



IBM Software Group

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

***Service invoke mediation primitive***



@business on demand.

© 2009 IBM Corporation  
Updated June 26, 2009

This presentation provides a detailed look at the service invoke mediation primitive.

## Goals

- Understand the service invoke mediation primitive



Service invoke

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

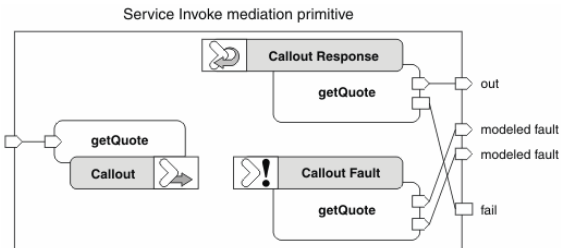
The goal of this presentation is to provide you with a full understanding of the service invoke mediation primitive.

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 service invoke primitive specific material in this presentation.

This presentation starts with an overview of the service invoke primitive and provides information about the primitive's use of terminals and its properties. A discussion of invocation styles is then presented. It is an important topic for helping you to understand the behavior of a mediation flow containing a service invoke. Then some error handling information is discussed and an example use case of the service invoke primitive is provided.

## Overview of function

- Invokes a service from within a flow
  - ▶ Can be used in either a request or response flow
- Conceptually similar to the combination of:
  - Callout node
  - Callout response node
  - Callout fault node



- Associated with:
  - ▶ A reference on the mediation flow component in the module assembly
  - ▶ A specific operation on the reference
    - Operation can be request/response or one way
  - ▶ Reference can be wired or dynamic endpoints can be used



The purpose of the service invoke primitive is to enable you to call an external service from within a mediation flow, either a request flow or a response flow.

In many ways, it is similar to a combination of a callout node and its associated callout response node and callout fault node. This is illustrated in the graphic. The service invoke is represented as the enclosing rectangle and the relationship between the terminals of a service invoke and the terminals of the three nodes are shown. This is described in more detail later in the presentation.

A service invoke primitive is associated with a reference on the mediation flow component in the assembly diagram, and to a specific operation on the interface of that reference. The operation can be a request response operation or can be a one way operation. The service invoke primitive can be configured to make use of a dynamic endpoint taken from the SMO rather than using the import the reference is wired to in the assembly.

## Overview of function

- Possible to be asynchronous to the mediation flow
  - ▶ Asynchronous with callback style of service invocation
  - ▶ Resumption of mediation flow upon callback
- A flow can have multiple service invoke instances
  - ▶ A series of service invocations
  - ▶ Parallelism possible using asynchronous with callback
- Configurable retry behavior
  - ▶ Retry invocation after a modeled or unmodeled fault
  - ▶ Number of times to retry
  - ▶ Delay between retries
  - ▶ Use of alternate endpoints

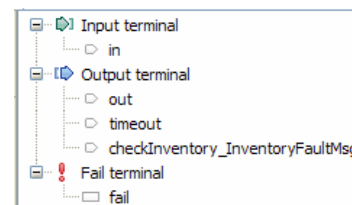
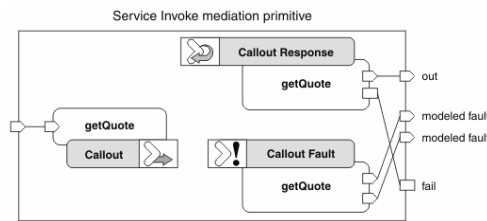
When a service invoke primitive makes a call to an external service, it is possible for that call to be either synchronous or asynchronous to the mediation flow. In the case where it is asynchronous, the mediation flow is resumed upon callback. A complete discussion of invocation styles and the effect of synchronous versus asynchronous calls are discussed later in this presentation.

There can be multiple service invoke primitive instances in the same mediation flow. This might be a series of service invocations in a sequential flow. Additionally, when there are multiple paths through a flow with each containing a service invoke, the service invocations have the possibility of being processed in parallel if the asynchronous with callback invocation style is used.

The service invoke primitive can be configured to retry failing service calls. The configuration allows you to specify if it should be done for modeled faults, unmodeled faults or both. Additional configuration properties specify a retry count, a retry delay and if alternate endpoints should be used when retrying the call.

