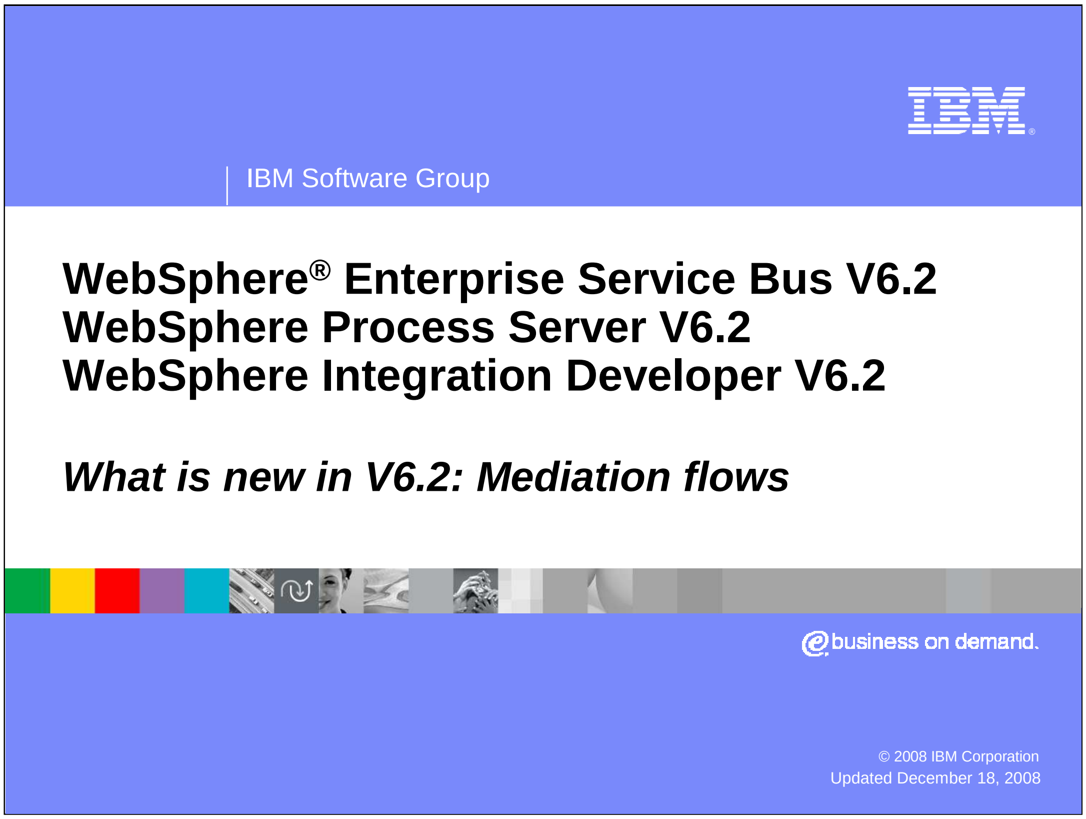


IBM Software Group

WebSphere® Enterprise Service Bus V6.2
WebSphere Process Server V6.2
WebSphere Integration Developer V6.2

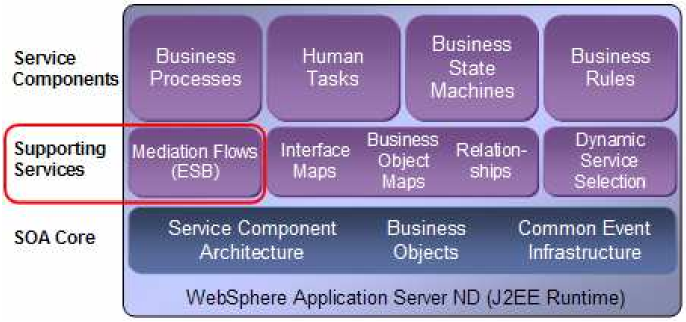
What is new in V6.2: Mediation flows

business on demand.

Updated December 18, 2008

This presentation provides an introduction to the new function delivered in version 6.2 for mediation flows in WebSphere Enterprise Service Bus, WebSphere Process Server and WebSphere Integration Developer.

Goals



Service Components: Business Processes, Human Tasks, Business State Machines, Business Rules

Supporting Services: Mediation Flows (ESB), Interface Maps, Business Object Maps, Relationships, Dynamic Service Selection

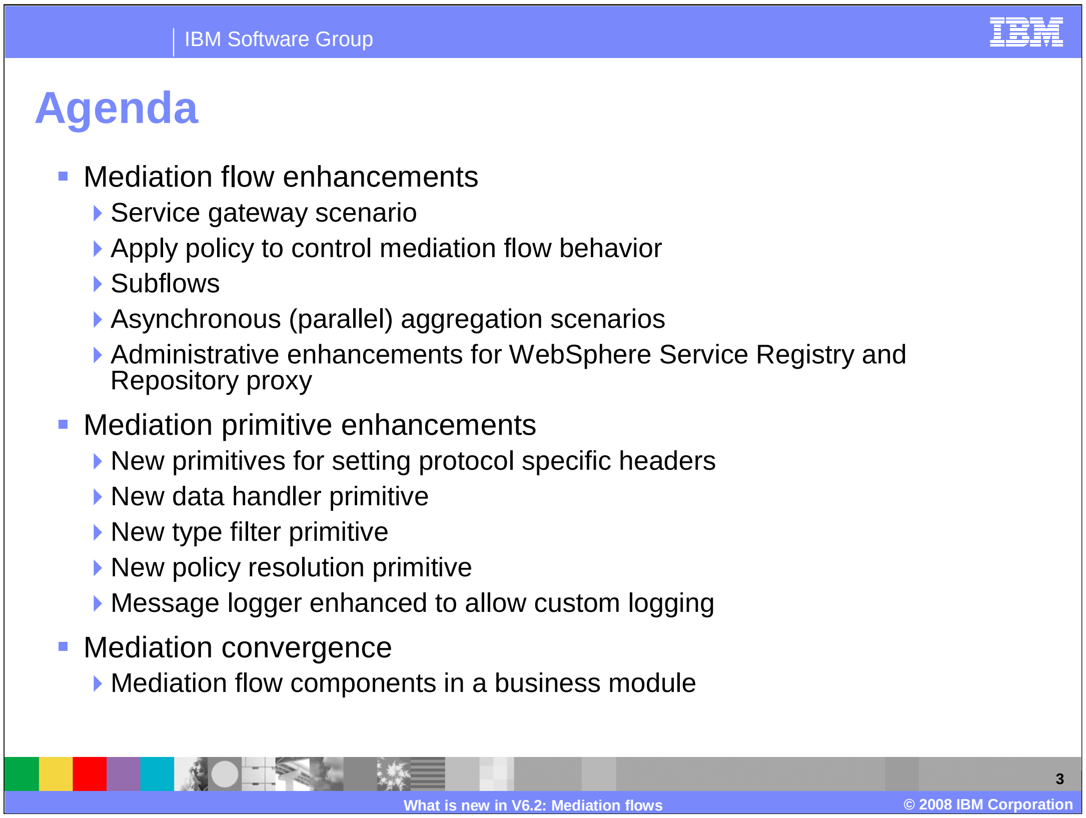
SOA Core: Service Component Architecture, Business Objects, Common Event Infrastructure

WebSphere Application Server ND (J2EE Runtime)

- Introduce enhancements to mediation flows made for V6.2
- Applies to:
  - WebSphere Enterprise Service Bus
  - WebSphere Process Server
  - WebSphere Integration Developer
- Prerequisites to understanding this presentation:
  - Knowledge of mediation flows in V6.1

The goal of this presentation is to introduce you to the enhancements that have been made to mediation flows for version 6.2. 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 6.1 of the products.



IBM Software Group

Agenda

- Mediation flow enhancements
  - Service gateway scenario
  - Apply policy to control mediation flow behavior
  - Subflows
  - Asynchronous (parallel) aggregation scenarios
  - Administrative enhancements for WebSphere Service Registry and Repository proxy
- Mediation primitive enhancements
  - New primitives for setting protocol specific headers
  - New data handler primitive
  - New type filter primitive
  - New policy resolution primitive
  - Message logger enhanced to allow custom logging
- Mediation convergence
  - Mediation flow components in a business module

