



IBM Software Group

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

Mediation Primitive Common Details



@business on demand.

© 2006 IBM Corporation
Updated May 1, 2006

This presentation discusses those aspects of mediation primitives that are common to mediation primitives in general.

Goals

- Provide the basic understanding needed before examining individual primitives
 - ▶ Review concepts of mediation primitives
 - ▶ Describe elements common to mediation primitives



The goal of this presentation is to provide a basic understanding prior to discussing each of the individual mediation primitives. This is done by reviewing the concepts of mediation primitives and describing the elements which are common across many or all of the primitives.

Section

Review of Mediation Primitive Concepts



In this section the concepts of mediation primitives are reviewed.

Place of Mediation Primitives in the Big Picture

- **Mediation Modules:**
 - ▶ Mediate messages flowing between service requestors and providers
 - Handle protocol transformations
 - Update content of the message
 - Modify format of the message
 - Dynamically route service requests/responses
 - ▶ Contain a Mediation Flow Component
- **Mediation Flow Components:**
 - ▶ Used to define the mediation flow logic
 - ▶ Unique flow logic defined for every operation of a service interface
- **Mediation Primitives**
 - ▶ Used to construct the logic of a mediation flow
 - ▶ Each primitive performs some specific part of the flow logic



In order to understand mediation primitives, it is important to understand where they fit into the big picture of mediations. Starting at the highest level of abstraction, there are Mediation Modules whose function is to mediate messages flowing between service requestors and service providers. Mediating a message might involve handling protocol transformations, updating the content of the message, modifying the format of the message and/or dynamically routing the message to an appropriate service provider. The Mediation Module contains a Mediation Flow Component, which is where the overall logic for the mediation is defined. For every operation defined on an input interface there is unique mediation flow logic for the operation's request and response. The flow logic is defined within the Mediation Flow Component using Mediation Primitives. Each mediation primitive provides some specific portion of the logic and the overall logic is defined by wiring these mediation primitives together into a logical flow.

To summarize, the highest level of a mediation is the Mediation Module, which contains a Mediation Flow Component, which contains mediation flows defined using Mediation Primitives.








Mediation Primitive Types

- Built-in primitives
 - ▶ Provide a pre-defined functional capability
 - ▶ Properties are used to configure details
- Custom Mediation primitive
 - ▶ User defined functionality implemented in Java
- User defined primitives
 - ▶ User defined functionality implemented in Java
 - ▶ Packaged as WebSphere Integration Developer plug-in
 - ▶ Provides custom function with look of a built-in
 - ▶ See article on the WebSphere Integration Developer support site
 - **Contributing your own mediation primitive plug-in**
 - ▶ How this is done may change in future releases

Mediation primitives fall into a few basic categories.

- Built-in primitives provide some pre-defined function, such as logging a message or doing a lookup in a database. You customize the pre-defined function through the specification of property values, such as identifying a database table.
- Custom Mediation primitive can be used when none of the built-in primitives meet your requirements. This allows you to define the primitive function using Java™ coded to your own specification.
- User defined primitive are similar to a custom primitive, and allow you to define functionality using Java. However, you then package it as a plug-in to WebSphere Integration Developer, which allows for reuse of the user defined primitive with the same look and feel as a built-in primitive. The use of user defined primitives is described in an article titled “Contributing your own mediation primitive plug-in” available from the WebSphere Integration Developer support Web site. Interfaces for user defined primitives in the current release are subject to change and therefore release to release compatibility might not be maintained.

Mediation Primitive Types

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

The different types of mediation primitives are shown here.

- The Message Logger mediation primitive is used to log all or part of the contents of the message to a database.
- The Message Filter mediation primitive is used to modify the path through a flow by selectively forwarding the message based on the evaluation of simple expressions. Each expression is associated with an output terminal defining where the message will be forwarded.
- The Database Lookup mediation primitive is used to access information from a database and insert it in the message. A field in the message is used as a key for the database access and selected fields from the resulting database row can be placed into the message.
- The XSLT mediation primitive is used to transform messages using XSL transformation. This can be used to change the format of the message when the target provider has a different interface than the incoming message.
- The Stop mediation primitive is used to stop a path through the mediation flow without raising an exception.
- The Fail mediation primitive is used for error conditions and will stop the mediation flow and cause an exception to be thrown.
- The Custom mediation primitive is used to do message processing not covered by the other mediation primitives. This is done through Java code that can be written as a visual or Java snippet or can also be written as an operation in a Java SCA component.