## Terminals

- One input terminal
  - ▶ Message type = request message for the operation
- 1 to n output terminals
  - ▶ One output terminal for the response
    - Message type = response message for the operation
    - Does not exist for one way operations
  - ▶ One output terminal for timeouts
    - Message type = request message for the operation
    - Only used for asynchronous timeouts
  - ▶ One output terminal for each modeled fault for the operation
    - Message type = fault message for the operation and fault
- One fail terminal
  - ▶ Message type = request message for the operation



5

Service invoke mediation primitive

© 2009 IBM Corporation

This slide looks at the terminals that are used with a service invoke primitive. In the top center of the slide is a screen capture of a service invoke primitive as it appears in a flow. On the center right is a screen capture of the terminals panel from the properties view. As was mentioned earlier, the service invoke can be thought of as a combination of the callout node, callout response node and callout fault node, which is illustrated in the upper right corner.

A service invoke primitive has one input terminal, called in, whose message type is defined by the operation being called on the external service. This is similar to the in terminal of a callout node.

The service invoke can have from one to n output terminals. The first output terminal is called out. It has a message type defined by the response for the operation being called. This can be compared to the out terminal of a callout response node. For a one way operation, this terminal does not exist.

The next output terminal is called timeout and is used when an asynchronous timeout occurs waiting for the response. The message type is the same as the message type of the in terminal. For synchronous operations, this terminal is present but will never be fired. Callouts have no equivalent terminal to the timeout terminal. They use the fail terminal of the callout response node to return asynchronous timeouts.

The remaining output terminals represent the faults defined for the operation being called, with one terminal for each fault. The message type of each terminal is defined by the fault. These are the same as the terminals of the callout fault node.

Finally, there is the fail terminal whose message type is the same as the message type of the in terminal. This terminal can be compared to the fail terminal of a callout response node.

IBM Software Group IBM

## Properties

- Properties controlling service invocation
  - ▶ Reference name
  - ▶ Operation name
  - ▶ Use dynamic endpoint
  - ▶ Async timeout
  - ▶ Asynchronous with callback not allowed
  - ▶ Invocation style

6

Service invoke mediation primitive © 2009 IBM Corporation

The Details panel of the Properties view is shown here.

The first two properties are the Reference name and the Operation name. They define the reference, and therefore the interface, that this service invoke is associated with and the specific operation on that interface to call. On this panel, these properties are read only. They are specified when the service invoke primitive is created and cannot be changed. When a service invoke is used in a subflow, the reference is defined on the subflow, then when an instance of the subflow is created, the reference is resolved to a reference on the parent flow.

The Use dynamic endpoint if set in the message header property, causes the service invoke primitive to check the SMO for an endpoint reference in the field headers/SMOHeader/Target. If there is one set, that endpoint reference is to used define the endpoint rather than the reference's wire target. If a target endpoint reference is not set in the SMO, the wire target of the reference is used.

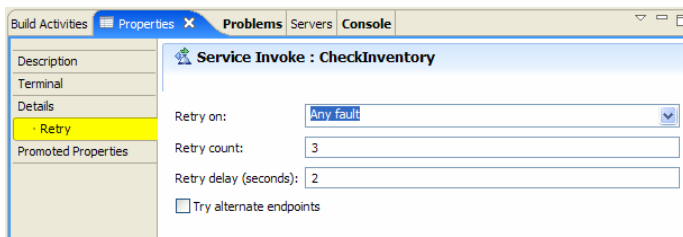
The Async timeout property specifies how many seconds to wait for a response before the timeout terminal is fired. It applies when the service is invoked using asynchronous with deferred response. If the call is synchronous or asynchronous with callback, this property is ignored.

The next property on this panel can be referred to as Asynchronous with callback not allowed. The actual wording on the panel says: Require mediation flow to wait for service response when the flow component is invoked asynchronously with callback. This property setting is used to prevent the asynchronous with callback invocation style from being used. In the case where asynchronous with callback can be used except for this property being selected, the actual invocation occurs as asynchronous with deferred response. Note that this property is implicitly set for any service invoke primitives that exist in a flow between a fan out and fan in primitive.

The last property on the panel is Invocation style, which can be set to Sync, Async or Default. The synchronous and asynchronous settings are used to override the invocation style that is selected by default based on other factors. The factors influencing invocation style are discussed later in this presentation.

