



IBM Software Group

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

Custom Mediation Primitive



@business on demand.

© 2006 IBM Corporation
Updated May 1, 2006

This presentation provides a detailed look at the Custom Mediation primitive.

Goals

- Understand the Custom Mediation primitive details



Custom Mediation

- ▶ Overview of function
- ▶ Use of terminals
- ▶ Definition of properties
- ▶ Structure of primitive and mediation module assembly
- ▶ Error handling
- ▶ Examine custom code samples

The goal of this presentation is to provide you with a full understanding of the Custom Mediation primitive. It is assumed that you are already familiar with the material presented in the **Mediation Primitive Common Details** presentation, which serves as a base for understanding mediation primitives in general. In this presentation, an overview of the Custom Mediation primitive is provided along with information about the primitive's use of terminals and its properties. The structure of the Custom Mediation primitive in relation to the mediation module assembly is then described. Finally, the error handling characteristics are presented and example usages of a Custom Mediation primitive are provided.

Custom Mediation – Overview of Function

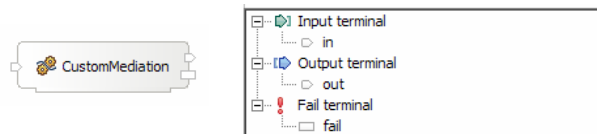
- Enables use of custom mediation logic
- Can be used when no built-in primitive provides a needed function
- Logic implemented in Java™ and defined as either:
 - ▶ Snippet (Visual or Java)
 - ▶ Service Component Architecture (SCA) Java component
- Similar to other mediation primitives in behavior
 - ▶ Usage of terminals, wiring, exception processing



The Custom Mediation primitive enable you to define your own custom mediation logic for use when the built-in primitives do not provide the needed functionality. Java is used to define the logic and can be coded as a visual snippet, a Java snippet or alternatively as an operation within a Java SCA component. The general behavior is similar to that of the built-in mediation primitives with respect to the use of terminals, wiring of primitives and exception handling.

Custom Mediation – Terminals

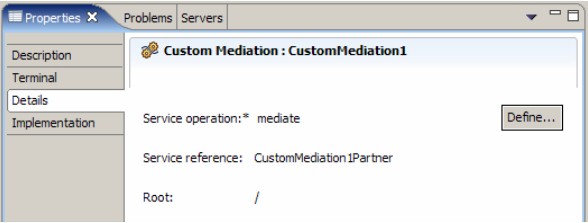
- Terminals:
 - ▶ Input terminal
 - ▶ One Output terminal
 - ▶ Fail terminal
- Message type of Input and Output terminal
 - Same type – for manipulation of values within a message
 - Different type – for changing format of the message body



The Custom Mediation primitive has one input terminal, one output terminal and a fail terminal. The output terminal can be for the same message type as the input terminal or for a different message type. When the message types are different, the Java code in the Custom Mediation must modify the structure of the body of the message to conform with the output terminal type. Shown here is a Custom Mediation primitive with its terminals and the terminals as seen in the properties view.

IBM Software Group IBM

Custom Mediation - Properties



- Service operation
 - ▶ Name of the operation to call on the service interface
 - ▶ Operation must:
 - Be a request/response operation
 - Take a single input parameter of type DataObject
 - Return a DataObject
- Service reference
 - ▶ Name of the Mediation Flow Component's reference to the service
 - ▶ The reference needs to be wired to a Java service component or an Import
- Root
 - ▶ The portion of the Service Message Object to be passed
 - ▶ Valid values are only: / and /body
- NOTE: Properties are read only on this panel – they are defined using the Define Custom Mediation dialog accessed using the Define... button

Custom Mediation Primitive © 2006 IBM Corporation 5

In the upper right is a screen capture of the Details tab from the Properties view for a Custom Mediation primitive showing the following properties:

Service operation is a property that contains the name of the operation to be called on the service interface. The operation must be a request/response operation that takes a DataObject as its only input parameter and it must also return a DataObject.

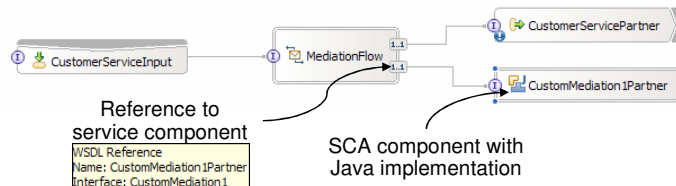
Service reference contains the name of a reference defined on the Mediation Flow Component that supports the service interface. In the assembly diagram, the reference must be wired to a Java SCA component or an SCA import.

Root is a property that defines the portion of the SMO that will be passed to the operation. There are only two valid values for root, / (slash), which indicates to pass the entire SMO and /body, which indicates to pass only the message body or payload.

The properties on this panel are read only and are set using the Define Custom Mediation dialog that is accessed by using the **Define...** button.

Custom Mediation – Structure

- Assembly diagram contains the following:
 - ▶ Reference on the Mediation Flow component
 - ▶ SCA component with Java implementation



- The reference and SCA component can be generated
- The custom logic can be coded in two ways:
 - ▶ In the Custom Mediation Properties view - Implementation tab
 - As a Visual Snippet or as a Java Snippet
 - Code to call the snippet is contained in the Java SCA component implementation
 - ▶ Directly in the Java SCA component implementation

The Custom Mediation primitive is the only mediation primitive that has a corresponding part, which is on the assembly diagram for the Mediation Module. The Custom Mediation primitive must have a reference on the Mediation Flow Component that is wired to a Java SCA component as shown on this slide. The development tools provide the capability for the reference and the Java SCA component to be generated after the Custom Mediation primitive properties have been defined. There are two approaches to defining the custom logic. The first is by coding either a visual or a Java snippet on the Implementation tab of the Properties view of the Custom Mediation. The Java SCA component contains generated code that invokes the snippet. The other approach is to write the custom code as part of the Java SCA component. In this case, the reference and the Java SCA component can still be generated, and will contain the stub of the operation to be implemented.

Custom Mediation – Structure

Code in Java Snippet

Generated Java SCA component implementation that calls the snippet

Java Snippet

```
public DataObject mediate(DataObject a_type) {
    // To override the generated Java Snippet, please comment out the following method call
    sca.component.mediation.java.impl.CustomMediation1PartnerCustomLogic.JavaSnippet snippet =
        new sca.component.mediation.java.impl.CustomMediation1PartnerCustomLogic.JavaSnippet();
    return snippet.execute(a_type);
}
```

Code in Java SCA Component

```
public DataObject mediate(DataObject a_type) {
    DataObject customerInfo =
        (DataObject)a_type.get("getCustomerInformation");
    String customerID =
        (String)customerInfo.get("customerID");
    System.out.println("$$$$ Customer ID =" + customerID);
    return a_type;
}
```

User Written Code

The screenshot shows an IDE window titled 'Custom Mediation: CustomMediation'. The Properties view on the left shows the 'Implementation' tab. The main editor displays the following code:

```
method signature: DataObject execute(DataObject a_type)
[Visual] [Java] [Open Java Editor...]
DataObject customerInfo =
    (DataObject)a_type.get("getCustomerInformation");
String customerID =
    (String)customerInfo.get("customerID");
System.out.println("$$$$ Customer ID =" + customerID);
return a_type;
```

Below the IDE window, the generated Java SCA component code is shown:

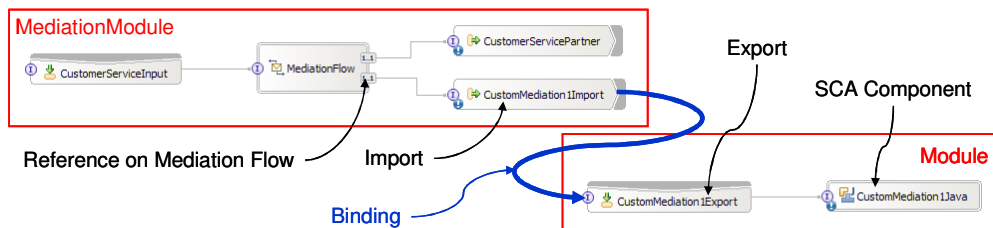
```
public DataObject mediate(DataObject a_type) {
    // To override the generated Java Snippet, please comment out the following method call
    sca.component.mediation.java.impl.CustomMediation1PartnerCustomLogic.JavaSnippet snippet =
        new sca.component.mediation.java.impl.CustomMediation1PartnerCustomLogic.JavaSnippet();
    return snippet.execute(a_type);
}
```

Red boxes and arrows indicate that the user-written code (the implementation of the mediate method) is identical in both the snippet and the component implementation.

This slide further examines the structure of a Custom Mediation. The top portion of the slide illustrates using a Java snippet, whereas the bottom portion shows placing the custom code directly into the Java SCA component. In the example, there is identical user written code. In the case of the Java snippet, the user code is contained on the Implementation panel of the Properties view and the generated code for calling the snippet is contained in the Java SCA component. In the case of the user code being implemented in the Java SCA component, it is inserted within the generated operation stub.

Custom Mediation – Alternate Structure

- Normal structure contains:
 - ▶ Reference on Mediation Flow Component
 - ▶ Java SCA Component
 - ▶ Custom code in Snippet or Java SCA Component
- An alternate structural possibility contains:
 - ▶ Reference on Mediation Flow Component
 - ▶ SCA Import with binding to an Export in another Module
 - ▶ Module with Export and SCA Component with the custom code



- Possible usage
 - ▶ Package complex custom mediation logic as a service
 - ▶ Use that service from many different mediation modules

This slide shows an alternative structure for a Custom Mediation primitive. Reviewing the typical structure, a Custom Mediation primitive is composed of a reference on the mediation flow component, a Java SCA component and custom code, which is either in a snippet or the Java SCA component. In the alternative structure there is still a reference on the mediation flow component. However, that reference is wired to an SCA import rather than a Java SCA component. In a separate SCA module, there is an export for the same service interface wired to an SCA component, which contains the custom code.

An example of a scenario where this structure might be useful is a case where there is complex custom mediation logic that must be used from several different mediation flows. The custom mediation is essentially packaged as a service and the service is used from each of the mediation flows.

Custom Mediation – Error Processing

- **MediationRuntimeException** thrown for
 - ▶ No operation or service reference specified in the properties
 - ▶ No matching reference exists on the Mediation Flow Component
- **MediationRuntimeException (Fail terminal flow)**
 - ▶ Note difference in behavior
 - Unlike most **MediationRuntimeExceptions**, these fire the Fail terminal if wired
 - ▶ The reference on the Mediation Flow Component is not wired
 - ▶ Custom code returns null
 - Must return a **DataObject**
 - Beware, default code is generated with **returns null;**
- **Any exception** throw by the custom java code
 - ▶ Thrown as is, not wrapped by any **MediationXxxxxException** type
 - ▶ If the Fail terminal is wired, it will be fired



The error processing details and considerations are examined in this slide.

A **MediationRuntimeException** will be thrown when the operation or service reference name has not been specified in the properties or if the specified reference does not exist on the Mediation Flow Component. A **MediationRuntimeException** can also occur for other conditions that are detected when processing the Custom Mediation primitive. Unlike most other **MediationRuntimeExceptions**, these will result in the Fail terminal flow being followed if the Fail terminal is wired. One of the conditions where this can occur is when the reference on the Mediation Flow Component is not wired. Another cause of this is when the custom code returns a null rather than a **DataObject**. This could be a common mistake because the generated code stub returns null and must therefore be changed to return a **DataObject**. The custom code can throw any exception and the exception thrown will not be wrapped by any of the mediation receptions. If the Fail terminal is wired, that flow will be followed. Otherwise, the exception is thrown and the mediation flow is terminated.

Custom Mediation – Custom Code Example

- A typical scenario
 - ▶ Custom Mediation used to construct a key prior to a Database Lookup
 - ▶ Data from the body of the message is manipulated to construct key
 - ▶ Key value is placed into the transient context
- Scenario for this code example:
 - ▶ Database Lookup will need first two digits of the customer ID as a key

```

public DataObject mediate(DataObject input1) {
    // Get the Service Message Object
    ServiceMessageObject smo = (ServiceMessageObject)input1;
    // Get the transient context
    ContextType context = smo.getContext();
    DataObject transientContext = (DataObject)context.getTransient();
    // Get the customerID from the body
    DataObject body = (DataObject)smo.getBody();
    DataObject customerInfo = (DataObject)body.get("getCustomerInformation");
    String customerID = (String)customerInfo.get("customerID");
    // Move the prefix of the customerID to the transient context
    transientContext.setString("accountLocationID", customerID.substring(0,2));
    // Return the data object
    return (DataObject)smo;
}

```

10

Custom Mediation Primitive

© 2006 IBM Corporation

One of the typical scenarios for a custom mediation is constructing a key that can be used to perform a database lookup. In such a scenario, the custom mediation can access various data elements in the SMO and use them to construct a key value that is then placed into the transient context to be used by a subsequent database lookup. In this particular scenario, the first two digits of a customer number are extracted to be used as a key. Looking at the code, you will see that the mediate operation takes a DataObject as input and returns a DataObject. In this example, the root property was specified as / (slash), indicating that the DataObject passed in and returned is the entire SMO. The first step is to cast the input parameter **input1** to a ServiceMessageObject type and assign it to a ServiceMessageObject variable called **smo**. This variable can then be used to perform type safe SMO specific operations. The next step is to obtain the transient context, which is done in two steps. The **getContext** operation is performed on **smo**, which returns the context portion of the SMO and is assigned to a ContextType variable called **context**. Next, the **getTransient** operation is performed on **context** to obtain the transient context, which is assigned to a DataObject type called **transientContext**. The **customerID** must now be obtained from the body of the SMO. Using the **smo** variable, the **getBody** operation returns the body of the message, which is assigned to a DataObject called **body**. Using the generic **get** operation on **body** the **getCustomerInformation** property is obtained and similarly the **customerID** is then obtained from it. A substring of the first two digits of **customerID** is obtained and placed into the **accountLocationID** element of the **transientContext** using the generic **setString** operation. Finally the original SMO is returned with the transient context updated to contain the desired key value.

Custom Mediation – Throwing an Exception

Scenario

- ▶ Check customer ID for validity
- ▶ Throw MediationBusinessException when invalid
- ▶ Exception contains user defined message insert

```
// Method signature has "throws" clause added
public DataObject mediate(DataObject input1) throws MediationBusinessException {
    // Get the customer ID
    DataObject body = (DataObject) input1.get("body");
    DataObject customerInfo = (DataObject)body.get("getCustomerInformation");
    String customerID = (String)customerInfo.get("customerID");
    // Perform whatever logic is needed to validate the customerID
    if (customerID.length() < 6) {
        // Throw exception if invalid
        String msg = "MyOwnErrorCode: CustomerID is too short, ID=" + customerID;
        throw new MediationBusinessException(msg);
    }
    // CustomerID is OK, return
    return input1;
}
```

Resulting Log

```
[1/18/06 17:48:50:354 CST] 0000004b ExceptionUtil E CNTR0020E: EJB threw an unexpected (non-declared) exception during invocation of method
"transactionNotSupportedActivitySessionSupports" on bean "BeanId(CustomerRoutingMediationApp#CustomerRoutingMediationEJB.jar#Module, null)".
Exception data: com.ibm.wsspi.sibx.mediation.MediationBusinessException: MyOwnErrorCode: CustomerID is too short, ID=22222
at sca.component.java.impl.CustomMediation1PartnerImpl.mediate(CustomMediation1PartnerImpl.java:49)
```

Exception Type Code where error occurred User defined message



The code example on this slide examines custom code that throws a `MediationBusinessException` containing a user defined message insert if the `customerID` does not pass a validity check.

The operation has the same signature as the previous example, except that it now has a `throws` clause to declare that the operation can throw a `MediationBusinessException`. In the code, the `customerID` is obtained from the SMO by drilling down to it using the generic `DataObject` `get` operation. First the `body` is obtained from `input1`, which contains the SMO. Then `getCustomerInformation` is obtained from the `body` and finally the `customerID` is obtained from the `getCustomerInformation`. Validation checking is then done on the `customerID`, which in the case of this example is simply checking to make sure its length is not less than six digits. If it is less than six digits the `MediationBusinessException` containing the user defined error message is thrown. Assuming that the Fail terminal of the Custom Mediation primitive has not been wired, the mediation flow will be terminated and a log message written. An example of such a log is shown at the bottom of the slide. Notice that the log contains the exception type and the user defined error message, and also identifies the custom mediation code as the source of the exception.

Summary

- Examined the Custom Mediation primitive details



Custom Mediation

- ▶ Overview of function
- ▶ Use of terminals
- ▶ Definition of properties
- ▶ Structure of primitive and mediation module assembly
- ▶ Error handling
- ▶ Examine custom code samples

In summary, this presentation provided details regarding the Custom Mediation primitive, along with an overview of its function and information about the primitive's use of terminals and its properties. The structure of the primitive in relationship to the mediation module assembly was also described. Finally, information about error handling was presented and two code examples were examined.

Trademarks, Copyrights, and Disclaimers

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

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

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

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

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

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

Linux is a registered trademark of Linus Torvalds.

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

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

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR 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 (e.g., IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

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

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

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

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

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