures in the lower part of the slide you see the result of using the XPath expression builder dialog to expand /body/send/data. The example on the left shows the expansion before the set message type and the expansion on the right is after the set message type.

From this example, you can see that use of this primitive allows you to more effectively build your flow because the tools can make use of the type data you provide.



IBM Software Group

Business object map primitive

Existing 6.0.2	▪ Business object maps can not be used in mediation flows
New 6.1	▪ Business object map primitive added ▸ Enables use of business object maps in a mediation flow ▸ Same business object maps that are used for: ▪ Parameter mapping in interface maps ▪ Mapping service APIs ▸ Business object map editor invoked from mediation flow editor
Benefits	▪ Utilize existing investment in business object maps ▪ Enables relationship service within mediation flows ▪ Enables handling of business graphs within a mediation flow ▪ Configure event settings to raise CEI events

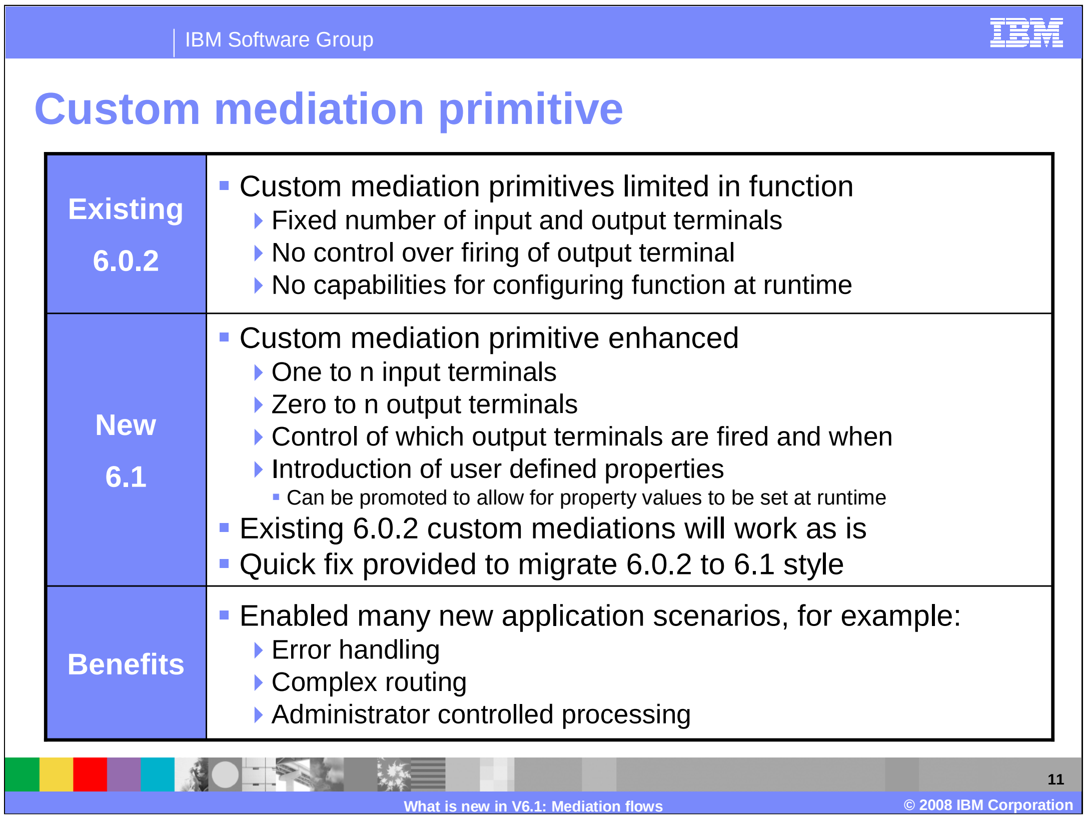
10

What is new in V6.1: Mediation flows

© 2008 IBM Corporation

In version 6.0.2 it is not possible to use a business object map in a mediation flow. In version 6.1 the business object map primitive allows you to integrate the use of business object maps into the mediation flow. These are the same business object maps that are used in WebSphere Process Server for parameter mapping within interface maps and are also used through the mapping service APIs. The business object map editor is invoked from the mediation flow editor when configuring a business object map primitive.

There are several benefits derived from the use of business object maps in mediation flows. If you already have an investment in developing business object maps you can now use them in mediation flows. There are also some capabilities that business object maps enable that are not possible using XSL transformations. These include the use of the relationship service for identity and lookup relationships and the use of business graphs which are important for some adapter usage scenarios. Also, the ability to configure the raising of CEI events during processing of the map is enabled.


IBM Software Group

Custom mediation primitive

Existing 6.0.2	▪ Custom mediation primitives limited in function ▸ Fixed number of input and output terminals ▸ No control over firing of output terminal ▸ No capabilities for configuring function at runtime
New 6.1	▪ Custom mediation primitive enhanced ▸ One to n input terminals ▸ Zero to n output terminals ▸ Control of which output terminals are fired and when ▸ Introduction of user defined properties ▪ Can be promoted to allow for property values to be set at runtime ▪ Existing 6.0.2 custom mediations will work as is ▪ Quick fix provided to migrate 6.0.2 to 6.1 style
Benefits	▪ Enabled many new application scenarios, for example: ▸ Error handling ▸ Complex routing ▸ Administrator controlled processing

11

What is new in V6.1: Mediation flows

© 2008 IBM Corporation

In version 6.0.2 the custom mediation is a very useful primitive. However, it is somewhat limited in function if it is to be thought of as a general purpose mediation primitive. For example, there are a fixed number of input and output terminals, there is no control over when the output terminal is fired and nothing is provided to allow for runtime configuration of the primitive’s behavior.

In version 6.1 the custom mediation has been enhanced to address these limitations. The terminals can be customized so that you can have from one to n input terminals and 0 to n output terminals. From within the custom mediation your code can control which of these output terminals are fired and control where they are fired within the custom mediation logic. You can define properties to associate with the custom mediation and by promoting the properties you enable administrative control of their values at runtime.

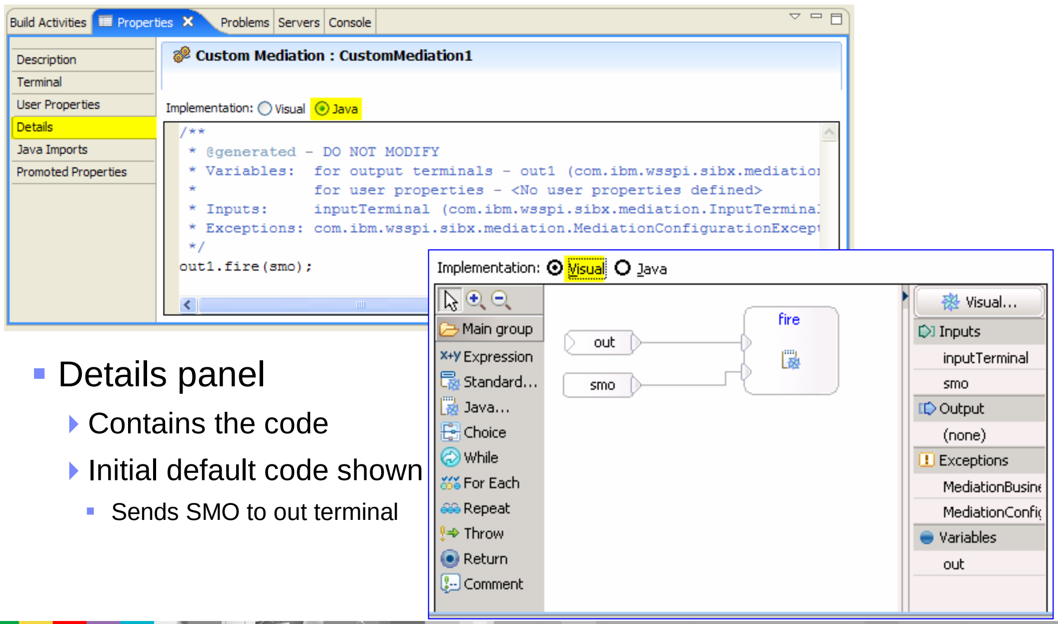
When you have existing version 6.0.2 custom mediations you have two choices. They will continue to run without change or you can use a quick fix capability to convert them from the version 6.0.2 style to the 6.1 style.

The benefit of this enhancement is that it enables many new application scenarios. One example is having an error handling custom mediation with several input terminals. You can wire every fail terminal from the primitives in your flow into this one custom mediation. This was not possible in 6.0.2 unless all of the fail terminals were of the same message type, which is not typically the case.

The next example is complex routing in the case where the routing decision requires more logic than can be done in a message filter primitive. The custom mediation can have several output terminals and the custom mediation can analyze the input service message object to determine which of the output terminals to fire.

Another example is allowing the administrator to control processing. For example, the custom mediation might optionally produce some problem determination information based on the setting of a user defined property which was promoted to allow the administrator to set it.

Custom mediation – default code



Custom Mediation : CustomMediation1

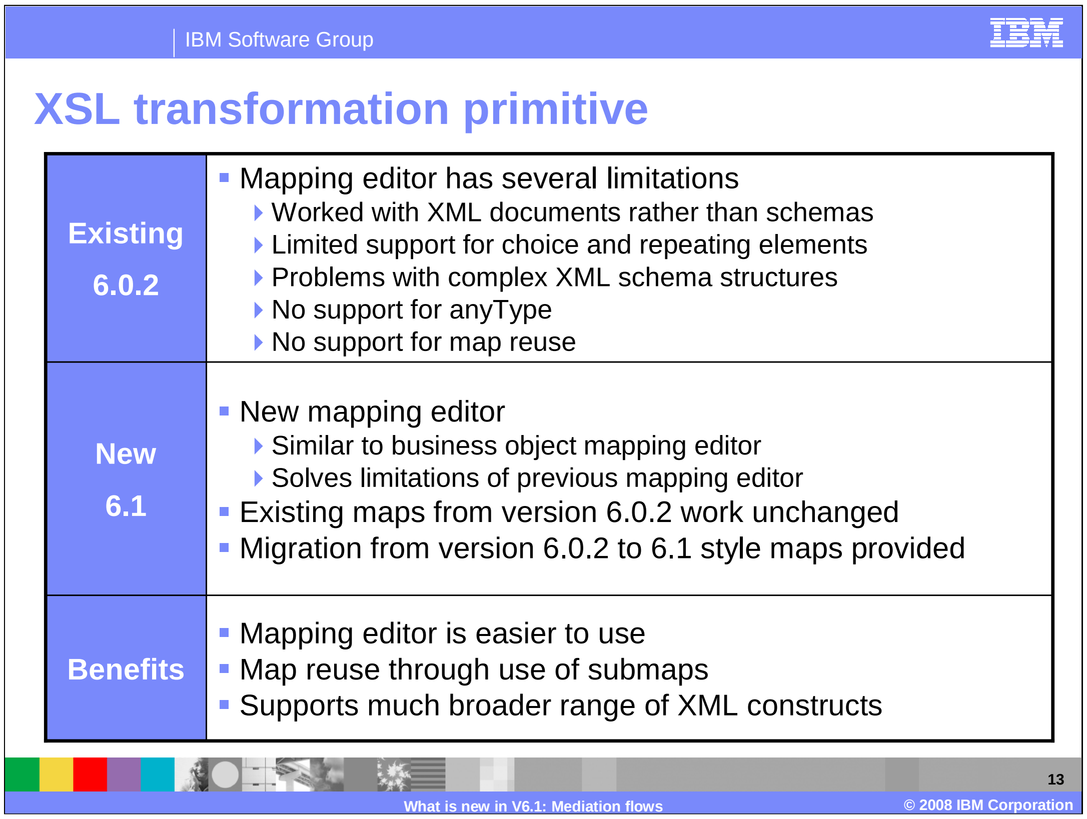
Implementation: Visual [✓ Java]

```
/**
 * @generated - DO NOT MODIFY
 * Variables:  for output terminals - out1 (com.ibm.wsspi.sibx.mediatio
 *             for user properties - <No user properties defined>
 * Inputs:     inputTerminal (com.ibm.wsspi.sibx.mediation.InputTermina
 * Exceptions: com.ibm.wsspi.sibx.mediation.MediationConfigurationExcep
 */
out1.fire(smo);
```

Implementation: [✓ Visual] Java

- Details panel
  - Contains the code
  - Initial default code shown
    - Sends SMO to out terminal

This panel shows examples of a Java and a visual custom mediation. The code shown is the default code that is provided when you first create the custom mediation. The mediation is passed an input named smo which contains the entire service message object. The code fires the default output terminal out1 returning the service message object.


IBM Software Group

XSL transformation primitive

Existing 6.0.2	▪ Mapping editor has several limitations ▸ Worked with XML documents rather than schemas ▸ Limited support for choice and repeating elements ▸ Problems with complex XML schema structures ▸ No support for anyType ▸ No support for map reuse
New 6.1	▪ New mapping editor ▸ Similar to business object mapping editor ▸ Solves limitations of previous mapping editor ▪ Existing maps from version 6.0.2 work unchanged ▪ Migration from version 6.0.2 to 6.1 style maps provided
Benefits	▪ Mapping editor is easier to use ▪ Map reuse through use of submaps ▪ Supports much broader range of XML constructs

13

What is new in V6.1: Mediation flows

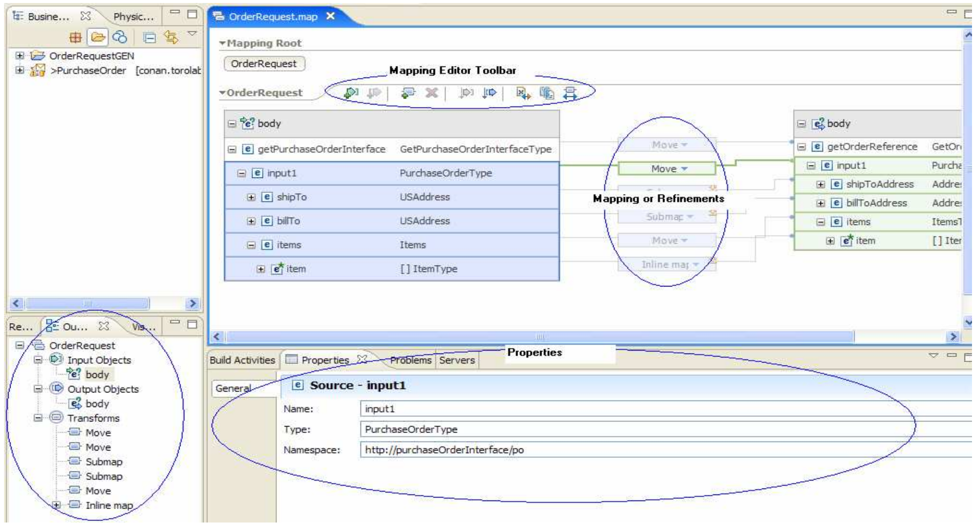
© 2008 IBM Corporation

In version 6.0.2 the mapping editor used with the XSL transformation primitive has several limitations. The editor works with XML documents rather than with XML schema definitions and has limited support for repeating elements and choice constructs. Many complex XML structures are not handled correctly and there is no support for anyType. There is also no support for map reuse.

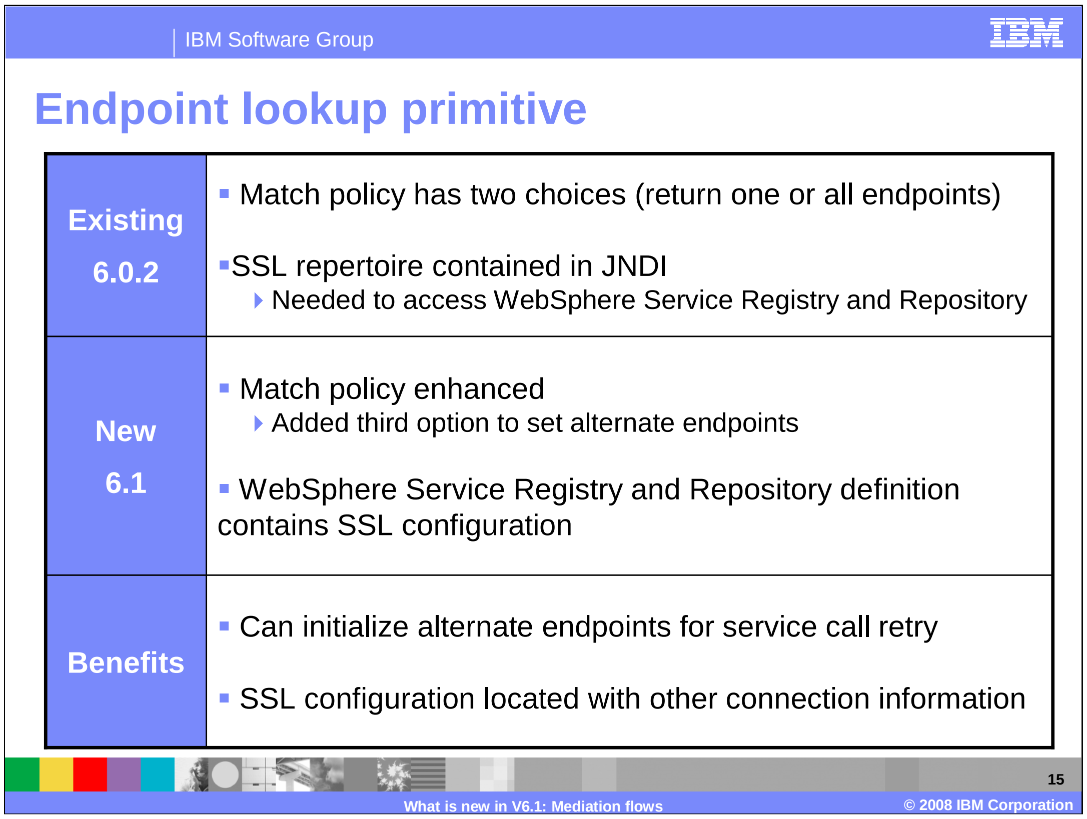
In version 6.1 the XSL transformation primitive has been enhanced to make use of a new mapping editor. The visual presentation and interaction with the editor are similar to those of the business object mapping editor and the limitations of the previous editor have been solved. If you have existing version 6.0.2 mediations containing XSL transformations they will continue to work as is. However, the XSL transformation provides a migrate button that will take the version 6.0.2 style maps and convert them into version 6.1 maps.

Advantages gained from this enhancement include ease of use of the mapping editor and the capability to reuse maps through the use of submaps. Also, a much broader range of XML constructs is now supported.

XSL transformation – mapping editor



This is a screen capture of the mapping editor used with the XSL transformation. On the top right is the map itself with both the input and output objects expanded. Transformations are defined by connecting an input field to an output field and then defining what type of transform it is. In the bottom is the properties view which is used to configure each transform. The mapping editor contains a toolbar that provides easy access to several functions. Finally, the lower left shows the outline view of the map.



Endpoint lookup primitive

Existing 6.0.2	▪ Match policy has two choices (return one or all endpoints) ▪SSL repertoire contained in JNDI ‣ Needed to access WebSphere Service Registry and Repository
New 6.1	▪ Match policy enhanced ‣ Added third option to set alternate endpoints ▪ WebSphere Service Registry and Repository definition contains SSL configuration
Benefits	▪ Can initialize alternate endpoints for service call retry ▪ SSL configuration located with other connection information

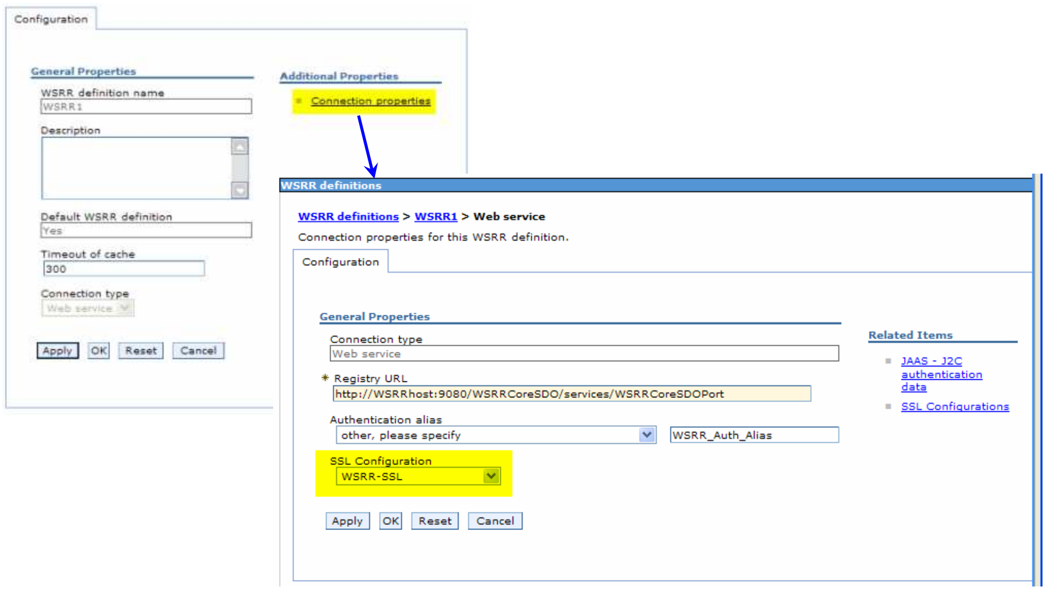
Two separate enhancements have been made to the endpoint lookup primitive.

The first enhancement relates to the match policy property of the primitive. In version 6.0.2 the match policy provided two choices, either to return only one endpoint that matched the request or to return all endpoints that matched the request. In version 6.1 a third option has been added to match policy which will return all matching endpoints and use them to initialize the alternate target address array. This alternate target address array is a new addition in version 6.1 used with the service call retry function.

The second enhancement relates to setting the SSL repertoire needed to connect to the WebSphere Service Registry and Repository. In version 6.0.2 you were required to bind into the JNDI namespace an object that contained the SSL configuration information. In version 6.1 the information has been added to the connection properties of the WebSphere Service Registry and Repository definition which is part of the administrative configuration of the server. The advantage is that the SSL information is now located with other connection information and is therefore generally easier to administer.

IBM Software Group

Endpoint lookup – SSL configuration



Configuration

General Properties

WSRR definition name: WSRR1

Description:

Default WSRR definition: Yes

Timeout of cache: 300

Connection type: Web service

Additional Properties

Connection properties

WSRR definitions

WSRR definitions > WSRR1 > Web service

Connection properties for this WSRR definition.

Configuration

General Properties

Connection type: Web service

* Registry URL: http://WSRRhost:9080/WSRRCoreSDO/services/WSRRCoreSDOPort

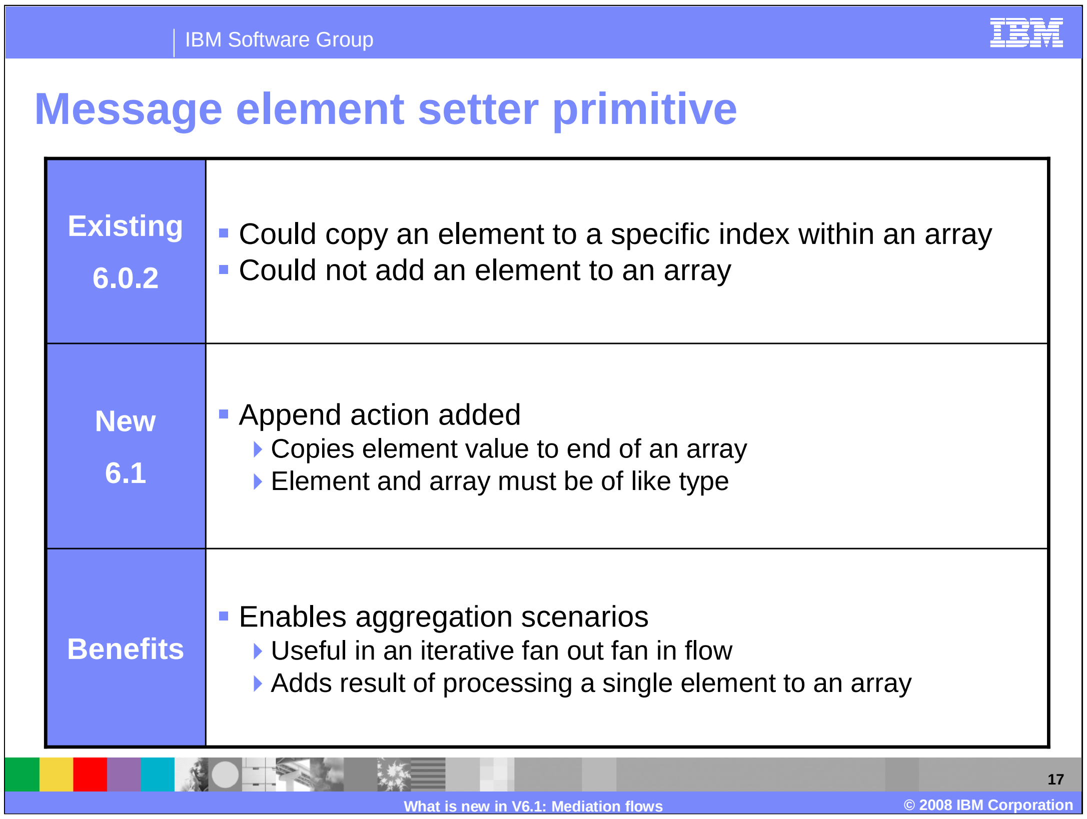
Authentication alias: other, please specify — WSRR_Auth_Alias

SSL Configuration: WSRR-SSL

Related Items

- JAAS - J2C authentication data
- SSL Configurations

This slide shows the panels from the administrative console used to configure the SSL information for a WebSphere Service Registry and Repository. The upper left is the WebSphere Service Registry and Repository definition panel which contains a link to the connection properties panel where the SSL configuration is specified.


IBM Software Group

Message element setter primitive

Existing 6.0.2	▪ Could copy an element to a specific index within an array ▪ Could not add an element to an array
New 6.1	▪ Append action added ▸ Copies element value to end of an array ▸ Element and array must be of like type
Benefits	▪ Enables aggregation scenarios ▸ Useful in an iterative fan out fan in flow ▸ Adds result of processing a single element to an array

17

What is new in V6.1: Mediation flows

© 2008 IBM Corporation

In version 6.0.2 the message element setter primitive can be used to set a value to a specific index location in an array, but it is not possible to add an element to an array.

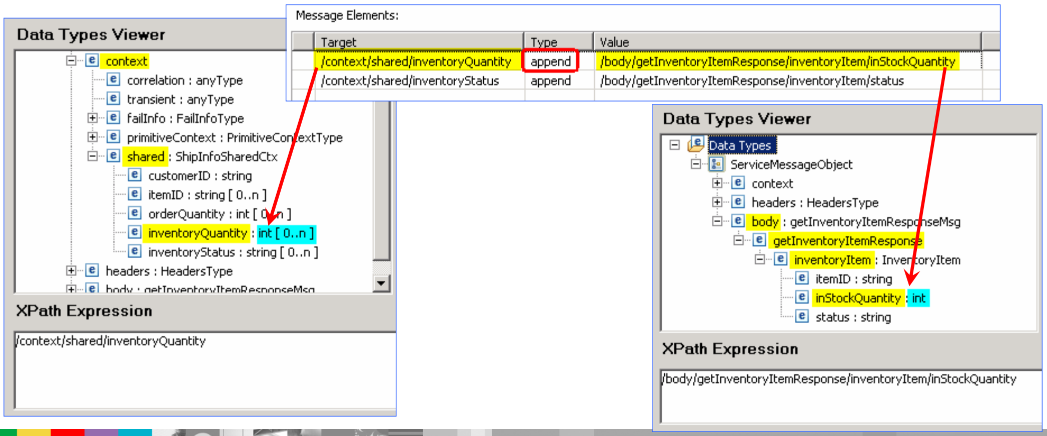
In version 6.1 the append action has been added which will append an element to the end of an array, provided the element being appended is of like type with the array.

This enhancement is a key contributor to enabling the aggregation scenarios that were previously described. During iterative processing it is used to append the results of the processing of a single element to an array in the shared context.

IBM Software Group

Message element setter – append

- Append capability
  - Element can be a simple or complex type
  - Array must be at the leaf of the XPath expression



Message Elements:

Target	Type	Value
/context/shared/inventoryQuantity	append	/body/getInventoryItemResponse/inventoryItem/inStockQuantity
/context/shared/inventoryStatus	append	/body/getInventoryItemResponse/inventoryItem/status

Data Types Viewer

```
context
  correlation : anyType
  transient : anyType
  failInfo : FailInfoType
  primitiveContext : PrimitiveContextType
  shared : ShipInfoSharedCtx
    customerID : string
    itemID : string [ 0..n ]
    orderQuantity : int [ 0..n ]
    inventoryQuantity : int [ 0..n ]
    inventoryStatus : string [ 0..n ]
headers : HeadersType
body : getInventoryItemResponseMsg
```

XPath Expression

```
/context/shared/inventoryQuantity
```

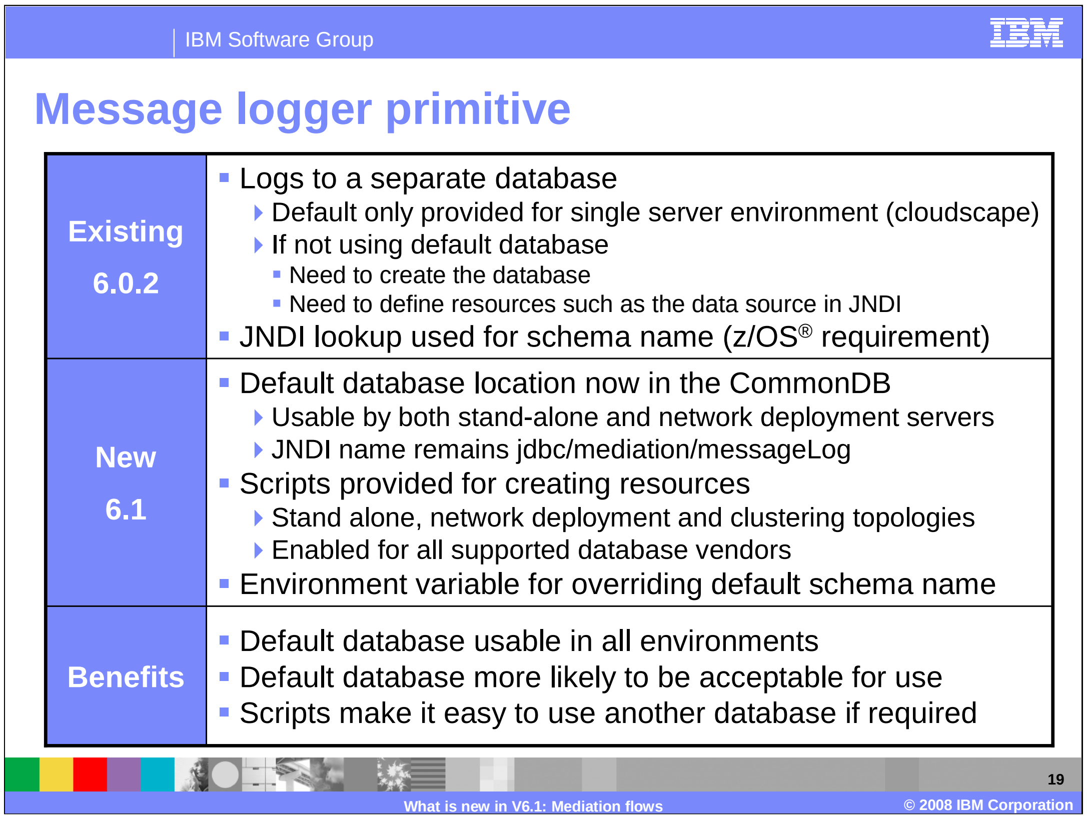
Data Types Viewer

```
Data Types
  ServiceMessageObject
    context
    headers : HeadersType
    body : getInventoryItemResponseMsg
      getInventoryItemResponse
        inventoryItem : InventoryItem
          itemID : string
          inStockQuantity : int
          status : string
```

XPath Expression

```
/body/getInventoryItemResponse/inventoryItem/inStockQuantity
```

This slide illustrates the use of the append option for a message element setter. At the top you can see the message elements table which is a property of the message element setter. Notice that the target is configured to an array of int values located at /context/shared/InventoryQuantity and the appended int value is being copied from /body/getInventoryItemResponse/inventoryItem/inStockQuantity. The value being copied to the target array must be of the same type as the target array. In this case it is a simple type, but complex types can be used also, as long as the types match. Also, the array being appended to must be at the leaf of the XPath expression.



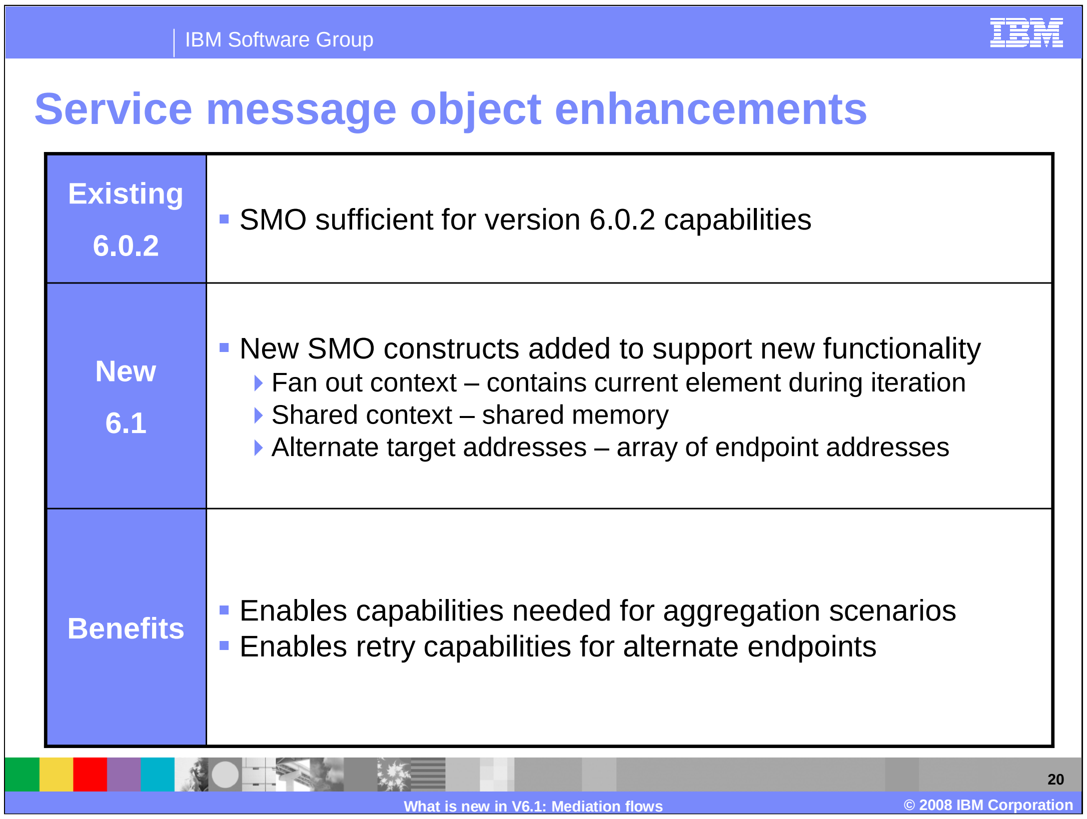
Message logger primitive

Existing 6.0.2	▪ Logs to a separate database ‣ Default only provided for single server environment (cloudscape) ‣ If not using default database ▪ Need to create the database ▪ Need to define resources such as the data source in JNDI ▪ JNDI lookup used for schema name (z/OS® requirement)
New 6.1	▪ Default database location now in the CommonDB ‣ Usable by both stand-alone and network deployment servers ‣ JNDI name remains jdbc/mediation/messageLog ▪ Scripts provided for creating resources ‣ Stand alone, network deployment and clustering topologies ‣ Enabled for all supported database vendors ▪ Environment variable for overriding default schema name
Benefits	▪ Default database usable in all environments ▪ Default database more likely to be acceptable for use ▪ Scripts make it easy to use another database if required

In version 6.0.2 the message logger primitive logs to a separate database and the provided default cloudscape database is only usable in a stand-alone server environment. For any case where you choose to or are required to use another database you need to create the database and define the resources used with it. Additionally, for z/OS where there is a requirement that the schema name also be supplied, a JNDI resource needs to be created containing the database schema name.

In version 6.1 the default database used by the message logger is now part of the CommonDB which will always be defined for the server environment. This database is usable for network deployment servers and stand-alone servers. The default JNDI name remains the same while pointing to the new default database and therefore existing version 6.0.2 message logger primitives using the default do not need to be reconfigured. Although the default database is now much more usable there are still situations where you want to create a separate database. To help accomplish that, scripts have been provided that deal with all supported database vendors and work for all stand-alone, network deployment and clustering topologies. Finally, the schema name needed for z/OS can now be set with an environment variable.

The advantages of these changes include the fact that the default database is both usable in all environments and is also more likely to be an acceptable database to use. In the cases where it is not, the scripts needed to create a database and associated resources are provided, making your job much easier.


IBM Software Group

IBM

Service message object enhancements

Existing 6.0.2	▪ SMO sufficient for version 6.0.2 capabilities
New 6.1	▪ New SMO constructs added to support new functionality ▸ Fan out context – contains current element during iteration ▸ Shared context – shared memory ▸ Alternate target addresses – array of endpoint addresses
Benefits	▪ Enables capabilities needed for aggregation scenarios ▪ Enables retry capabilities for alternate endpoints

20

What is new in V6.1: Mediation flows

© 2008 IBM Corporation

The service message object as defined in version 6.0.2 is sufficient for what it needs to do in that version. However, enhancements to version 6.1 placed additional requirements on the service message object.

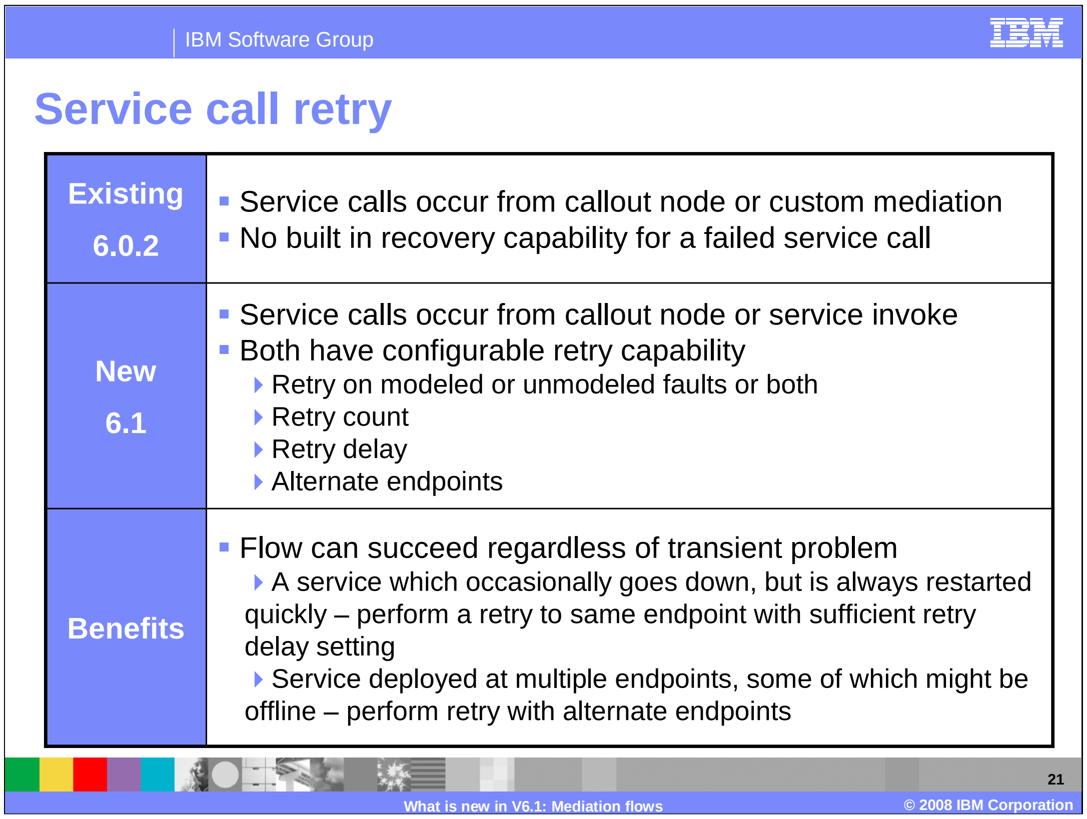
These include the fan out context, which is used by the fan out primitive when in iterate mode as a place to put the value of the current element being processed. This allows the primitives between the fan out and fan in to know which element they are processing.

The shared context provides a shared memory area that is required during aggregation scenarios using fan out and fan in.

This shared memory area is needed because the service message object is cloned from the original at the start of each iteration.

Finally, there are the alternate target addresses which are used with service call retry support.

The benefit of these additions is that they provide the necessary data areas needed to support aggregation and service call retry scenarios.


IBM Software Group

Service call retry

Existing 6.0.2	▪ Service calls occur from callout node or custom mediation ▪ No built in recovery capability for a failed service call
New 6.1	▪ Service calls occur from callout node or service invoke ▪ Both have configurable retry capability ▸ Retry on modeled or unmodeled faults or both ▸ Retry count ▸ Retry delay ▸ Alternate endpoints
Benefits	▪ Flow can succeed regardless of transient problem ▸ A service which occasionally goes down, but is always restarted quickly – perform a retry to same endpoint with sufficient retry delay setting ▸ Service deployed at multiple endpoints, some of which might be offline – perform retry with alternate endpoints

21

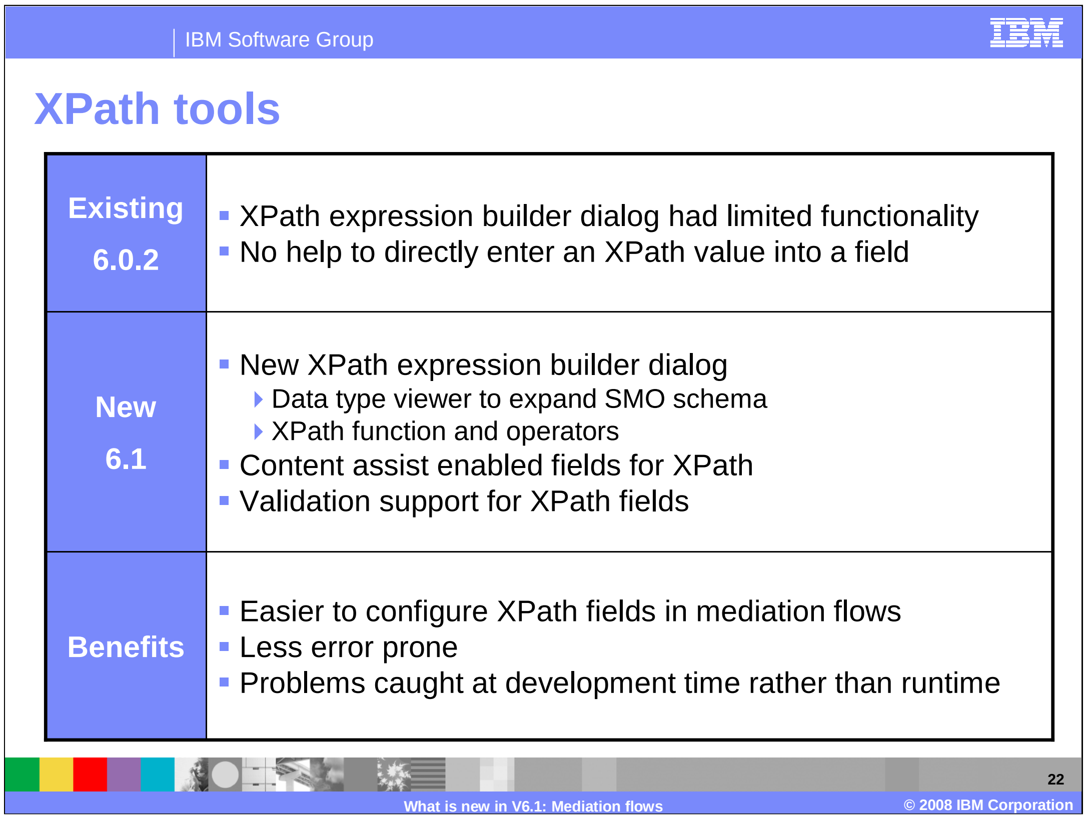
What is new in V6.1: Mediation flows

© 2008 IBM Corporation

In version 6.0.2 calls to services only occur through the callout node or from a custom mediation primitive. If a service call fails there is no built in mechanism to attempt to recover from the failure.

In version 6.1 service calls can occur from the callout node or from a service invoke primitive. Both of these now have retry capabilities built in. Through configuration properties you can specify if you want failing service calls to be retried, and if so, whether modeled faults, unmodeled faults or both types of failures should be retried. Along with that you specify the retry count to indicate how many times to retry before giving up and the retry delay which indicates how long to wait between retries. Finally, if the service invoke or callout node are utilizing dynamic endpoints, the retries can be configured to use alternate endpoints.

Clearly the advantage of this is that a flow no longer needs to fail because of transient problems. For example, the failure might be caused by a problem in the service which caused it to stop. If the service was then immediately restarted, using retry with a retry delay solves the failure. The delay allows for the time needed for the service to come back up so that the retry is successful. Another example is when a service has multiple providers but there are times when one or more of the providers is offline. In this case, when the call is to a service that is offline, using retry with alternate endpoints enables a service provider endpoint to be found that is currently up and running.



XPath tools

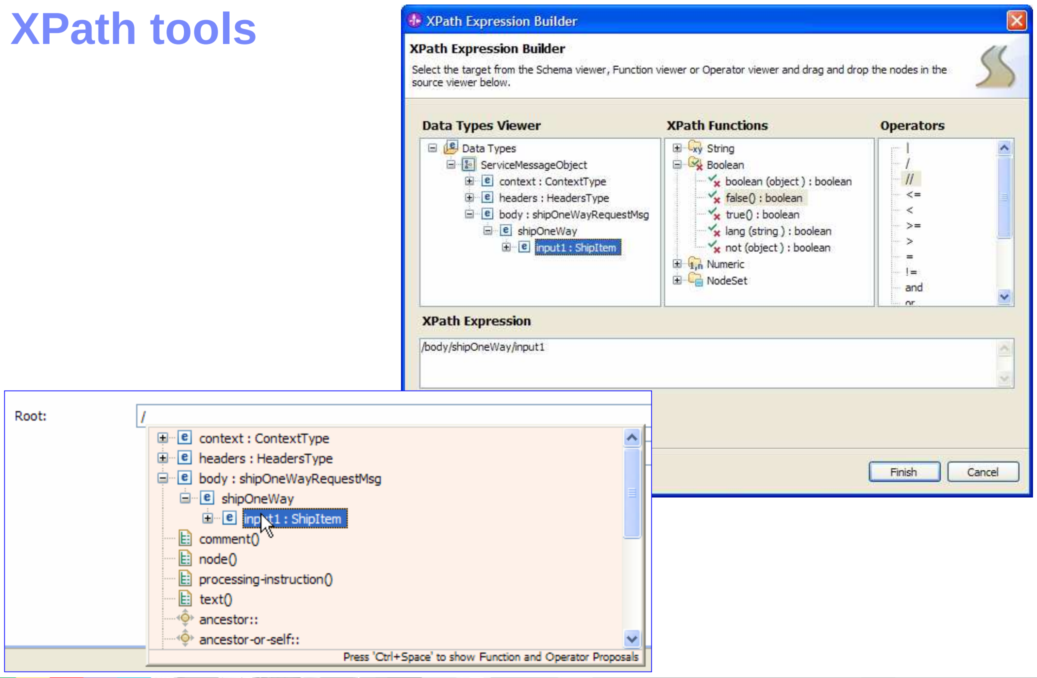
Existing 6.0.2	▪ XPath expression builder dialog had limited functionality ▪ No help to directly enter an XPath value into a field
New 6.1	▪ New XPath expression builder dialog ▸ Data type viewer to expand SMO schema ▸ XPath function and operators ▪ Content assist enabled fields for XPath ▪ Validation support for XPath fields
Benefits	▪ Easier to configure XPath fields in mediation flows ▪ Less error prone ▪ Problems caught at development time rather than runtime

In version 6.0.2 the XPath expression builder dialog is useful for building some XPath expressions. However, its function is limited so that sometimes you need to complete the expression by hand coding a part of it. Also, if entering an XPath expression directly into a field without using the XPath expression builder there is no help provided and the expression has to be totally coded by hand.

In version 6.1 there is a new XPath expression builder dialog that provides a greater level of functionality. It has a data type viewer from which you can expand the service message object schema and it also has XPath function and XPath operator viewers that help you build up the expression. Additionally, for entry of XPath expressions directly into a field without using the dialog there is now content assist provided and validation checking of the XPath expression.

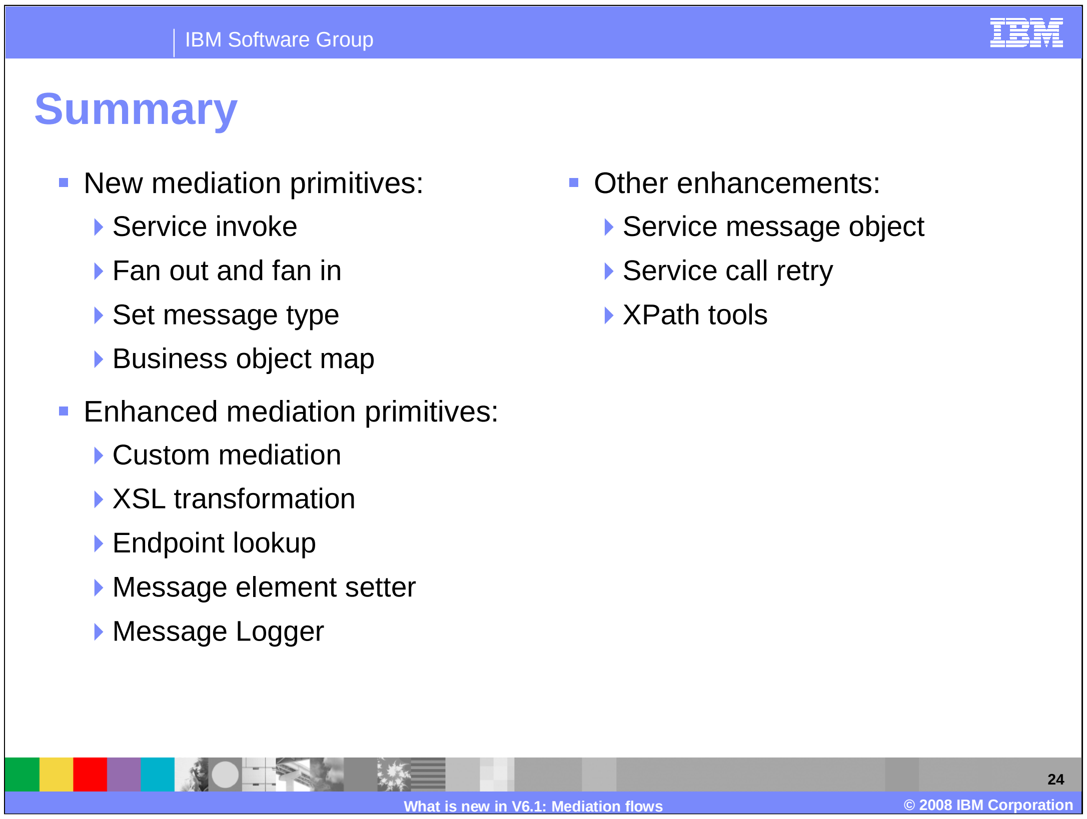
These enhancements make it much easier to specify XPath expressions when developing a mediation flow and therefore the process is less prone to error. The validation checking helps catch errors at development time rather than discovery of errors being deferred to runtime.

XPath tools



In the upper right is a screen capture of the XPath expression builder. The three viewers are used to select appropriate parts of the expression to be added to the expression built up in the bottom part of the dialog.

In the lower left there is an example of content assist for entering an XPath expression directly into a field.


IBM Software Group

Summary

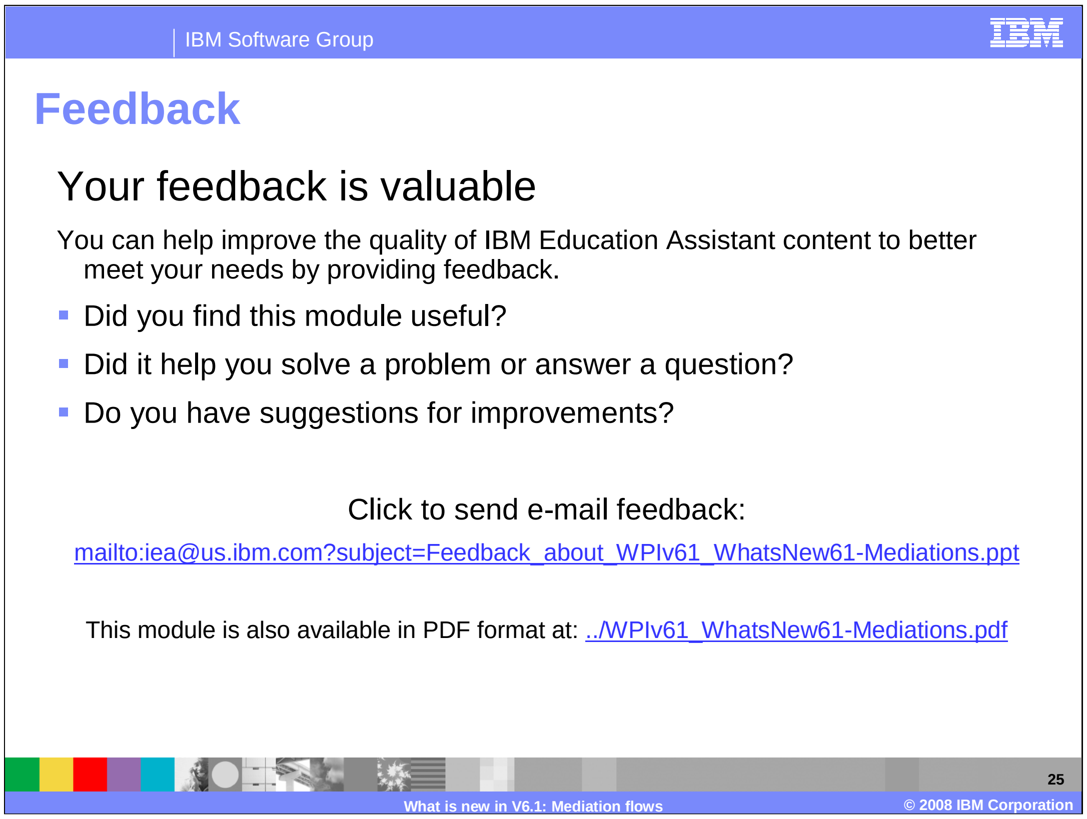
- New mediation primitives:
  - Service invoke
  - Fan out and fan in
  - Set message type
  - Business object map
- Enhanced mediation primitives:
  - Custom mediation
  - XSL transformation
  - Endpoint lookup
  - Message element setter
  - Message Logger
- Other enhancements:
  - Service message object
  - Service call retry
  - XPath tools

24

What is new in V6.1: Mediation flows

© 2008 IBM Corporation

In this presentation you were introduced to the new features for version 6.1. These included the new mediation primitives which are the service invoke primitive, the fan out and fan in primitives, the set message type primitive and the business object map primitive. There are many existing primitives which were enhanced, including the custom mediation primitive, the XSL transformation primitive, the endpoint lookup primitive, the message element setter primitive and the message logger primitive. Finally, other enhancements were described such as the modifications to the service message object, the ability to perform retry for failing service calls and enhanced tools for specifying XPath expressions.



Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

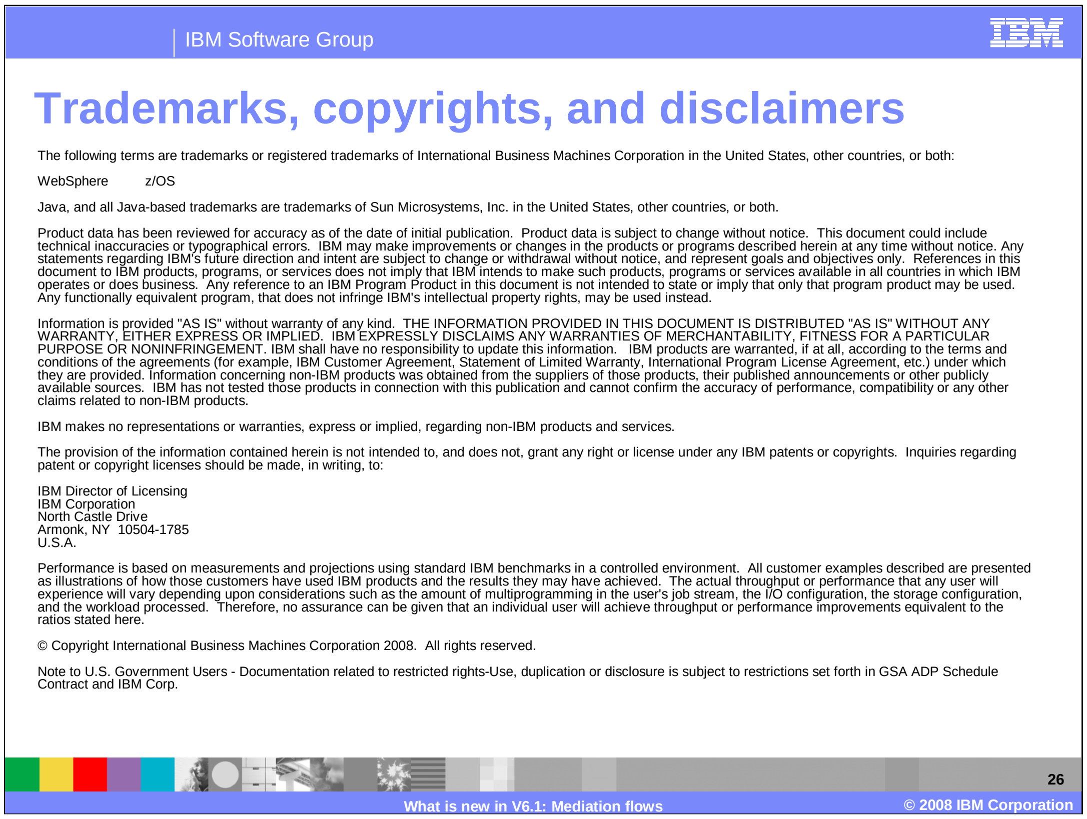
- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send e-mail feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_WPIv61_WhatsNew61-Mediations.ppt

This module is also available in PDF format at: ../WPIv61_WhatsNew61-Mediations.pdf

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



Trademarks, copyrights, and disclaimers