## Properties for retry

- Retry on
  - ▶ Never
  - ▶ Any fault
  - ▶ Modeled fault
  - ▶ Unmodeled fault
- Retry count
- Retry delay
- Try alternate endpoints



\* See separate [Service call retry](#) presentation for details



The properties for retry are specified on their own panel.

The Retry on property can be set to never, indicating that retry processing should not be performed. It can also be set to Modeled fault, Unmodeled fault or Any fault, indicating which type of faults should result in retry processing.

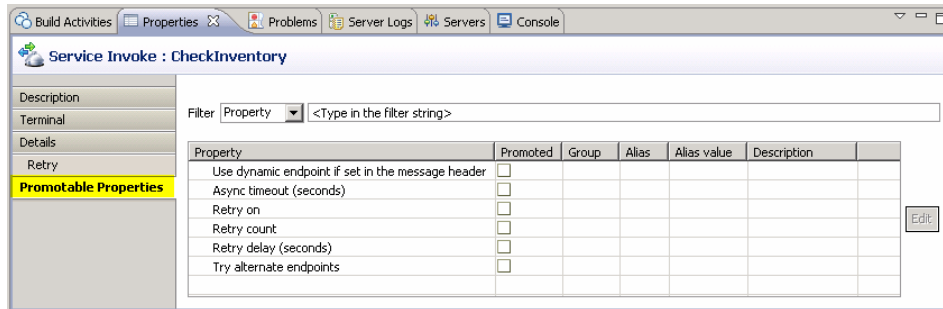
The Retry count property specifies how many times to attempt a retry when faults continue to occur during retry processing. Once the retry count is reached, the fault that occurred on the final attempt is returned to the flow.

The Retry delay specifies how many seconds to wait between a fault being returned and the next retry attempt.

Finally the Try alternate endpoints specifies that alternate target endpoint references from the SMO should be used when performing the retries. These are located in a sequence at headers/SMOHeader/AlternateTarget.

Details of retry processing, as it applies to callouts and service invoke primitives, is explained in a separate presentation entitled Service call retry.

## Promotable properties



- Promotable
  - ▶ Use dynamic endpoint
  - ▶ Async timeout
  - ▶ Retry on
  - ▶ Retry count
  - ▶ Retry delay
  - ▶ Try alternate endpoints
- Not promotable
  - ▶ Reference name
  - ▶ Operation name
  - ▶ Asynchronous with callback not allowed
  - ▶ Invocation style



This slide shows the Promotable Properties panel for a service invoke primitive.

The properties Use dynamic endpoint and Async timeout are both promotable.

Also, all of the properties associated with retry processing are promotable, which are Retry on, Retry count, Retry delay and Try alternate endpoints.

The Reference name and Operation name are not promotable. Changing either of these requires corresponding development time changes to the flow. Likewise, the Asynchronous with callback not allowed and Invocation style have the potential to require changes to the logic of the flow and are therefore not promotable.



## Invocation styles – SCA defined styles

- Synchronous
  - ▶ Synchronous call to the service (SCA “invoke”)
  - ▶ Mediation thread blocked until service returns
- Asynchronous with deferred response
  - ▶ Asynchronous call to the service (SCA “invokeAsync”)
  - ▶ Mediation thread blocked until response received (SCA “invokeResponse”)
  - ▶ Async timeout property specifies maximum wait time
- Asynchronous with callback
  - ▶ Asynchronous call to the service (SCA “invokeAsyncWithCallback”)
  - ▶ Mediation thread continues
    - Anything additional wired on input side of service invoke primitive continues
    - Terminates once all additional processing is completed
  - ▶ Callback with response starts new mediation thread
    - Anything wired on output side of service invoke primitive runs on this thread

9

Service invoke mediation primitive

© 2009 IBM Corporation

These next few slides discuss the topic of invocation styles used by service invoke primitives. The invocation styles are the service component architecture defined styles of invocation which provide the underlying implementation. This slide describes these SCA defined styles as they apply to mediation flows.

The Synchronous style of invocation uses the SCA API invoke operation to make the call. This causes the thread on which the mediation is running to block until the response is received.

The Asynchronous with deferred response style of invocation uses the SCA API invokeAsync operation. From an SCA perspective, this allows the thread making the call to continue in parallel, and then it can use the SCA invokeResponse operation to receive the response. However, the service invoke primitive performs the invokeResponse immediately after the invokeAsync, so effectively no parallel processing occurs. It does allow the asynchronous timeout property to cause a timeout if the response is not received within that timeframe.

The Asynchronous with callback style of invocation uses the SCA API invokeAsyncWithCallback. From an SCA perspective, this allows the thread making the call to continue in parallel, and the response to be received on a new thread kicked off by a callback. As this applies to the mediation flow, the thread on which the mediation is running will continue if there is more work to be done. Basically, this means that in a mediation with multiple flow paths, anything wired on the input side of the service invoke will continue until all is completed. The thread then terminates. The response is received on a new thread which is kicked off starting on the output side of the service invoke primitive.

## Invocation styles - Comparing styles

- Comparing synchronous versus asynchronous with deferred response
  - ▶ Very little difference in overall behavior of the mediation
  - ▶ Asynchronous with deferred response appears synchronous, with these exceptions:
    - One way operations with a reference qualifier of asynchronous invocation set to commit
    - Async timeout property and timeout terminal apply
  - ▶ Iterative aggregation scenarios are an exception
    - Parallel processing of a sequence of items is possible



In the context of the service invoke primitive, generally there is really very little difference in the behavior you will see between the synchronous and asynchronous with deferred response styles of invocation. This is because the service invoke primitive calls `invokeResponse` right after `invokeAsync`, and thus the mediation flow blocks waiting for the response similar to the synchronous style. There are a couple of differences that are seen between these two styles. There is a qualifier used on references, called asynchronous invocation, which can have a value of `call` or `commit`. When a one way operation is invoked using the asynchronous with deferred response style and asynchronous invocation is set to `commit`, the actual invocation of the service does not happen until the containing transaction commits. The second difference is that the asynchronous with deferred response style can result in a timeout causing the timeout terminal to be fired.

In general, other than these differences, the two styles exhibit the same behavior in a mediation flow. However, there is one key exception to this. It is for an aggregation scenario, where the service invoke primitive is contained in an iterative flow between a fan out and fan in primitive. In this case, the asynchronous with deferred response can be used to enable parallel processing of a sequence of items. This case is described later in the presentation.

## Invocation styles – Default style determination

- Default invocation style determined by:
  - ▶ Invocation style used to call the mediation flow component
  - ▶ The preferred invocation style of the target service
  - ▶ Whether the request is one way or request/response

MFC Invocation Style	Target Preferred Interaction Style	OneWay/Request-Response	Invocation Style used for Callout
invoke(sync)	ANY	OneWay	Async
		Request-Response	Sync
	both	Sync	
invokeAsync	ASYNC	both	Async
	ANY	OneWay	Async
		Request-Response	Sync
SYNC	both	Sync	
invokeAsyncWith Callback	ANY	both	Async
		Request-Response	AsyncWithCallback
	SYNC	both	Sync
	ASYNC	OneWay	Async
		Request-Response	AsyncWithCallback

11

Service invoke mediation primitive

© 2009 IBM Corporation

The default invocation style used by the service invoke is based on a combination of factors. One of the factors is the invocation style that was used to call the mediation flow component. Then, the preferred invocation style of the target service is considered and whether the operation is one way or request response.

This slide contains a table defining the invocation style based on these various factors. The first column defines the invocation style used to call the mediation flow component. The second column defines the preferred interaction style of the target. The third column defines if the operation is one way or request response. Finally, the rightmost column indicates the resulting invocation style used by the service invoke primitive. You might notice that the last column title indicates the invocation style used for callout, but the rules followed are the same for callouts and service invoke primitives. Rather than going through the entire table, it is provided for your reference.

## Invocation styles – Applicable properties

- Invocation style property
  - ▶ Default – determined as described on previous slide
  - ▶ Sync – synchronous call used
  - ▶ Async – either of the asynchronous styles can be used
- Asynchronous with callback not allowed
  - ▶ Forces asynchronous with callback to be asynchronous with deferred response



In addition to the determination of the default invocation style as described on the previous slide, there are a couple of additional properties that influence the actual invocation style used. The first property to consider is the invocation style property. Specifying the value default results in the default behavior just described. Specifying the value sync results in the service call being made synchronously. Specifying the value async results in an asynchronous with deferred response call, unless the default is determined to be asynchronous with callback, and then that is used.

The other property influencing this is the asynchronous with callback not allowed property. When all the other factors indicate that the invocation style of asynchronous with callback should be used, setting this property causes the call to be done using asynchronous with deferred response.

## Invocation styles – