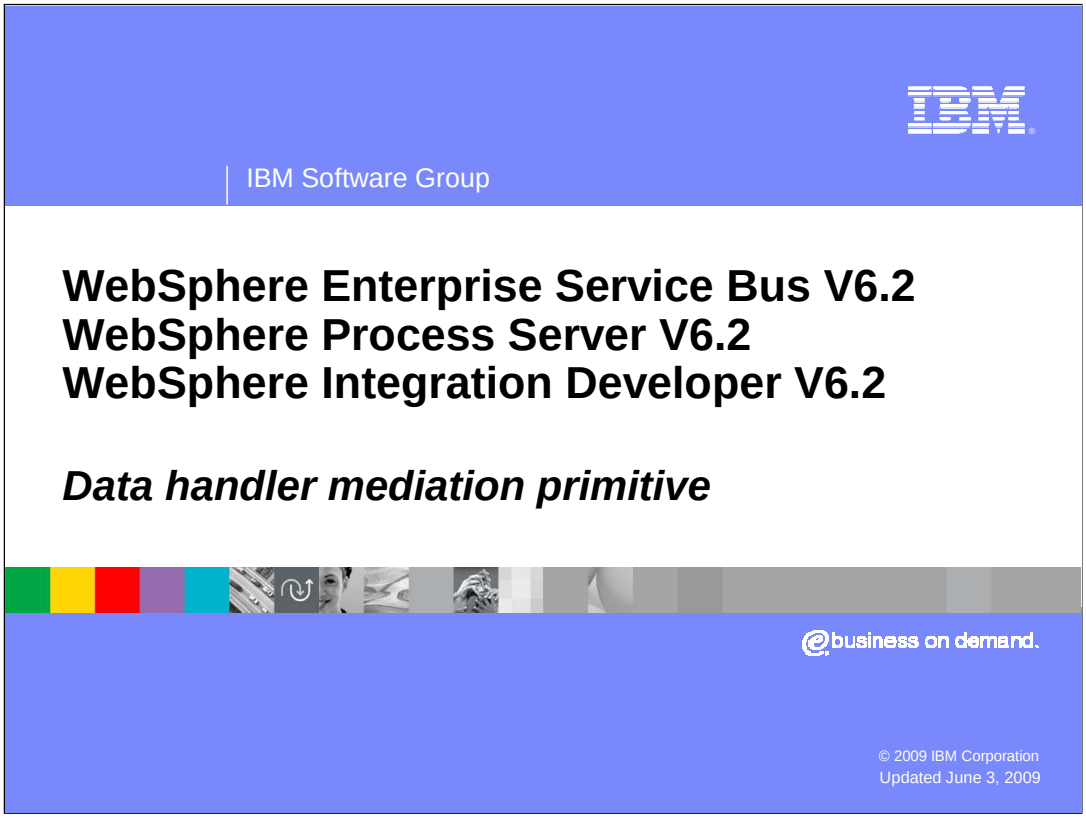


This presentation provides a detailed look at the data handler mediation primitive, which is a new primitive introduced in version 6.2.

Goals

- Understand the data handler mediation primitive

Data handler

- Overview of function
- Use of terminals
- Definition of properties
- Error handling
- Example usage

The goal of this presentation is to provide you with a full understanding of the data handler mediation primitive.

The presentation assumes that you are already familiar with the material presented in the presentations that cover common elements of all mediation primitives, such as properties, terminals, wiring and the use of promoted properties. The general knowledge of mediation primitives they provide is needed to understand the data handler primitive specific material in this presentation.

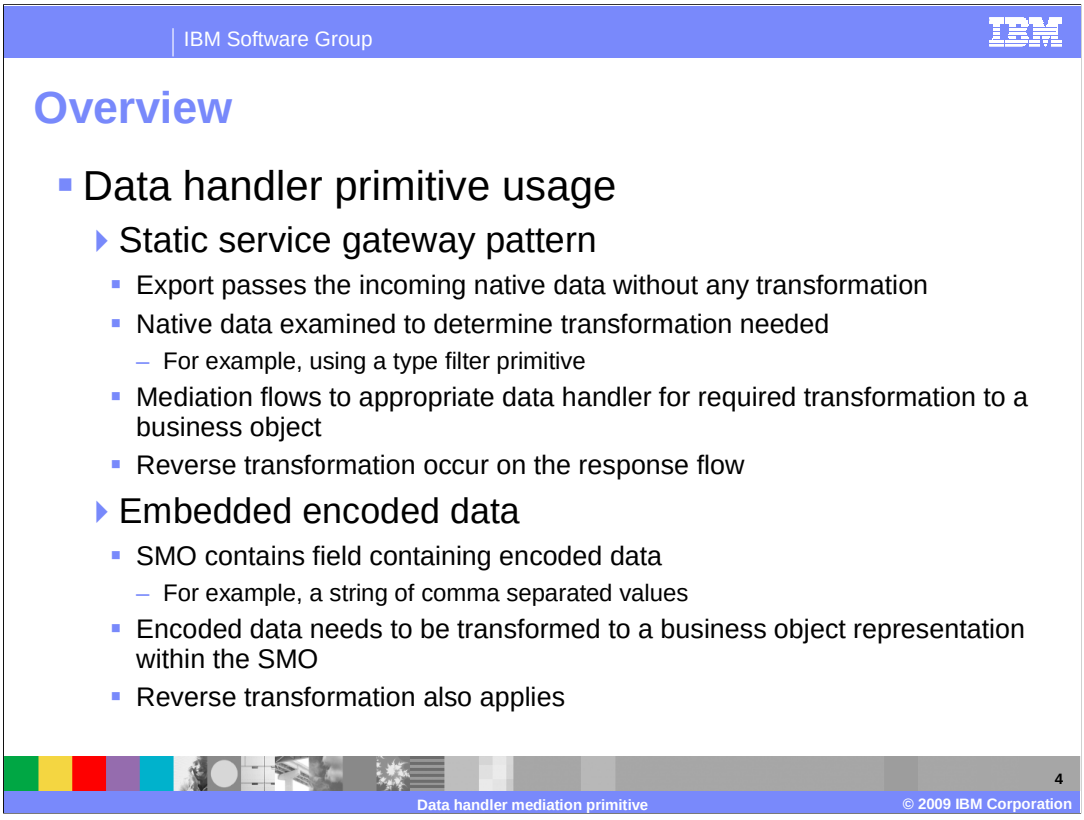
The presentation contains an overview of the function provided by the data handler primitive, along with information about the primitive's use of terminals and its properties. The error handling characteristics are then covered and finally an example usage of a data handler primitive is provided.

Overview

- The data handler primitive:
  - Enables data transformation between:
    - Native formats and business objects
    - Business objects and native formats
  - Provides the same data transformation capabilities in a mediation flow as is normally done by an export or import
  - Is configured using a binding resource configuration
    - Combination of a data handler implementation and configuration parameters
- Data handlers used can be:
  - Prepackaged implementations provided with the product
  - Custom implementations provided by you

The purpose of the data handler is to enable transformations between business objects and native data formats within a mediation flow. The type of transformations enabled are the same as those normally done at the edges of the flow by an export or an import. Similar to the export and import, the data handler is configured with a binding resource configuration which combines a data handler implementation with configuration parameters, the combination of which define the transformations that are performed.

Some data handler implementations are provided with the product as are some binding resource configurations. These can be used if they meet your requirements, or you can provide your own custom implementation of a data handler if your transformation requirements differ.



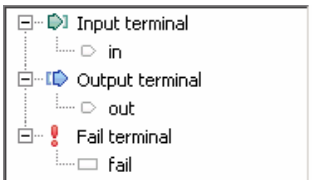
In some scenarios it makes sense for a data handler primitive to perform transformations between business objects and native formats within a mediation flow, rather than on the edges as is normally done using imports and exports.

One such scenario is a mediation implementing a static service gateway pattern. With this pattern the export supports the generic service gateway interface and the imports support some concrete interfaces. The export passes the data to the mediation flow without any transformation of the inbound native data. Within the flow, the native data is examined to determine the type of transformation that is needed, which might be done using a type filter, message filter or custom mediation primitive. The mediation then flows to an appropriate data handler primitive to perform the native format to business object transformation. The response flow will contain a data handler primitive to perform the business object to native format transformation of the return data.

Another scenario for use of the data handler primitive is when a business object contains a field with encoded data, such as a comma separated value string. Within the flow, the data handler primitive can be used to perform the transformation of the encoded data to a business object. Reverse transformations from business object to encoded data can also be done when required to build an outbound message.

Terminals

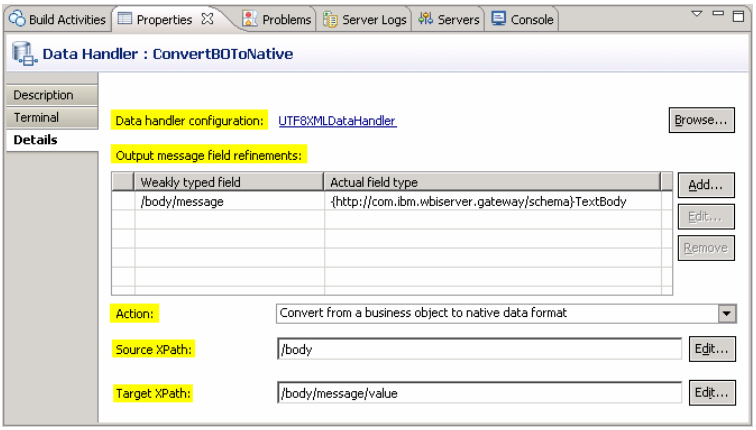
- Terminals:
  - One input terminal
  - One output terminal
  - Fail terminal
- Terminal message types
  - Message type is not propagated between input and output terminals
  - Input and output determined:
    - Implicitly by the message type of the terminal they are wired to
    - Explicit by setting of message type
  - Fail terminal is the same message type as the input terminal



The data handler primitive has one input terminal, one output terminal and a fail terminal. The input and output terminals do not need to be of the same message type. Therefore, the message type is not propagated between these terminals when wiring a flow. The message type for these terminals is determined implicitly by the message type of the terminal they are wired to, unless the message type has been explicitly set using the change terminal type dialog. The fail terminal always has the same message type as the input terminal.

Properties

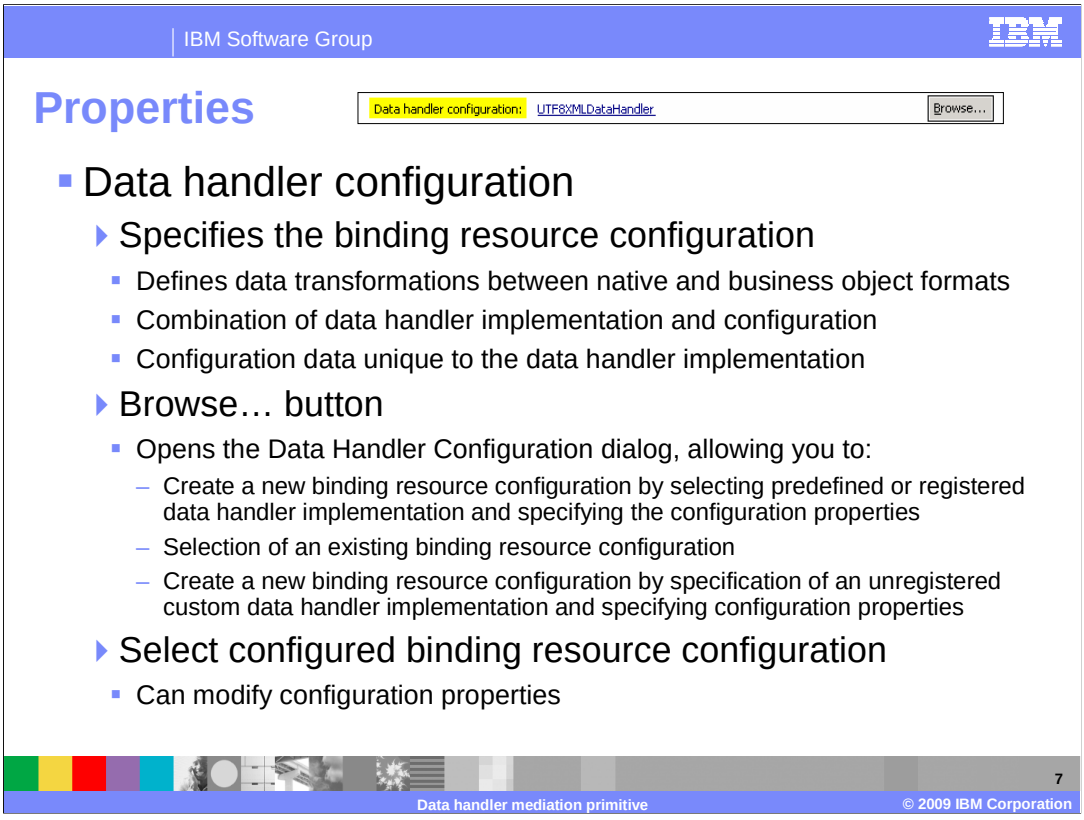
- Data handler properties
  - Details panel



  - No promotable properties

This slide shows a screen capture of the Details panel of the Properties view for a data handler primitive, with the property names highlighted. Each of these properties are described in the subsequent slides.

Note that there is no Promotable Properties tab in the Properties view, indicating that none of the data handler properties are promotable.

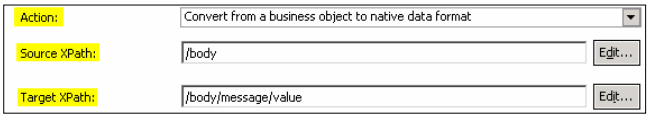


The first property is the Data handler configuration which is used to specify the binding resource configuration used by this data handler. The binding resource configuration defines the transformations that occur between business object and native formats. It combines the specification of a data handler implementation along with the configuration parameters that are needed to customize the data handler's behavior. The configuration parameter data needed is unique to each of the data handler implementations. For example, the configuration parameters for a fixed width data handler implementation are different than those needed for a delimited string data handler implementation.

You set the value for this property using the Data Handler Configuration dialog which is accessed using the Browse… button. This dialog gives you some choices as to how you specify the binding resource configuration. One approach is to create a new binding resource configuration. This is done by selecting from a list of predefined and registered data handler implementations and then specifying the configuration properties needed to customize the data handler's behavior. Alternatively, in the list of data handler implementations are sub lists showing existing binding resource configurations which can be selected. Some of these are predefined by the product and some are ones that you had previously defined. If none of the predefined or registered implementations or binding resource configurations suit your requirements, then you can select the Java™ class of an implementation which you have provided and specify the configuration parameters it needs. The dialog allows you to register this so that it is placed in the list of predefined and registered implementations, enabling you to easily reuse it.

If your data handler primitive already has a binding resource configuration specified, clicking on the name of it in this panel opens a panel that allows you to modify the configuration properties.

Properties

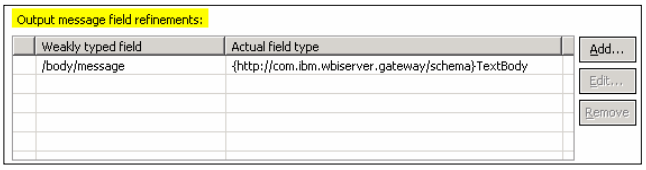


- Action
  - Specifies the direction of the conversion
    - Convert from native data format to a business object
    - Convert from a business object to native data format
- Source XPath and Target XPath
  - Specifies the SMO elements for source and target
  - Edit… button opens the XPath Expression Builder dialog

The Action property is used to define the direction of the transformation, indicating either to transform from native data format to business object or from business object to native data format.

The Source XPath and Target XPath define the elements in the service message object that are the source and target of the transformation. Using the Edit… button opens the XPath Expression Builder, enabling you to easily construct the appropriate XPath expression.

Properties



- Output message field refinements
  - Table defining refinements of weakly typed fields in SMO
    - Strong typing required by some data handlers, must know output type
    - Adds type refinement to message type of the output terminal similar to function of the set message type primitive
  - Add… and Edit… buttons
    - Open Add/Edit properties dialog
    - XPath Expression Builder dialog used for specification of the weakly typed field
    - Data Type Selection dialog used for specification of the actual field type
  - Commonly needed for response flow in static service gateway
    - Data handler primitive is created in the response flow by the service gateway wizard
    - Unlike other data handlers, in this case the type refinement is not added to the output terminal

The Output message field refinements property is a table that allows you to provide type information for weakly typed fields. When the service message object contains a weakly typed field as the target of the transformation, the data handler implementation might need to know its exact type. Examples of a weakly typed field are an anyType or a concrete type from which other types are derived.

The Add… and Edit… buttons are used to open the Add/Edit properties dialog which is used to define a row in the table. In the dialog, the weakly typed field can be set using the XPath Expression Builder dialog and the actual field type can be set using the Data Type Selection dialog.

A very common use for this property occurs in the static service gateway pattern response flow where the gateway return type is defined to be an anyType. An anomaly occurs if the data handler is created in a response flow by the service gateway wizard. The type refinement is not added to the output terminal in this case because it is typically not needed and can lead to conflicts in wiring if there are multiple data handlers in the flow.

Error processing

- MediationRuntimeException thrown for:
  - Incomplete configuration data
    - For example, missing source or target XPath
  - Incorrect transformation type
    - For example, specifying business object to native format transformation with source XPath identifying native format data
- MediationBusinessException thrown for:
  - Wrapper of DataHandlerException
    - For example, input or output type in SMO not supported by data handler
  - Wrapper of JXPathException
    - Input XPath not found in SMO

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

A MediationRuntimeException is thrown for cases resulting in incorrect configuration of the data handler primitive. An example of this is a missing XPath expression for either the source or target. Another example is an incorrect transformation such as specifying a business object to native format transformation when it should be a native format to business object transformation.

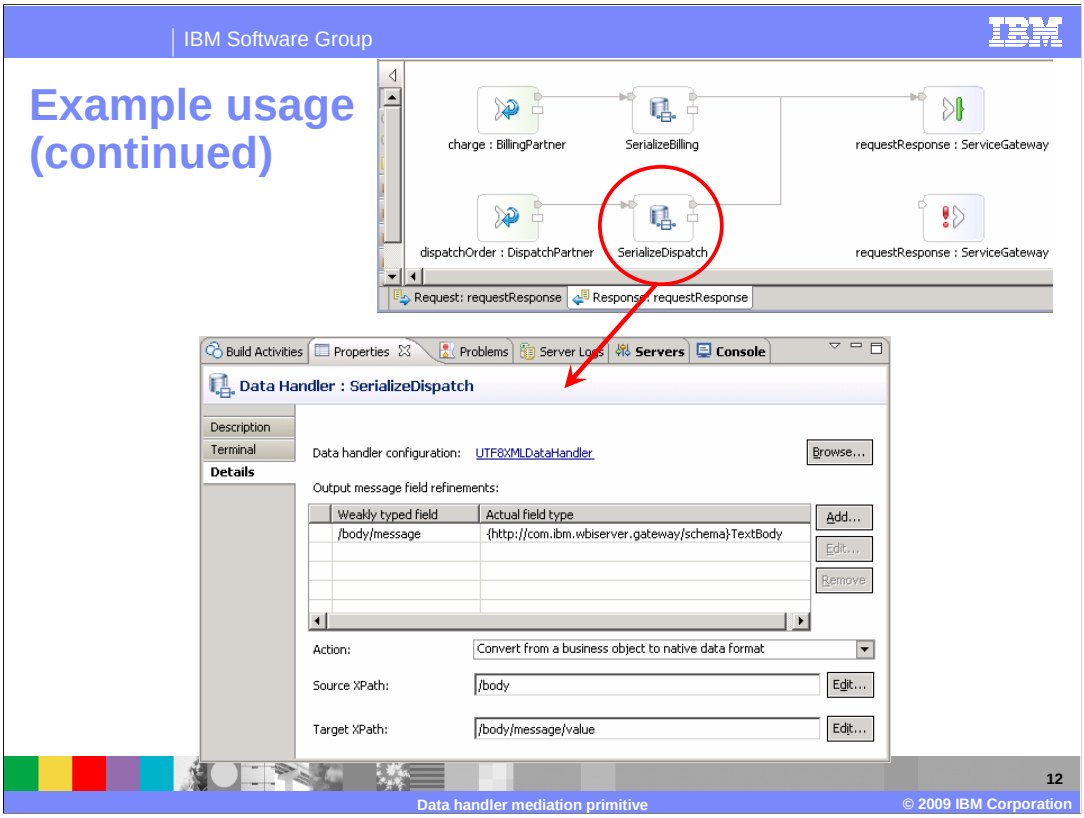
The MediationBusinessException is generally thrown as a wrapper to an exception thrown by the data handler implementation. For example, a DataHandlerException results when a source or target type in the service message object is not supported by the data handler. Another example is a JXPathException when an XPath specified for source or target is not found in the service message object.

Example usage

- Example
  - Basic characteristics of the flow
    - Static service gateway
    - Service gateway interface on export
    - Concrete interfaces on imports
    - Web service bindings for both imports and exports
  - Illustration – use of data handler in response flow
    - Data handler receives business object containing response data
    - Response data needs to be serialized to be returned
    - Built in UTF8XMLDataHandler resource configuration used to serialize data
    - Service gateway interface return value is an anyType
    - Data handler needs to know specific type for serialized target
    - Service gateway predefined TextBody type specified in field refinements

This slide introduces an example usage of a data handler primitive in the context of a flow that implements a static service gateway pattern. In this pattern, the export supports the service gateway interface and the imports support concrete interfaces. For this example, both the export and imports are configured with Web service bindings. The example is explained on this slide and screen captures illustrating the data handler primitive properties are shown on the next slide.

Although data handlers are used in both the request and response flow for this pattern, this example focuses on the response flow because the data handler needs to make use of the output message field refinements property. The data handler in the response flow receives a business object containing the response data that needs to be serialized before it is returned. To do this, the predefined UTF8XMLDataHandler binding resource configuration is used. The service gateway interface has a return value which is an anyType. However, the data handler needs to know what the specific type of the target is. Therefore, the predefined TextBody type provided for use with service gateways is specified in the output message field refinements table.



This slide contains screen captures illustrating the example described on the previous slide. The top portion shows a response flow for a static service gateway mediation flow that handles two operations. The properties for one of the data handler primitives in the flow is shown at the bottom of the slide. This is the typical data handler configuration for the response flow of a static service gateway.

Summary

- Examined the data handler mediation primitive

Data handler

  - Overview of function
  - Use of terminals
  - Definition of properties
  - Error handling
  - Example usage

In summary, this presentation provided details regarding the data handler mediation primitive. It presented an overview of the data handler primitive's function, along with information about the primitive's use of terminals and its properties. Error handling characteristics were then presented and finally an example usage of a data handler was provided.

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_DataHandlerPrimitive.ppt

This module is also available in PDF format at: ../WBPMv62_DataHandlerPrimitive.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

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.