Details of each of the mediation primitives are covered in individual presentations. The remainder of this presentation will continue to cover common aspects of all mediation primitives.

Mediation Primitives in the Mediation Flow Editor

The screenshot displays the Mediation Flow Editor interface for a CustomerRoutingFlow. The top panel, titled 'Operation connections', shows a flow from 'CustomerService' to 'CustomerServicePartner', both with 'getCustomerInformation' operations. The middle panel, 'Mediation Flow', shows a sequence of operations: 'CustomerService_get...', 'LogMessage', 'DBLookup', and another 'CustomerService_get...'. The bottom panel, 'Properties', shows the details for the 'Database Lookup : DBLookup' primitive, including its display name, name, and description.

Four callout boxes on the left side of the screenshot identify key components:

- Operation Connection:** Points to the red-bordered box in the top panel.
- Mediation Flow:** Points to the green-bordered box in the middle panel.
- Mediation Primitives:** Points to the blue-bordered box in the middle panel.
- Properties of Selected Mediation Primitive:** Points to the purple-bordered box in the bottom panel.

At the bottom of the screenshot, the text 'Mediation Primitive Common Details' and '© 2006 IBM Corporation' is visible.

Shown here is a screen capture of the Mediation Flow Editor. The top panel of the editor is called Operations Connections and contains the input and output interfaces along with all of their operations. Every operation on the input interface must be wired to operations on the output interface. Selecting any particular input operation will show the mediation flow for that operation in the middle panel of the editor. Within the flow, mediation primitives are wired together to define the logic of the flow. Selecting any specific mediation primitive on the editor will display the properties for that primitive in the Properties view which is in the bottom panel. This is where the properties are specified to configure the behavior of the primitive.

Section

Common Elements of Mediation Primitives

Now that the concepts of mediation primitives have been covered, this section will go into more detail on those elements that are common to the mediation primitives.

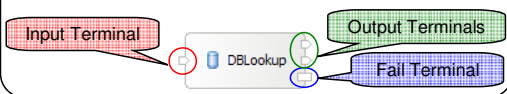
Terminals - Mediation Primitive Input and Output

- Terminals:
 - ▶ Define a Mediation Primitive's input and output message type
 - ▶ Message types define the content of the Service Message Object body
- Input terminals
 - ▶ Defines input message type
 - ▶ One per primitive
- Output terminals
 - ▶ Defines output message type
 - ▶ Zero, one or more output terminals per primitive (based on primitive type)
 - ▶ Possibly required to have same message type as input terminal (based on primitive type)
- Fail terminal
 - ▶ Used when a primitive fails during the flow
 - ▶ Message type must be the same as input terminal message type
 - ▶ Propagates the original message updated to contain failure information

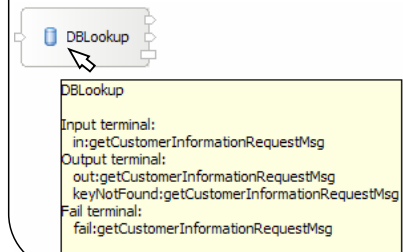
All mediation primitives have terminals, which are used to define the input and output of the primitive, specifically identifying the message type that will flow through the terminal. The message type is defined by the structure of the Service Message Object body that will be present in that part of the flow. There is one Input terminal per primitive to define the input message type. An Output terminal defines the output message type. The number varies by type of primitive with each primitive having zero, one, two or a variable number of output terminals. For many mediation primitives, the output terminal must be for the same message type as the input terminal since the primitive is not capable of changing the structure of the SMO body. However, in some cases the output terminal can be for a different message type. The Fail terminal is used when the mediation primitive fails in some way while processing the message. Because the original message is propagated when there is a failure, the Fail terminal will always be for the same message type as the input terminal. The message will be updated to contain information about the failure.

Terminals in the Mediation Flow Editor

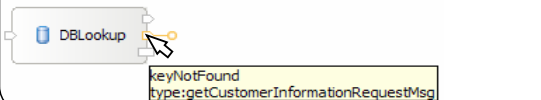
Terminal representation



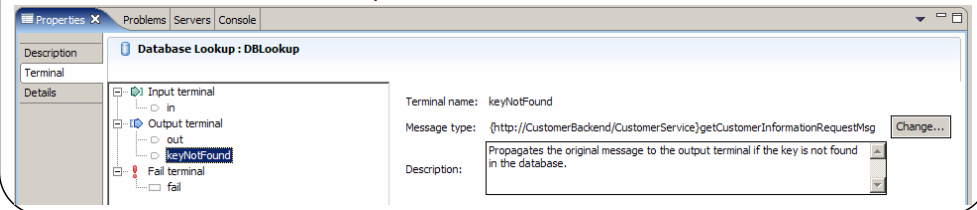
Hover over primitive displays all terminals and type



Hover over terminal displays name and type



Terminals are shown in Properties view



10

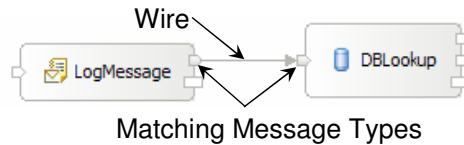
Mediation Primitive Common Details

© 2006 IBM Corporation

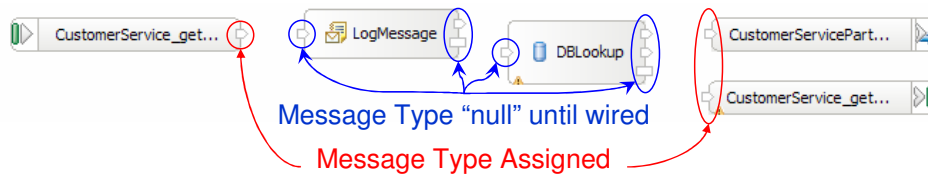
This slide examines how terminals are represented in the Mediation Flow Editor. Starting in the upper left is a mediation primitive showing the Input terminal, which will always be on the left side, two Output terminals, which will be on the right side, and the Fail terminal, which will be the lower terminal on the right side. Notice that the Fail terminal has a different shape than either the Input or Output terminals. Moving down to the center left illustration, you will see that if you hover the mouse pointer over a terminal, a popup will appear specifying the name of the terminal and the message type associated with that terminal. The illustration on the upper right shows that when you hover the mouse pointer over the primitive, a popup appears specifying the name of the primitive along with the name and message type of all the terminals for that primitive. Finally, on the bottom is a screen capture of the Terminal tab in the Properties view of the mediation primitive. Selecting any terminal in the list on the left displays the name and type of the terminal on the right. Notice that the message type here is the fully qualified type rather than the short version of the type shown in the popups. Also notice the **Change...** button which will open a dialog for modifying the message type associated with the terminal.

Wiring of Mediation Primitive Terminals

- Connections between terminals are represented with wires
- A connection must have matching terminal message types



- Editor dynamically manages terminal message types
 - ▶ Input and callout nodes have fixed terminal message types
 - ▶ Primitives have dynamically configured terminal message types



The next couple of slides will look at the behavior of the Mediation Flow Editor relative to the assignment of message types to terminals during the process of wiring a mediation flow. As illustrated in the upper graphic, connections between terminals are represented with wires, which can only connect terminals having the same message type. During the process of wiring together terminals, the editor dynamically manages the terminal types. In the lower graphic you see a mediation flow that has one input node on the left, two callout nodes on the right and two mediation primitives in the middle. At this point, none of these are wired together. Notice that the nodes have message types assigned to their terminals, whereas the mediation primitives do not yet have message types assigned to their terminals.

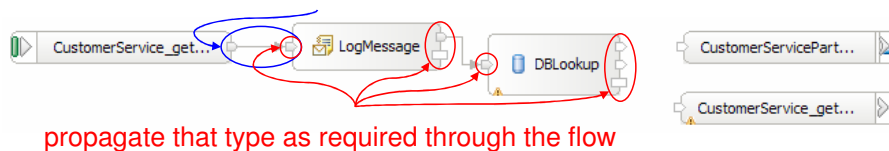
Wiring of Mediation Primitive Terminals

- Example of wiring and message type handling

Wiring "null" to "null" type keeps type as "null"



Adding a wire from a terminal with a defined type will ...



propagate that type as required through the flow

- Terminals can be given static message type values
 - Message type can be statically set in Properties View
 - Attempts to wire with terminals of unlike message type prevented

Continuing from the previous slide, the top graphic shows that when you wire the null message type output terminal of the LogMessage primitive to the null message type input terminal of the DBLookup primitive, the terminals will still have a null message type.