3

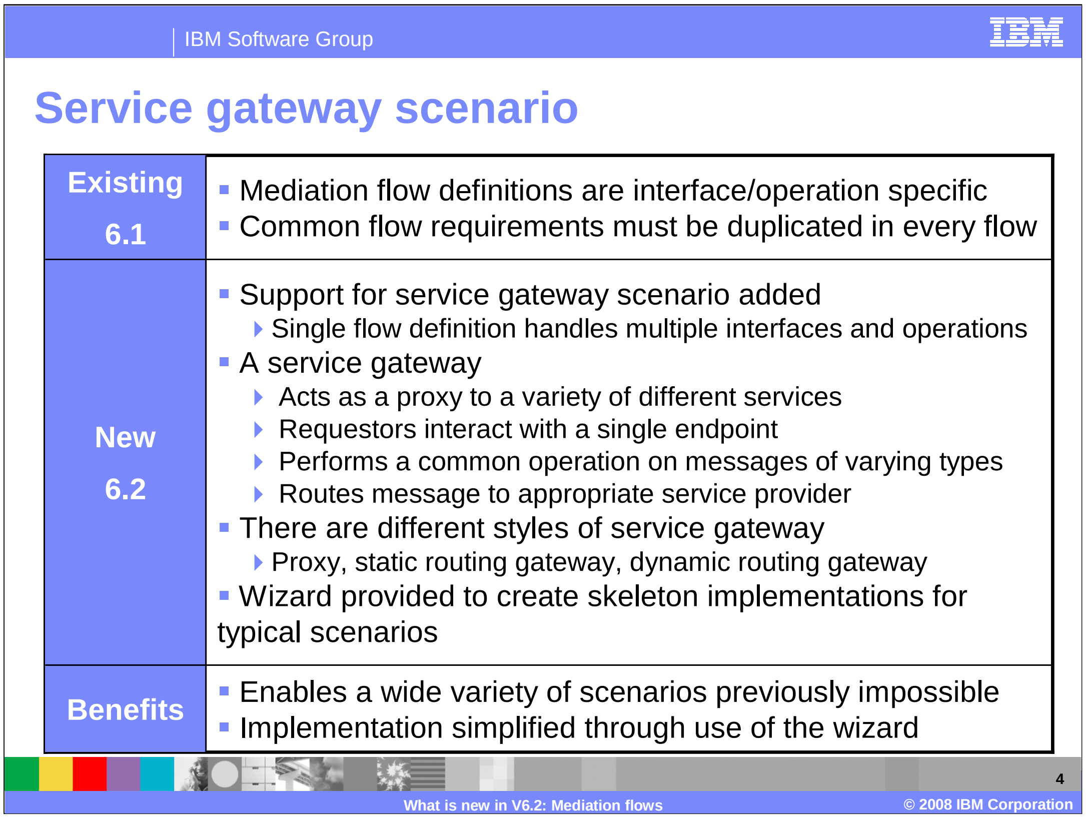
What is new in V6.2: Mediation flows

© 2008 IBM Corporation

The new features for version 6.2 are presented as indicated in this agenda. The first grouping are those features that have to do with mediation flows. The service gateway scenario enables mediation flows that provide common processing across multiple interfaces and operations. The ability to use policies to control mediation flow behavior on a per flow instance basis has been added. The introduction of subflows enables reuse within the mediation programming model. Aggregation scenarios enabled by the fan out and fan in primitives are now able to make use of asynchronous parallel processing for service invocations within the flow. The administrative proxy used to interface with the WebSphere Service Registry and Repository had been enhanced to make administration of the registry easier.

The next group of enhancements is for new and updated mediation primitives. There are four new primitives specifically designed for use accessing and updating headers in the system management object. There is a data handler primitive which you configure to do conversions between native data formats and business object formats. The new type filter primitive allows you to make routing decisions based on element types within the service message object. The policy resolution primitive interacts with a repository to look up policies that control mediation behavior. The existing message logger primitive has been enhanced to allow for custom logging implementations to be used in addition to the existing logging to a database functionality.

The last topic addressed is mediation convergence, which involves enabling mediation flow components to be used in business modules.



IBM Software Group

Service gateway scenario

Existing 6.1	▪ Mediation flow definitions are interface/operation specific ▪ Common flow requirements must be duplicated in every flow
New 6.2	▪ Support for service gateway scenario added ▸ Single flow definition handles multiple interfaces and operations ▪ A service gateway ▸ Acts as a proxy to a variety of different services ▸ Requestors interact with a single endpoint ▸ Performs a common operation on messages of varying types ▸ Routes message to appropriate service provider ▪ There are different styles of service gateway ▸ Proxy, static routing gateway, dynamic routing gateway ▪ Wizard provided to create skeleton implementations for typical scenarios
Benefits	▪ Enables a wide variety of scenarios previously impossible ▪ Implementation simplified through use of the wizard

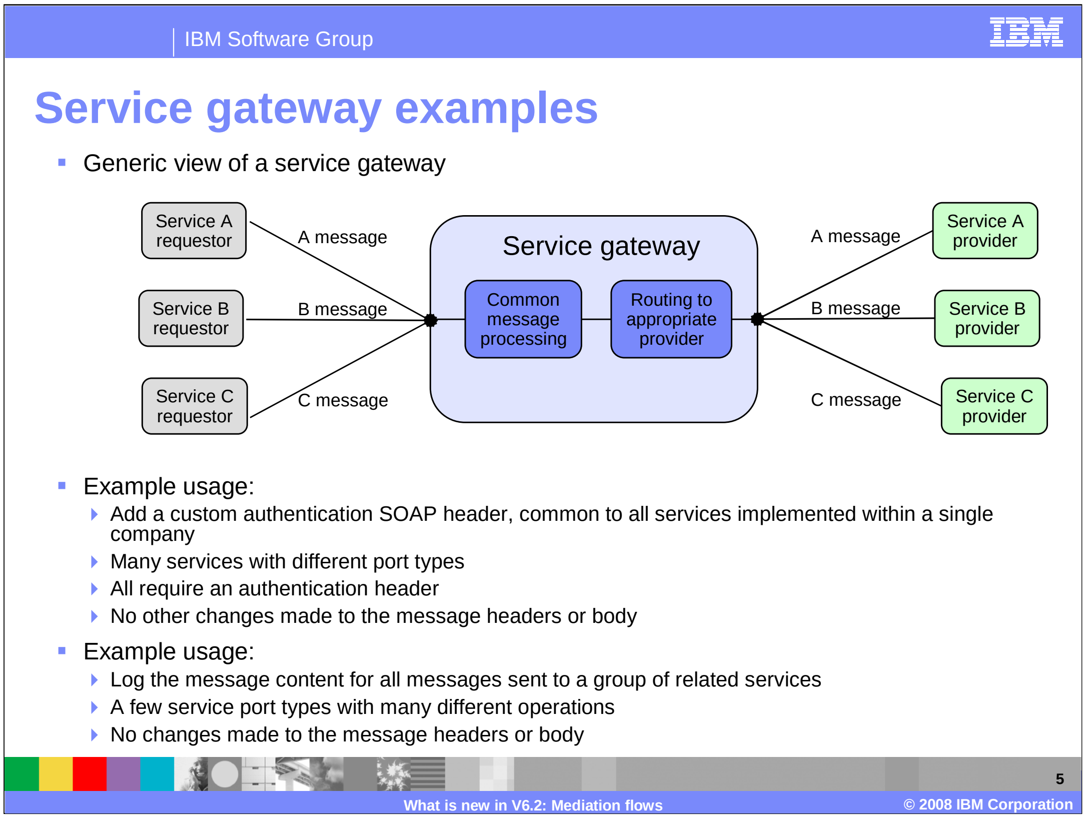
4

What is new in V6.2: Mediation flows

© 2008 IBM Corporation

This enhancement provides for the enablement of service gateway scenarios. In version 6.1 a mediation flow is always specific to a single operation. For all interfaces defined on a mediation flow component, there needs to be a separate flow definition for every operation defined in the interfaces. If there are common flow requirements, then they need to be implemented in every one of these flows. With the introduction of service gateway flow capabilities in version 6.2 it is possible to have a single flow definition which can handle multiple operations from multiple interfaces. A service gateway can be a proxy to one or more services. Clients interact with a single gateway endpoint. The gateway performs some common operation on messages of varying types and then forwards them to the appropriate service provider. There are different requirements that can be satisfied by a gateway which result in differing implementations or styles of gateway. These include the proxy gateway, the static routing gateway, and the dynamic routing gateway. The WebSphere Integration Developer provides a wizard that can generate a skeleton implementation for various gateway styles based on a set of questions about your gateway requirements.

The benefit of this enhancement is the enablement of many scenarios that were previously impossible or too difficult to implement. The wizard also provides for easier development of a gateway implementation.



IBM Software Group

Service gateway examples

- Generic view of a service gateway

Service A requestor
A message
Service B requestor
B message
Service C requestor
C message
Service gateway
Common message processing
Routing to appropriate provider
A message
Service A provider
B message
Service B provider
C message
Service C provider

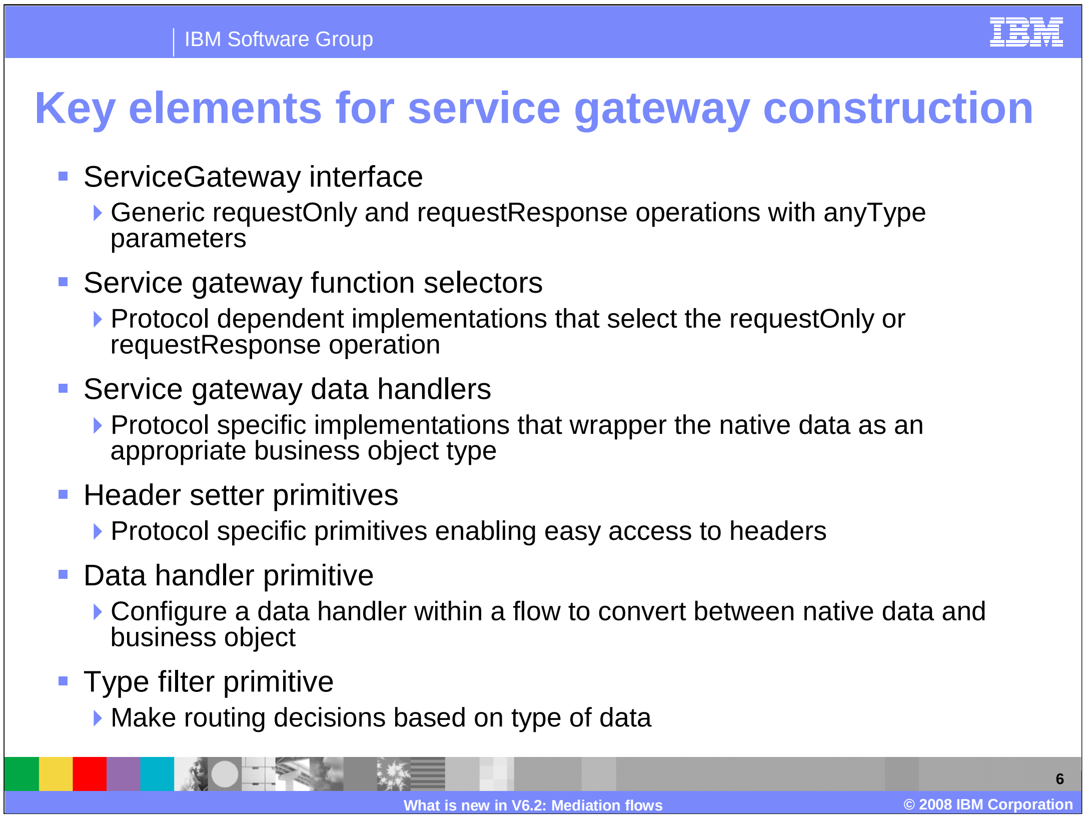
- Example usage:
  - Add a custom authentication SOAP header, common to all services implemented within a single company
  - Many services with different port types
  - All require an authentication header
  - No other changes made to the message headers or body
- Example usage:
  - Log the message content for all messages sent to a group of related services
  - A few service port types with many different operations
  - No changes made to the message headers or body

5

What is new in V6.2: Mediation flows

© 2008 IBM Corporation

The graphic in this slide illustrates a generic view of a service gateway. On the left side are some clients sending messages of different types to a single endpoint for the gateway. The gateway performs some kind of common processing on these messages. The gateway then does some kind of logic to determine where to route the message and forwards to the appropriate service provider. There are a couple of examples provided to help illustrate this. The first is for a company that uses custom authentication SOAP headers. They have many different services with different port types that all require this custom header. The gateway can be used to add the headers without making any other changes to the message headers or body. The next example is for a group of related services which have a requirement to log all messages. There a few services but they have many operations. Sending all the messages through the service gateway allows them to be logged and forwarded.



Key elements for service gateway construction

- ServiceGateway interface
  - Generic requestOnly and requestResponse operations with anyType parameters
- Service gateway function selectors
  - Protocol dependent implementations that select the requestOnly or requestResponse operation
- Service gateway data handlers
  - Protocol specific implementations that wrapper the native data as an appropriate business object type
- Header setter primitives
  - Protocol specific primitives enabling easy access to headers
- Data handler primitive
  - Configure a data handler within a flow to convert between native data and business object
- Type filter primitive
  - Make routing decisions based on type of data

This slide identifies the key elements that are used in a service gateway implementation. The first three are specific to a service gateway and are always used. The last three, the primitives, are used according to the gateway scenario being implemented, but can also have applicability to non-gateway scenarios.

The service gateway interface provides two operations, one for request response operations and the other for one way operations. All the parameters are defined to be anyType. This definition fits any possible operation. A service gateway always uses this interface for incoming messages, and might also use it for outbound messages.

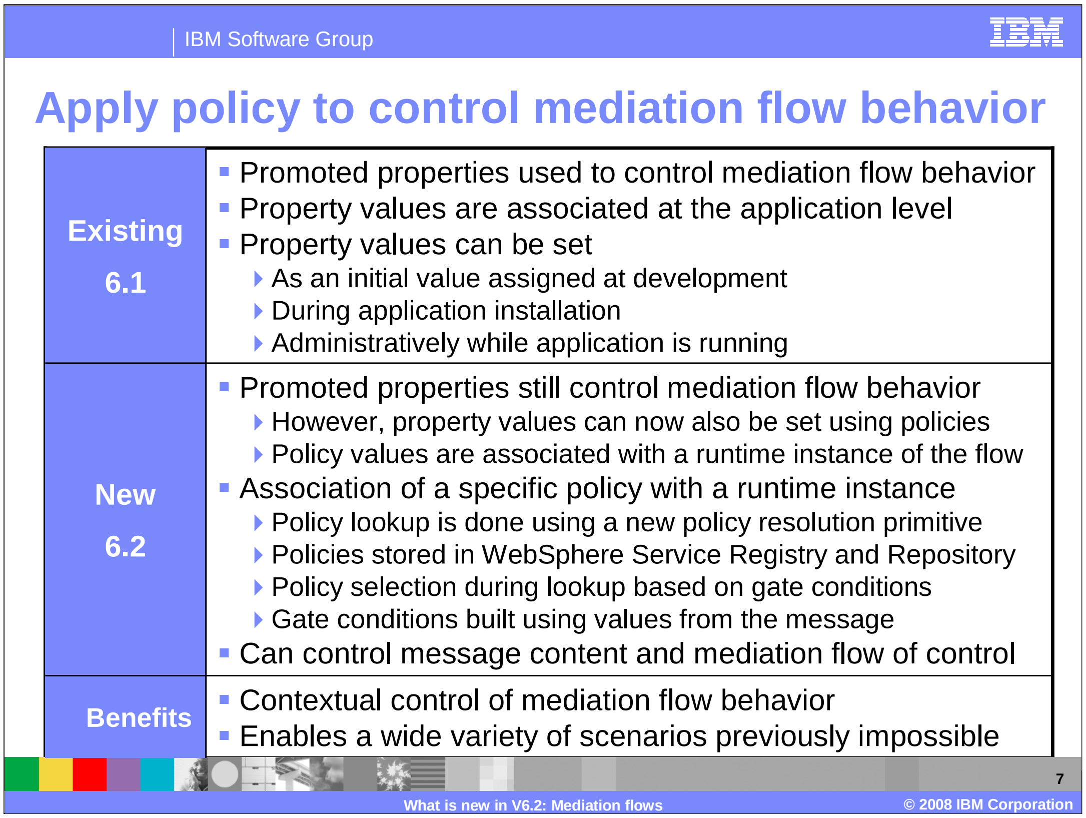
The service gateway function selectors are used to determine if an incoming message is a request response message or a one way message, so that the right operation can be invoked on the gateway. These function selectors are protocol dependent, so a different implementation exist for each of the binding types.

Service gateway data handlers are also protocol specific and provide the conversion between native data and business objects passed through the mediation. However, the business objects are really just wrappers of the native data, so there is no transformation of the native data performed.

The header setter primitives are used by gateway implementations that have to inspect or update the protocol specific headers of a message.

The data handler primitive is used in a flow where the native data, contained in a business object wrapper, actually has to be transformed. This primitive allows configuration of a data handler similar to how one normally is used by an import or export in a non-gateway scenario.

The type filter primitive is useful in making routing decisions within the flow. This can be important for a gateway scenario which had differing types of native data so that it can differentiate between the different business object wrappers used to contain the native data.



IBM Software Group

Apply policy to control mediation flow behavior

Existing 6.1

- Promoted properties used to control mediation flow behavior
- Property values are associated at the application level
- Property values can be set
  - As an initial value assigned at development
  - During application installation
  - Administratively while application is running

New 6.2

- Promoted properties still control mediation flow behavior
  - However, property values can now also be set using policies
  - Policy values are associated with a runtime instance of the flow
- Association of a specific policy with a runtime instance
  - Policy lookup is done using a new policy resolution primitive
  - Policies stored in WebSphere Service Registry and Repository
  - Policy selection during lookup based on gate conditions
  - Gate conditions built using values from the message
- Can control message content and mediation flow of control

Benefits

- Contextual control of mediation flow behavior
- Enables a wide variety of scenarios previously impossible

7

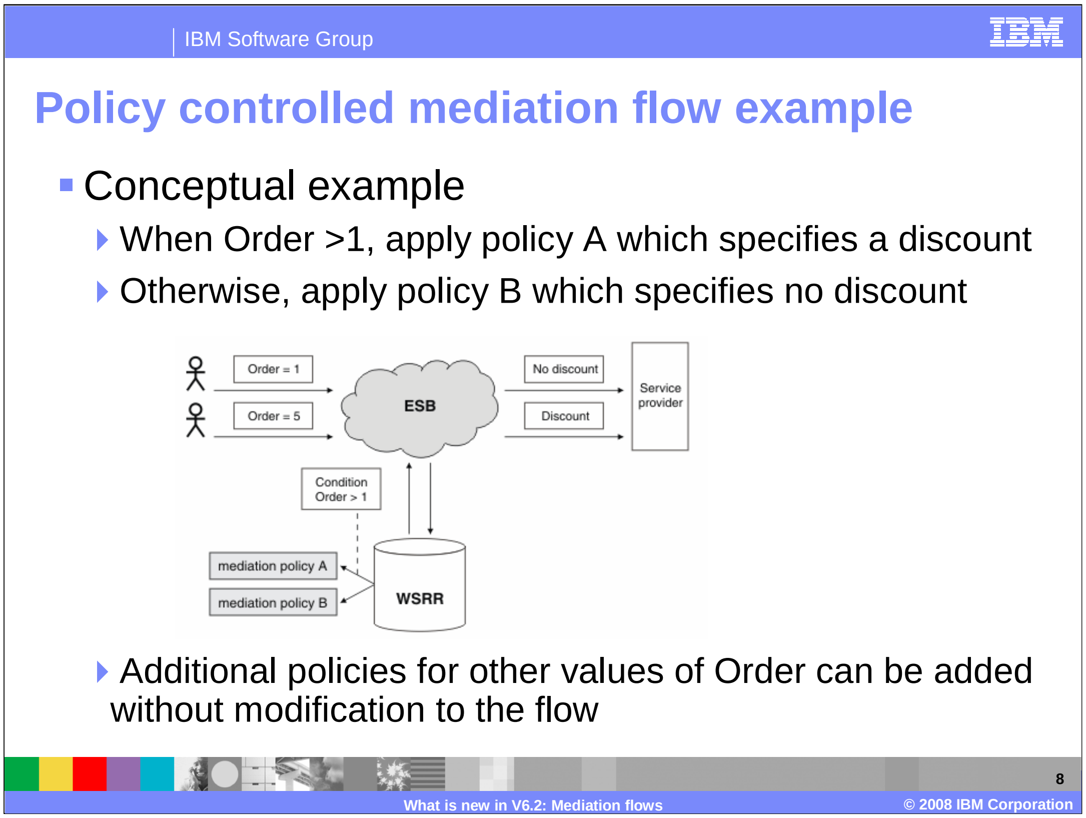
What is new in V6.2: Mediation flows

This enhancement provides the capability to control the behavior of runtime instances of mediation flows through the use of policies.

In version 6.1 promoted properties are used to control mediation flow behavior. The promoted properties are associated with the application and therefore affect the behavior for all runtime instances of the flow. The property values can be set during development, during application install or during administration of an installed application.

With the introduction of policy controlled mediation flows in version 6.2, promoted properties are still the mechanism used to control mediation flow behavior. However, the values for the promoted properties can now be set through the use of a property that is accessed from a registry. Additionally, the property values are associated with a runtime instance of a flow rather than for all instances of the flow. The policy is looked up from the WebSphere Service Registry and Repository using the new policy resolution primitive. The lookup is done using gate conditions based on values of elements within the message. The result is that message content and mediation flow of control can both be affected on a per runtime instance basis through the setting of promoted property values from policies which match gate conditions.

Among the benefits of this enhancement is the contextual control of mediation flow behavior. This enables many scenarios which were previously difficult or impossible to implement within a mediation flow.