In the next graphic, the output terminal of the input node is wired to the input terminal of the LogMessage primitive. Because the output terminal of the input node has a specific message type assigned, that message type is dynamically assigned to the input terminal of the LogMessage primitive so that the wire connects terminals of like message type. Since a Message Logger primitive must have the same output message type as its input message type, the editor dynamically assigns the message type to the output and fail terminals of the LogMessage primitive. Since there is a wire between the output terminal of the LogMessage primitive and the input terminal of the DBLookup primitive, the message type is propagated so that the wire is connecting terminals of like type. Finally, since a Database Lookup primitive must have the same output message type as its input message type, the editor dynamically assigns that type to the output and fail terminals of the DBLookup primitive.

In addition, the editor would return all the terminals to have a null message type if the wire that started the message type propagation was removed. So you can see that the mediation flow editor makes it quite easy to manage terminal message types when wiring a flow.

It is also possible to assign a specific message type to a terminal so that it is static and will override the dynamic assignment of message type. When you do this, the editor prevents you from wiring the terminal with the static message type to anything other than a terminal with the same type or a terminal with a null message type.

Mediation Primitive Exceptions

- Exceptions associated with mediation primitives:
 - ▶ **MediationConfigurationException**
 - For a configuration problem or a transient external resource failure
 - Example: Database table cannot be found or accessed
 - ▶ **MediationBusinessException**
 - For business error when executing the primitive
 - Example: A key that should be in a message is not found
 - ▶ **MediationRuntimeException**
 - There are runtime problems when setting up the mediation flow
 - Example: Incorrect JNDI name for a datasource

The specific exceptions that Mediation primitives can throw are described here.

MediationConfigurationException is used when there is a configuration problem or a transient problem with an external resource, such as not being able to find or access a database.

MediationBusinessException is used when what appears to be a business logic problem occurs when executing a primitive. An example of this kind of problem would be when the database key value configured for a primitive cannot be found in the message.

MediationRuntimeException occurs when there is some kind of problem initializing a mediation flow. An example of this would be when the JNDI name for a datasource is incorrect.

These exceptions and error processing are described in more detail in following slides.

Mediation Primitive Exception Handling

- Mediation primitive exceptions can be thrown:

- ▶ While setting up and initializing the flow

- MediationRuntimeException
- MediationConfigurationException

- ▶ While processing the primitive itself

- MediationConfigurationException
- MediationBusinessException
- MediationRuntimeException

- Exception behavior is the following:

```
IF
    exception thrown during processing of primitive itself
    AND the Fail terminal is wired
THEN
    continue without logging exception
    follow connection from Fail terminal
ELSE
    log the exception
    terminate the mediation flow
```

In order to provide an understanding of the exception processing behavior, it is important to know the different points at which a mediation exception can be thrown. Some initialization is done by the runtime to set up a mediation flow. This initialization occurs prior to control being given to any mediation primitives. Exceptions that might be thrown at this time are the `MediationRuntimeException` or the `MediationConfigurationException`, with the `MediationRuntimeException` being the most common. After initialization of the flow, an exception can be thrown during the processing of a mediation primitive. Normally these will be a `MediationConfigurationException` or a `MediationBusinessException`, but in some cases a `MediationRuntimeException` could also be thrown.

The behavior for processing exceptions will be different based on a couple of factors, such as whether or not the exception is thrown during the processing of a primitive and whether or not the Fail terminal is wired. The behavior can be described as follows: If the exception is thrown during the processing of the primitive and the Fail terminal is wired to some other primitive or node, then the exception is not logged and the mediation flow continues following the wire from the Fail terminal. In all other cases, the exception will cause a log message to be written and the mediation flow to terminate.

Mediation Primitive Configuration Exceptions

- Configuration parameters are checked at different times
 - ▶ During setup and initialization of the flow
 - ▶ During the processing of the primitive itself
- Handling of these exceptions will therefore be different
- Example:
 - ▶ Database Lookup mediation primitive
 - Fail terminal is wired
 - Configuration parameters include:
 - Datasource JNDI name
 - Database table name
 - ▶ Datasource JNDI name not found is discovered during setup, therefore
 - MediationRuntimeException thrown
 - Log written
 - Flow terminates
 - ▶ Database table not found is discovered during primitive processing
 - MediationConfigurationException thrown
 - No log written
 - Fail terminal connection followed

When there is a problem with something in the configuration of a mediation primitive, you will see the different behaviors described on the previous slide. This is because configuration parameters are checked at different times, some being checked at the initialization of the flow and others being checked during the processing of the mediation primitive. For example, assume you have a Database Lookup primitive that has its Fail terminal wired. Two of the configuration parameters for a Database Lookup are the Datasource JNDI name and the Database table name. The Datasource JNDI name is checked during the setup of the flow, therefore the result of an incorrect JNDI name will be a `MediationRuntimeException` thrown with a log written and the mediation flow terminated. However, the Database table name is checked during the processing of the primitive itself. Therefore, when the `MediationConfigurationException` is initially thrown it is caught, no log is written and the mediation flow continues by following the wire from the Fail terminal of the Database Lookup.

In the presentations for each of the individual mediation primitives you will see specific information about which conditions result in which of these behaviors.

Error Information in Message

- When the Fail terminal connection is followed:
 - ▶ Error information is added to the Service Message Object
 - ▶ It is placed inside of the “context” with a “failInfo” tag
 - ▶ The following is an example:

```

- <context>
- <failInfo xml:lang="en">
  <failureString>CWSXM3206E: Unable to connect to the database using the property
  values specified in the mediation primitive.</failureString>
  <origin>DatabaseLookup1</origin>
  <invocationPath>
    <primitive inTerminal="in" name="MessageLogger1" outTerminal="out" />
    <primitive inTerminal="in" name="DatabaseLookup1" outTerminal="fail" />
  </invocationPath>
</failInfo>
</context>

```

Diagram annotations:

- Failing Primitive:** Points to the `<origin>DatabaseLookup1</origin>` tag.
- Error Message:** Points to the `<failureString>` tag.
- Path taken through flow:** Points to the `<invocationPath>` section.

When the Fail terminal is wired and an exception occurs within a mediation primitive, information about the error is added to the failInfo section of the context section of the Service Message Objects (SMO). The following information is added:

The failureString contains a text description of the error that occurred.

The origin contains the name of the mediation primitive in which the exception occurred.

The invocation Path contains a list of every mediation primitive through which the message flowed up to and including the primitive in which the error occurred. In addition, the names of the terminals through which the message passed are also listed with each primitive.

With this information, logic in the flow might be able to determine what action to take in response to the failure.

Mediation Primitives and XPath

- Mediation primitives operate on Service Message Objects
- Service Message Objects are accessed using XPath 1.0 expressions
- Many configuration properties are XPath expressions
 - ▶ These properties are set using the “XPath Expression Builder” dialog
 - ▶ Expressions built with the expression builder are called Custom XPath expressions
- Several primitives have a configuration property called “Root”
 - ▶ Represents the portion of the Service Message Object that will be used
 - ▶ Values for root can be selected with a drop down box and generally contain
 - / the entire Service Message Object
 - /body the body of the message (operation and parameter or return values)
 - /context the message context (transient context, correlation context and failInfo)
 - /headers protocol headers and arbitrary properties
 - ▶ Custom XPath expressions are sometimes valid for Root properties
- XPath expressions cannot be null

Another element common to mediation primitives is that they operate on Service Message Objects and that XPath 1.0 expressions can be used to access the data within the SMO. Many of the configuration properties used by primitives are expressed as XPath expressions. There is a dialog in WebSphere Integration Developer called the XPath Expression Builder that can be used to construct XPath expressions. Expressions built in this way are typically referred to as Custom XPath expressions.

Several of the primitives have a property called Root, which defines the portion of the SMO that is to be used by the primitive during its processing. The valid values for root properties are not consistent across all of the primitives. In some cases the root is specified with a drop down box with the choices of “/”, “/body”, “/context” or “/headers” with “/” meaning the whole SMO and the others referring to the each of the three major sections of the SMO. In other cases, the ability exists to use the XPath Expression Builder to construct the expression for root. Null XPath expressions will cause an exception at runtime.

Section

Summary

The next slide presents a summary of this presentation.

Summary

- Reviewed basic concepts of mediation primitives
 - ▶ Where they fit in the overall mediation picture
 - ▶ Types of primitives
 - ▶ High level look at the Mediation Flow Editor
- Described common elements of mediation primitives
 - ▶ Terminals
 - ▶ Wiring
 - ▶ Exceptions and error handling
 - ▶ XPath
- Provided the base knowledge needed to look at details of individual primitives



In this presentation, the basic concepts of mediation primitives were reviewed, including a description of where they fit into the overall mediation picture, what types of mediation primitives there are and an introduction to how they are edited in the Mediation Flow Editor.

Common elements of mediation primitives including terminals, wiring, exceptions, error handling and XPath usage were also examined.

With the understanding provided by this presentation, you should now be better prepared to understand the specifics of each individual mediation primitive.

Trademarks, Copyrights, and Disclaimers

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

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

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

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

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

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

Linux is a registered trademark of Linus Torvalds.

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

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

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

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

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

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

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

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