Policy controlled mediation flow example

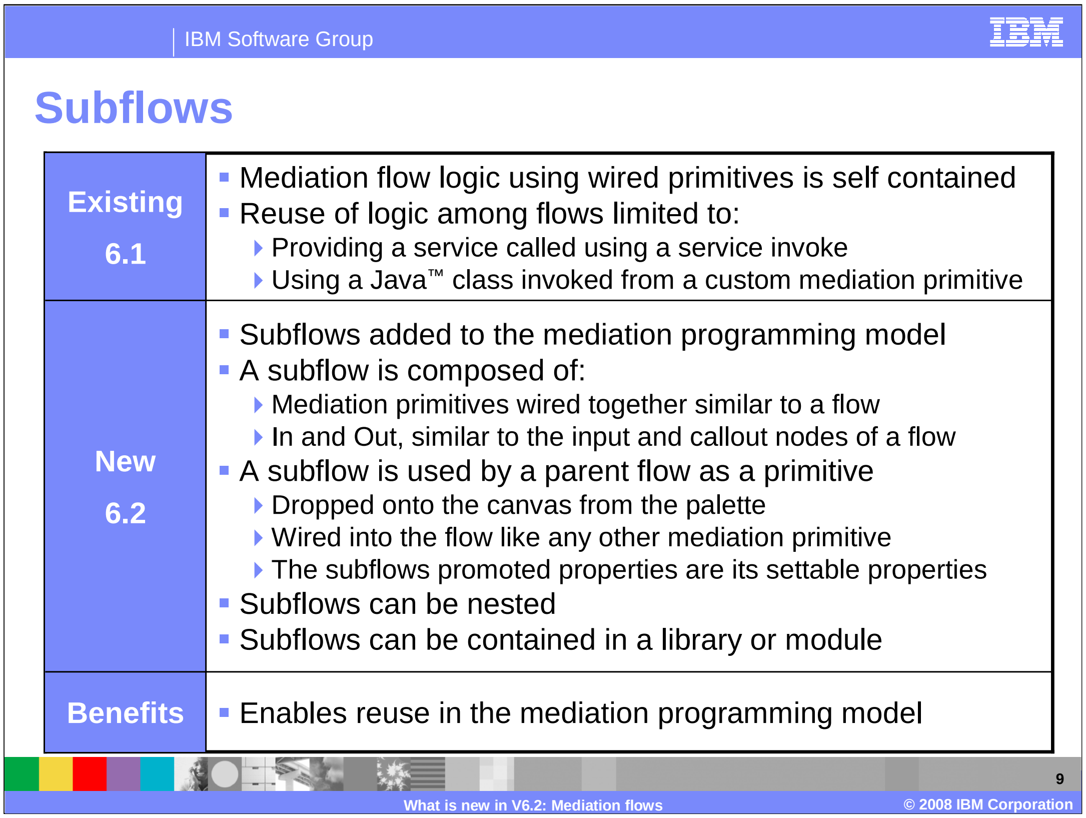
- Conceptual example
  - When Order >1, apply policy A which specifies a discount
  - Otherwise, apply policy B which specifies no discount

Order = 1
Order = 5
ESB
No discount
Discount
Service provider
Condition Order > 1
mediation policy A
mediation policy B
WSRR

  - Additional policies for other values of Order can be added without modification to the flow

This slide illustrates and example of a policy controlled mediation flow. As illustrated in the diagram, WebSphere Service Registry and Repository contains two policies, policy A which specifies a discount and policy B which does not specify a discount. Policy B is the default policy and policy A is associated with a gate condition that Order is greater than one. There are two requests shown, one with Order equal to one and another with Order equal to 5. The first request does not receive the discount and the second request does.

By adding additional policies to the registry for different values of Order, this behavior can be modified without changing the code.



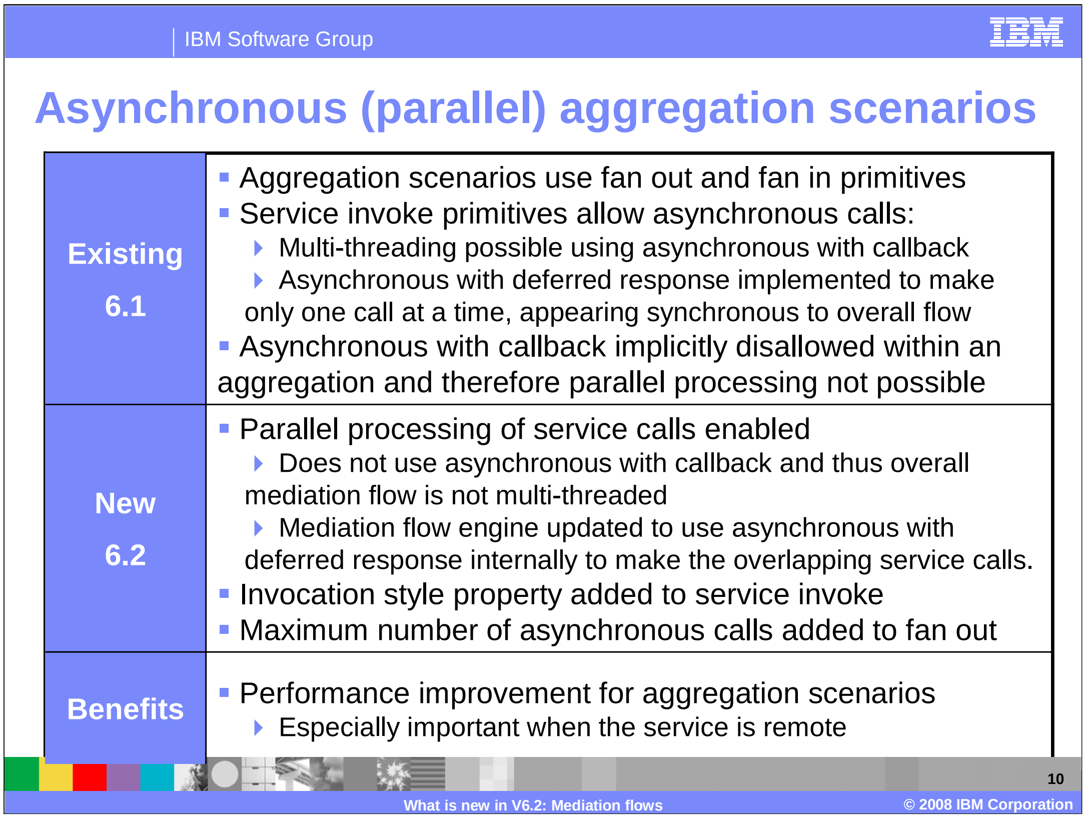
Subflows

Existing 6.1	▪ Mediation flow logic using wired primitives is self contained ▪ Reuse of logic among flows limited to: ‣ Providing a service called using a service invoke ‣ Using a Java™ class invoked from a custom mediation primitive
New 6.2	▪ Subflows added to the mediation programming model ▪ A subflow is composed of: ‣ Mediation primitives wired together similar to a flow ‣ In and Out, similar to the input and callout nodes of a flow ▪ A subflow is used by a parent flow as a primitive ‣ Dropped onto the canvas from the palette ‣ Wired into the flow like any other mediation primitive ‣ The subflows promoted properties are its settable properties ▪ Subflows can be nested ▪ Subflows can be contained in a library or module
Benefits	▪ Enables reuse in the mediation programming model

This enhancement provides for the use of subflows within a mediation. In version 6.1 the mediation flow logic defined by the wiring and configuration of primitives is self contained. In order to get reuse of logic between different flows you have two choices. You either abstract it into a service and call the service using a service invoke primitive, or develop a Java class and invoke it from a custom mediation primitive.

With the introduction of subflows in version 6.2 there is now an approach to reuse within the mediation flow programming model itself. A subflow is similar to a flow in that it is composed of mediation primitives that are configured and wired together to define the logic of the flow. Subflows have an in node and an out node, which are similar to the input and callout nodes of a flow. A subflow is used in a parent flow as any other primitive, being dropped onto the canvas, wired to other primitives and the properties configured. The way a subflow is configured is through the use of promoted properties which become the settable properties used by the parent flow.

Among the benefits of this enhancement is the introduction of reuse into the mediation flow programming model.



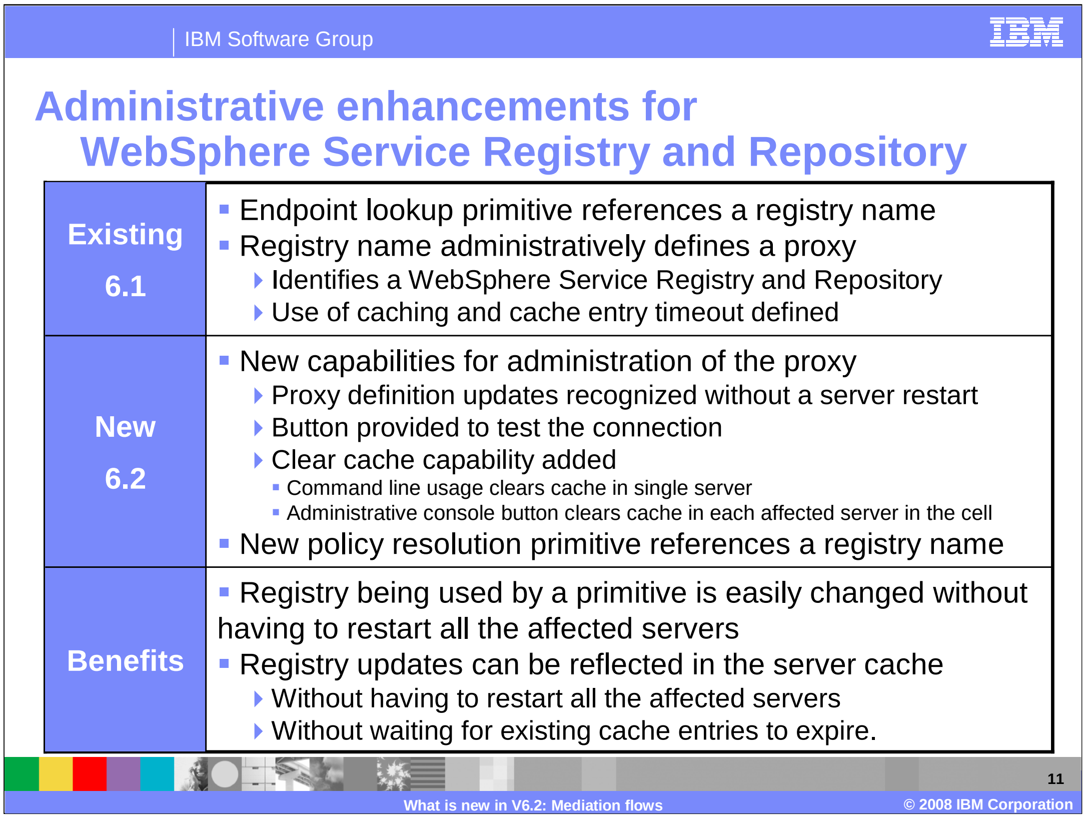
Asynchronous (parallel) aggregation scenarios

Existing 6.1	▪ Aggregation scenarios use fan out and fan in primitives ▪ Service invoke primitives allow asynchronous calls: ▸ Multi-threading possible using asynchronous with callback ▸ Asynchronous with deferred response implemented to make only one call at a time, appearing synchronous to overall flow ▪ Asynchronous with callback implicitly disallowed within an aggregation and therefore parallel processing not possible
New 6.2	▪ Parallel processing of service calls enabled ▸ Does not use asynchronous with callback and thus overall mediation flow is not multi-threaded ▸ Mediation flow engine updated to use asynchronous with deferred response internally to make the overlapping service calls. ▪ Invocation style property added to service invoke ▪ Maximum number of asynchronous calls added to fan out
Benefits	▪ Performance improvement for aggregation scenarios ▸ Especially important when the service is remote

This enhancement provides a performance improvement for mediation flows using service invoke within an aggregation. In version 6.1 the fan out and fan in primitives were introduced to enable aggregation scenarios and the service invoke primitive was introduced to allow service calls to be made from within a mediation flow. Under certain conditions, the asynchronous with callback invocation style is used with service invoke to enable multi-threaded mediation flows. However, within an aggregation using fan out and fan in, the asynchronous with callback style of invocation is implicitly turned off. Also, the asynchronous with deferred response style, although enabled, immediately waits for the response after the invocation, and therefore does not provide parallel processing of the service calls. The net result is that a performance improvement that might be obtained from parallel service calls is not possible within an aggregation.

In version 6.2 an enhancement is implemented that allows for the parallel processing of service calls from within a fan out and fan in aggregation flow. The implementation makes use of asynchronous with deferred response and does not enable the use of asynchronous with callback. The parallel processing is achieved in the mediation runtime engine by invoking multiple service calls asynchronously within the fan out and fan in aggregation. However, the overall mediation itself is completed on the same thread as it is started on and therefore does not appear multi-threaded. The parallel processing is configured using the newly added invocation style property of the service invoke primitive and the maximum number of asynchronous calls property of the fan out primitive.

The principal benefit of this enhancement is to provide a performance improvement for aggregation scenarios. This can be significant for an iterative aggregation with many elements when the service being called is remote.



Administrative enhancements for WebSphere Service Registry and Repository

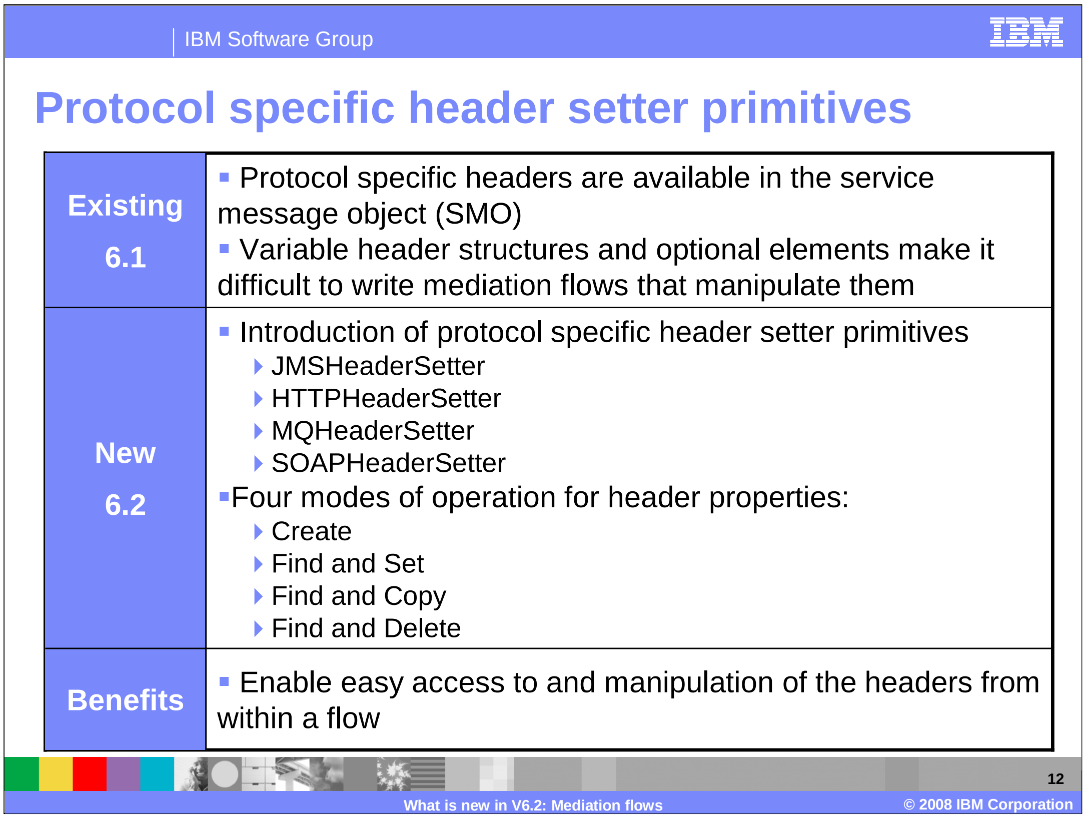
Existing **6.1**	▪ Endpoint lookup primitive references a registry name ▪ Registry name administratively defines a proxy ‣ Identifies a WebSphere Service Registry and Repository ‣ Use of caching and cache entry timeout defined
New **6.2**	▪ New capabilities for administration of the proxy ‣ Proxy definition updates recognized without a server restart ‣ Button provided to test the connection ‣ Clear cache capability added ▪ Command line usage clears cache in single server ▪ Administrative console button clears cache in each affected server in the cell ▪ New policy resolution primitive references a registry name
Benefits	▪ Registry being used by a primitive is easily changed without having to restart all the affected servers ▪ Registry updates can be reflected in the server cache ‣ Without having to restart all the affected servers ‣ Without waiting for existing cache entries to expire.

This enhancement provides improved capabilities for the WebSphere Service Registry and Repository proxy.

In version 6.1 the endpoint lookup primitive identifies the registry to use. This is done with a registry name that identifies an administrative proxy for a WebSphere Service Registry and Repository service. The proxy specifies the endpoint for the registry and provides a cache and a cache entry timeout value.

In version 6.2 this still works in the same way, but enhancements have been made to the proxy for improved administrative control. The first improvement is that updates to the configuration of the proxy are recognized by the server when the configuration is saved and a server restart is no longer required. The next improvement is the addition of a test connection button which can be used to verify that the endpoint configuration information is correct and can be used to connect to the registry. The third improvement is the capability to administratively clear the entire cache without having to restart the server. The proxy definition is cell wide. In a network deployment environment, clearing the cache from the administrative console will clear the cache for all servers in the cell. However, when using the command line to clear the cache, only the cache in the target server is cleared. Another enhancement to mention is not to the proxy itself, but rather the addition of the policy resolution primitive, adding another use of the proxy.

The benefit of this enhancement is improved administrative control of the proxy and its cache, eliminating the need for server restarts to reflect configuration changes or to clear the cache.



IBM Software Group

Protocol specific header setter primitives

Existing **6.1**	▪ Protocol specific headers are available in the service message object (SMO) ▪ Variable header structures and optional elements make it difficult to write mediation flows that manipulate them
New **6.2**	▪ Introduction of protocol specific header setter primitives ▸ JMSHeaderSetter ▸ HTTPHeaderSetter ▸ MQHeaderSetter ▸ SOAPHeaderSetter ▪Four modes of operation for header properties: ▸ Create ▸ Find and Set ▸ Find and Copy ▸ Find and Delete
Benefits	▪ Enable easy access to and manipulation of the headers from within a flow

12

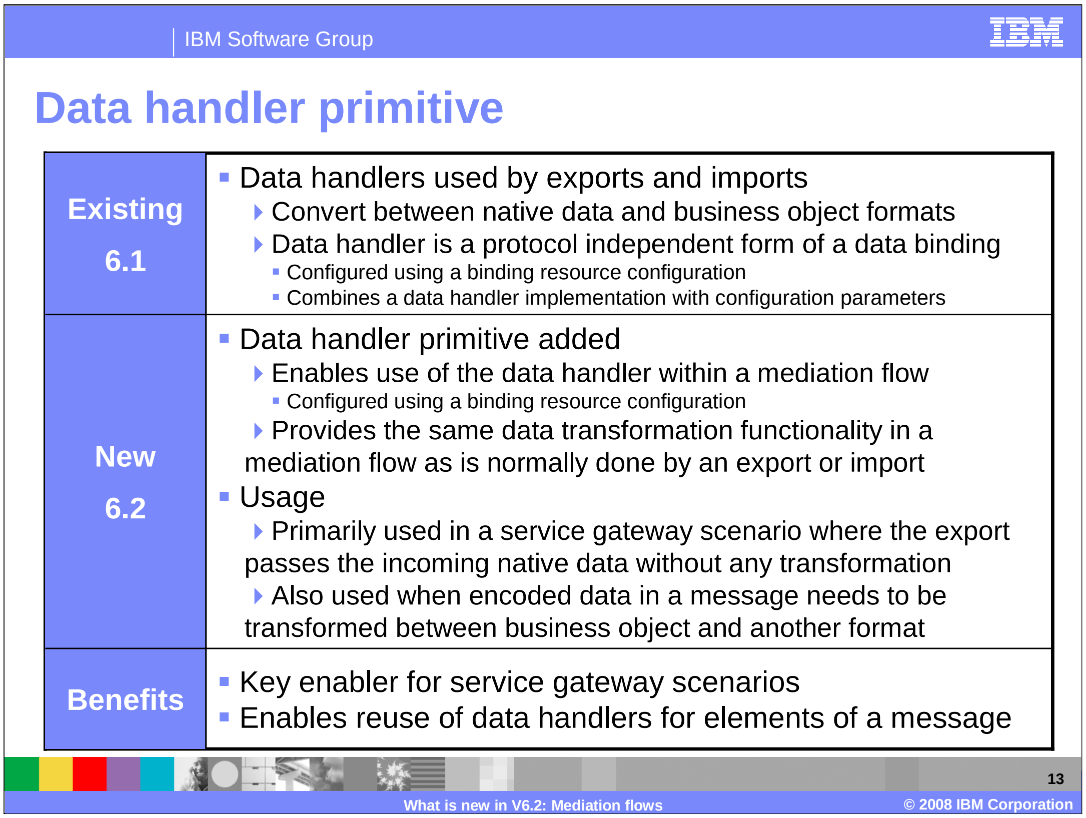
What is new in V6.2: Mediation flows

This enhancement provides four new primitives for the manipulation of protocol headers within the service message object.

In version 6.1 the protocol specific headers are available in the service message object. However, because of the structure of the headers and their use of optional elements it is often difficult to write a mediation flow that accesses and manipulates the headers.

In version 6.2 there are four new protocol specific header setter primitives added for JMS, HTTP, MQ and SOAP. These primitives allow you to access, update, create and delete header properties. The JMS and HTTP primitives are configured on an individual property basis and can be used for both the standard protocol properties and for user defined properties. The MQ and SOAP primitives are configured on a header type basis, where you define which properties within to manipulate.

The benefit of this enhancement is that protocol headers can now be easy manipulated from a mediation flow.



IBM Software Group

Data handler primitive

Existing 6.1	▪ Data handlers used by exports and imports ▸ Convert between native data and business object formats ▸ Data handler is a protocol independent form of a data binding ▪ Configured using a binding resource configuration ▪ Combines a data handler implementation with configuration parameters
New 6.2	▪ Data handler primitive added ▸ Enables use of the data handler within a mediation flow ▪ Configured using a binding resource configuration ▸ Provides the same data transformation functionality in a mediation flow as is normally done by an export or import ▪ Usage ▸ Primarily used in a service gateway scenario where the export passes the incoming native data without any transformation ▸ Also used when encoded data in a message needs to be transformed between business object and another format
Benefits	▪ Key enabler for service gateway scenarios ▪ Enables reuse of data handlers for elements of a message

13

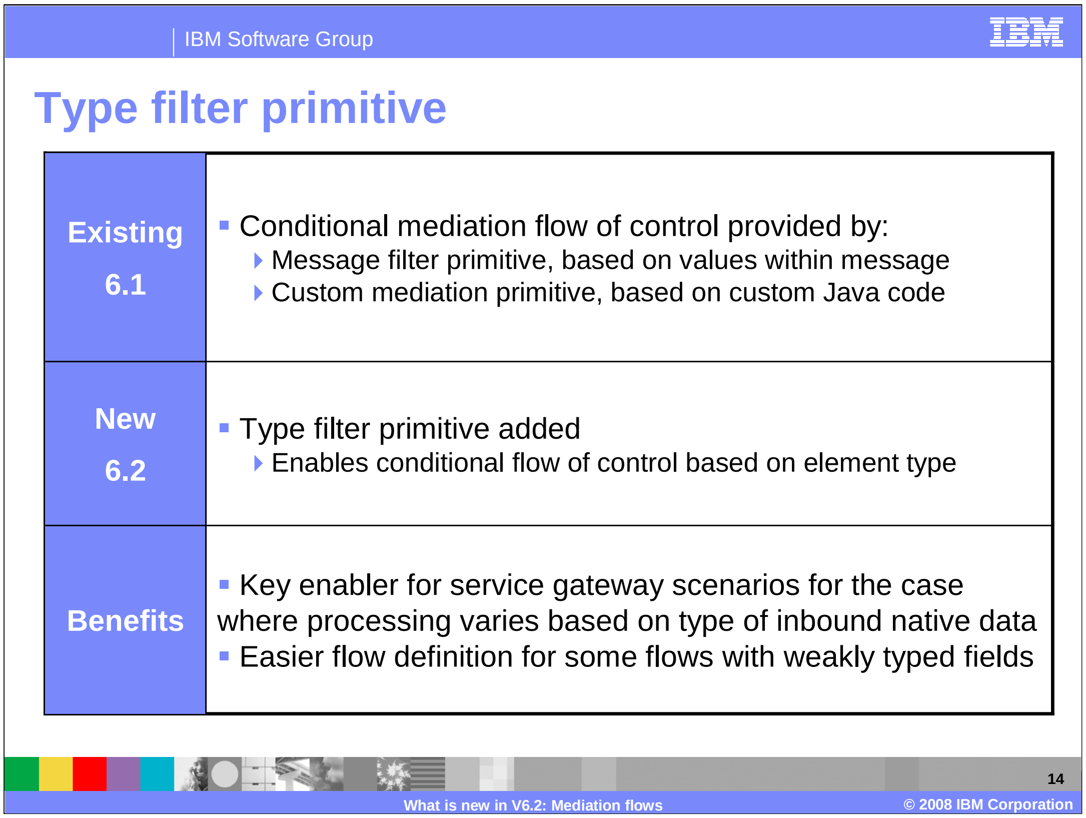
What is new in V6.2: Mediation flows

This enhancement provides a new primitive, the data handler primitive.

In version 6.1 data handlers are used with imports and exports to convert between native formats and business object formats. The data handler is a protocol independent form of a data binding which is configured using a resource configuration. The resource configuration combines a data handler implementation with configuration information required by the data handler.

With the introduction of the data handler primitive in version 6.2, the capability to do conversions between native formats and business object formats is now provided from within a mediation flow. This provides similar functionality to exports and imports. The primary use of this is for service gateway scenarios where the native data is wrapped by the export and passed into the mediation. There are also other possible uses when a message within a mediation contains elements that need to be converted between native and business object formats.

This enhancement is one of the key enablers to the service gateway scenario. It also enables reuse of data handlers for conversion of elements within a message.



IBM Software Group

Type filter primitive

Existing 6.1	▪ Conditional mediation flow of control provided by: ▸ Message filter primitive, based on values within message ▸ Custom mediation primitive, based on custom Java code
New 6.2	▪ Type filter primitive added ▸ Enables conditional flow of control based on element type
Benefits	▪ Key enabler for service gateway scenarios for the case where processing varies based on type of inbound native data ▪ Easier flow definition for some flows with weakly typed fields

14

What is new in V6.2: Mediation flows

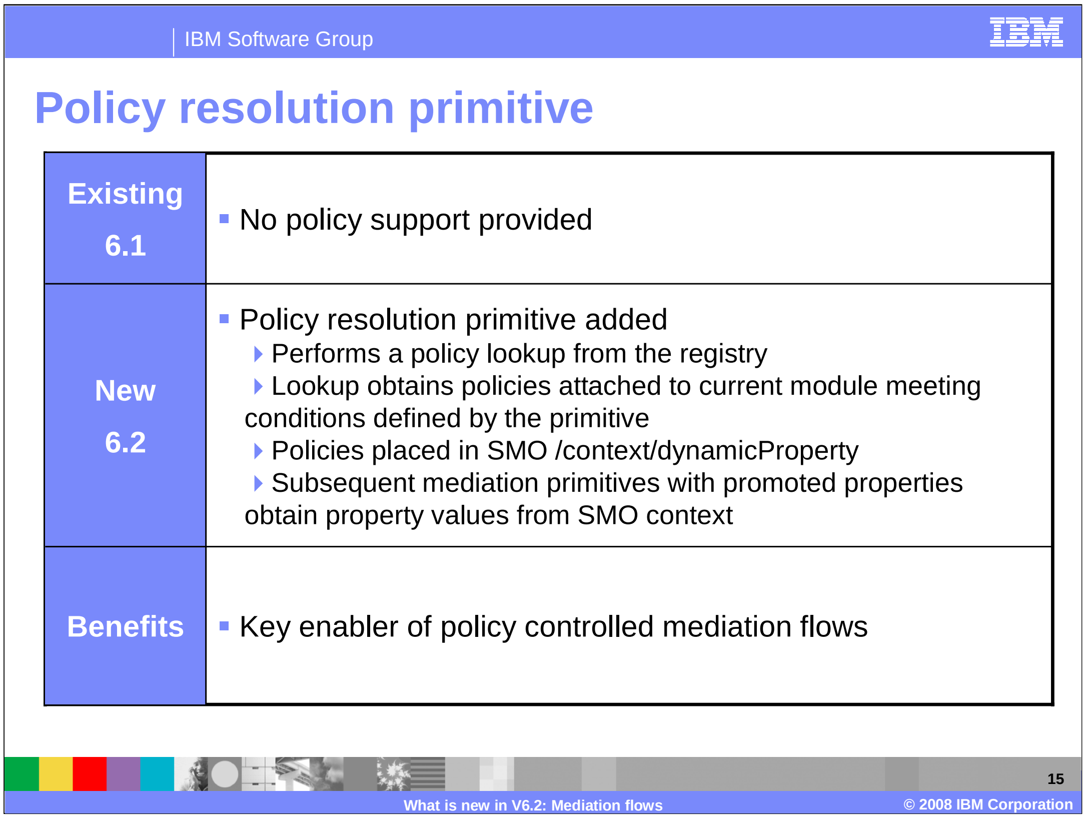
© 2008 IBM Corporation

This enhancement provides a new primitive called the type filter.

In version 6.1 you can make flow of control decisions for a mediation using a message filter primitive based on values of elements within the message. To manage flow of control based on other factors, such as the type of an element, you need to code a custom mediation primitive using Java code.

With the introduction of the type filter primitive in version 6.2 you can now make flow of control decisions for a mediation based on the type of an element within the message. This primitive is configured in a similar way to the message filter primitive.

Among the benefits of this enhancement is that it is a key enabler for the service gateway scenario where processing needs to vary based on the type of the inbound native data. Also, it can be used for any flow definition which must make decisions based on contents of a weakly typed field.



Policy resolution primitive

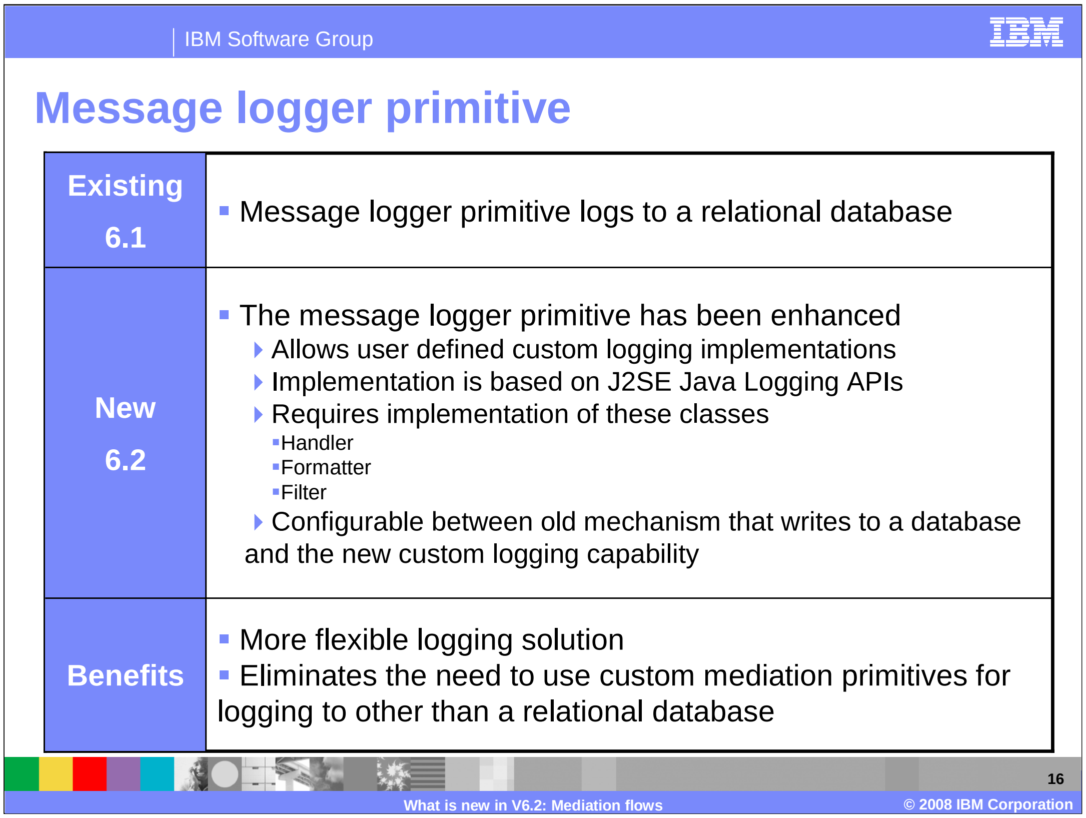
Existing **6.1**	▪ No policy support provided
New **6.2**	▪ Policy resolution primitive added ▸ Performs a policy lookup from the registry ▸ Lookup obtains policies attached to current module meeting conditions defined by the primitive ▸ Policies placed in SMO /context/dynamicProperty ▸ Subsequent mediation primitives with promoted properties obtain property values from SMO context
Benefits	▪ Key enabler of policy controlled mediation flows

This enhancement provides a new policy resolution primitive, which is a key part of the policy controlled mediation flow enhancement described earlier in this presentation.

In version 6.1 there was no policy support provided for mediation flows.

In version 6.2 the policy resolution primitive has been added. It is used to perform the lookup of polices from a WebSphere Service Registry and Repository. The lookup obtains policies which are attached to the current module and satisfy some specified gate conditions. The gate conditions are conditional expressions based on values of elements within the message. The policies are then placed into the service message object in a new context for dynamic properties. Subsequent mediation primitives in the flow which are configured with promoted properties obtain the values from those properties from the dynamic properties context.

The benefit of this enhancement is that it is a major part of the ability to have policy controlled mediation flows.



IBM Software Group

Message logger primitive

Existing 6.1	▪ Message logger primitive logs to a relational database
New 6.2	▪ The message logger primitive has been enhanced ▸ Allows user defined custom logging implementations ▸ Implementation is based on J2SE Java Logging APIs ▸ Requires implementation of these classes ▪Handler ▪Formatter ▪Filter ▸ Configurable between old mechanism that writes to a database and the new custom logging capability
Benefits	▪ More flexible logging solution ▪ Eliminates the need to use custom mediation primitives for logging to other than a relational database

16

What is new in V6.2: Mediation flows

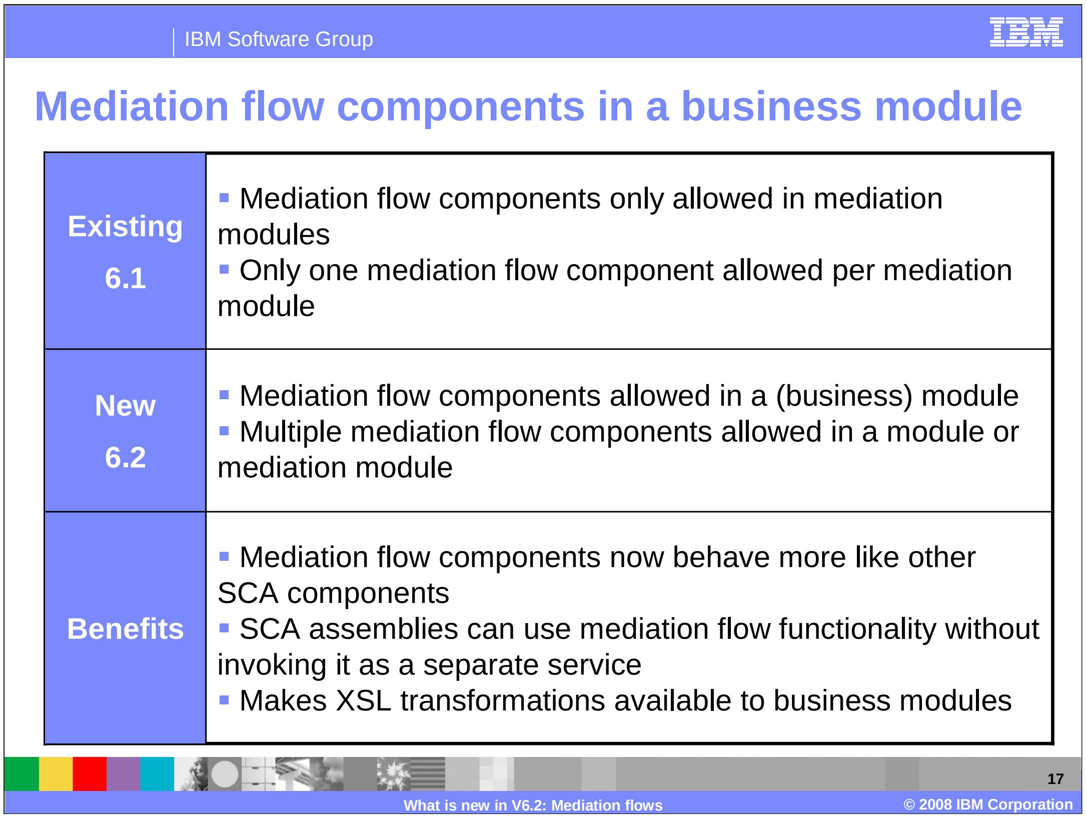
© 2008 IBM Corporation

This enhancement provides new capabilities to the message logger primitive.

In version 6.1 the message logger wrote to a relational database.

In version 6.2 the message logger can now optionally be configured to use a custom logging implementation rather than using the relational database. This allows you to implement your own logging solution and make use of it from a primitive in the mediation flow. The implementation is based on the J2SE Java logging APIs which make use of a handler, formatter, and filter.

Among the benefits of this enhancement is that the logging capabilities are now much more flexible. There is no longer a need to use custom mediation primitives when the logging requirement is for something other than logging to a relational database.



Mediation flow components in a business module

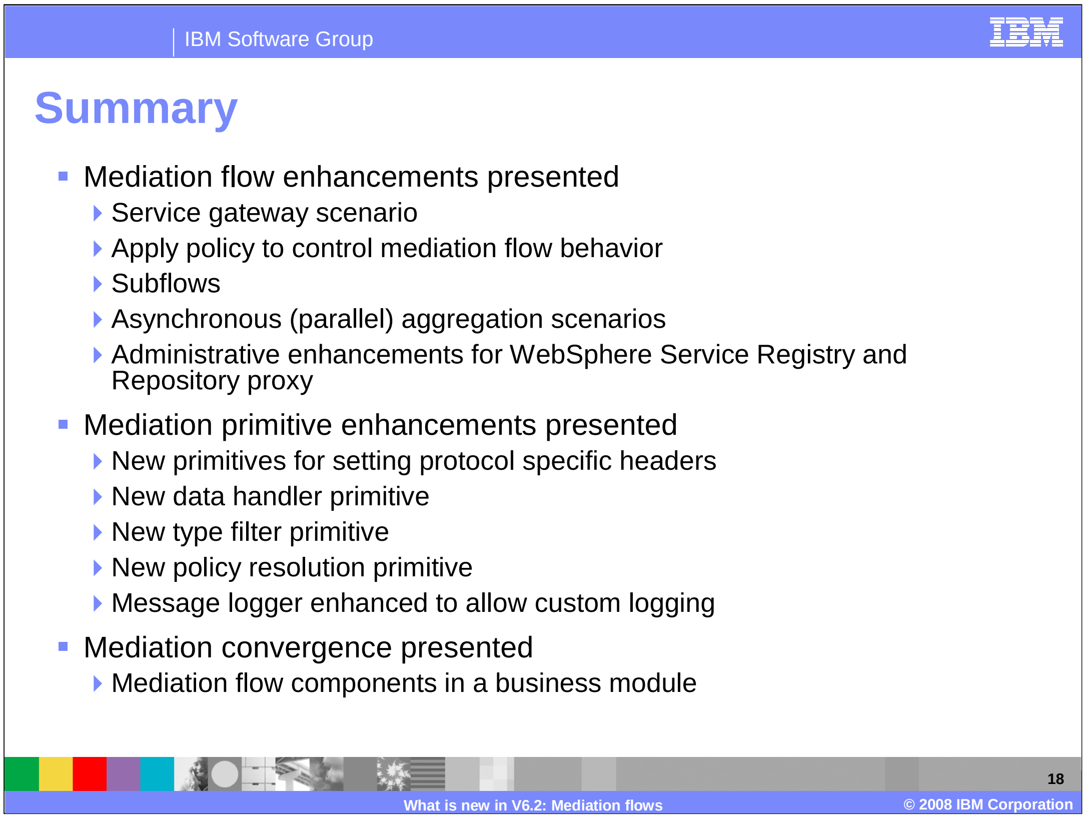
Existing 6.1	▪ Mediation flow components only allowed in mediation modules ▪ Only one mediation flow component allowed per mediation module
New 6.2	▪ Mediation flow components allowed in a (business) module ▪ Multiple mediation flow components allowed in a module or mediation module
Benefits	▪ Mediation flow components now behave more like other SCA components ▪ SCA assemblies can use mediation flow functionality without invoking it as a separate service ▪ Makes XSL transformations available to business modules

This enhancement makes using a mediation flow component more like that of any other SCA component.

In version 6.1 a mediation flow component is restricted to use only within a mediation module. Also, there can only be one instance of a mediation flow component per mediation module.

In version 6.2 a mediation flow component can be used within a module, sometimes referred to as a business module. Furthermore, more than one mediation flow component can be used in either a module or mediation module.

Among the benefits of this enhancement is that mediation flow components now behave more like other SCA components. SCA assemblies can use mediation functionality without having to invoke it as a separate service. One of the uses for this is to enable XSL transformations to be used within a business module.



IBM Software Group

Summary

- Mediation flow enhancements presented
  - Service gateway scenario
  - Apply policy to control mediation flow behavior
  - Subflows
  - Asynchronous (parallel) aggregation scenarios
  - Administrative enhancements for WebSphere Service Registry and Repository proxy
- Mediation primitive enhancements presented
  - New primitives for setting protocol specific headers
  - New data handler primitive
  - New type filter primitive
  - New policy resolution primitive
  - Message logger enhanced to allow custom logging
- Mediation convergence presented
  - Mediation flow components in a business module

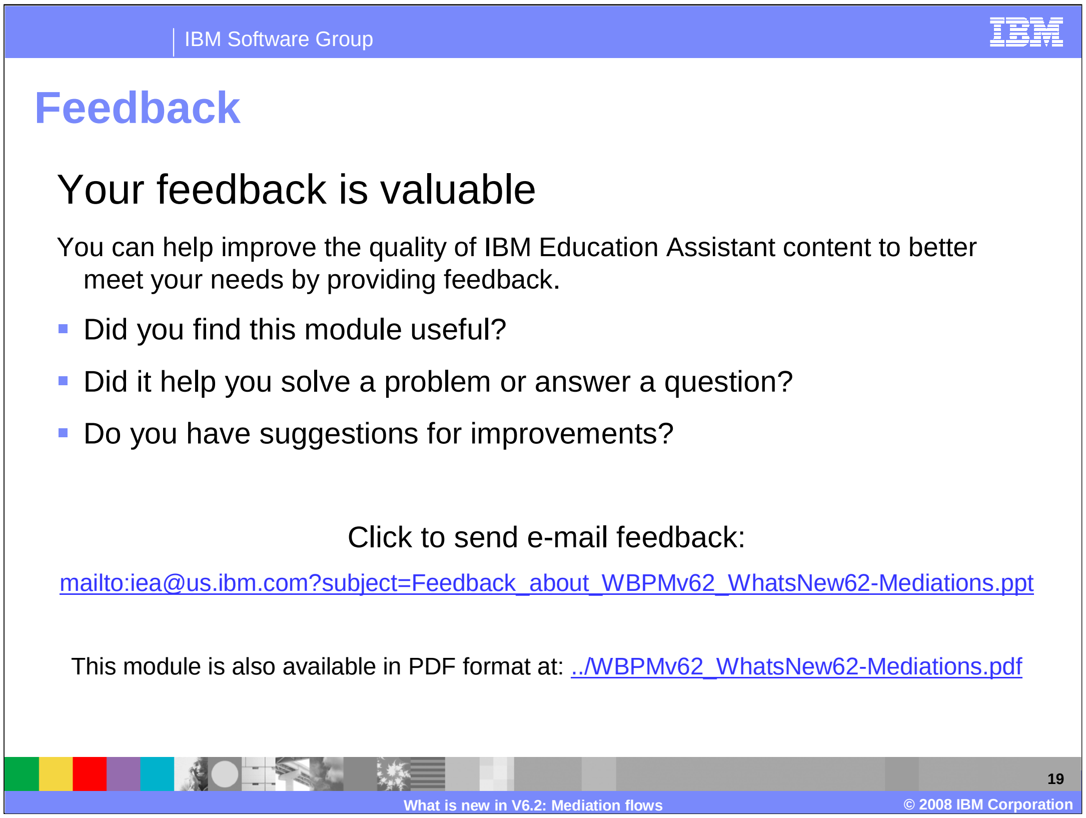
18

What is new in V6.2: Mediation flows

In this presentation you were introduced to the new features for version 6.2. The first group includes those features that have to do with mediation flows. The service gateway scenario enables mediation flows that provide common processing across multiple interfaces and operations. The ability to use policies to control mediation flow behavior on a per-flow instance basis was described. The introduction of subflows enables reuse within the mediation programming model. Aggregation scenarios enabled by the fan out and fan in primitives are now able to make use of asynchronous parallel processing for service invocations within the flow. The administrative proxy used to interface with the WebSphere Service Registry and Repository had been enhanced to make administration of the registry easier.

The next group of enhancements was for new and updated mediation primitives. There are four new primitives specifically designed for use accessing and updating headers in the system management object. There is a data handler primitive which you configure to do conversions between native data formats and business object formats. The new type filter primitive allows you to make routing decisions based on element types within the service message object. The policy resolution primitive interacts with a repository to look up policies that control mediation behavior. The existing message logger primitive has been enhanced to allow for custom logging implementations to be used in addition to the existing log-to-a-database functionality.

The last topic addressed was mediation convergence, which involves enabling mediation flow components to be used like any other SCA component.



Feedback

Your feedback is valuable

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

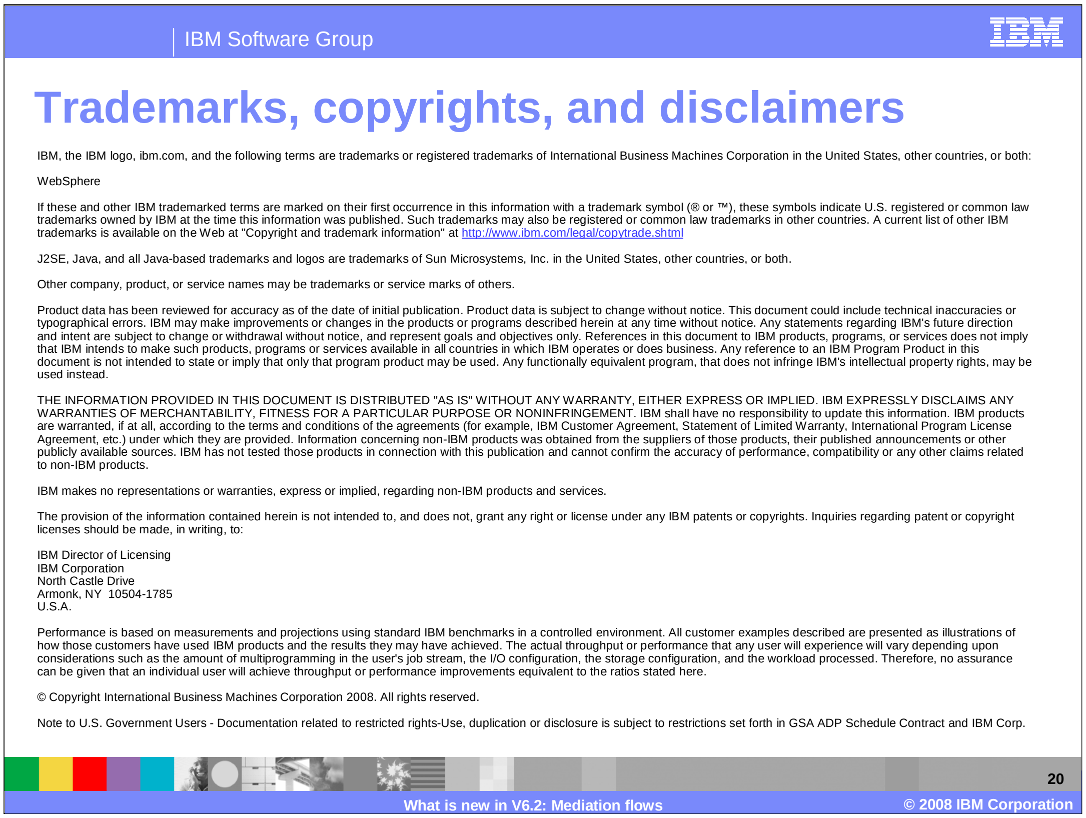
- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send e-mail feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_WBPMv62_WhatsNew62-Mediations.ppt

This module is also available in PDF format at: ../WBPMv62_WhatsNew62-Mediations.pdf

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



Trademarks, copyrights, and disclaimers

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

WebSphere

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

J2SE, Java, and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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

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

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

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

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

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

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

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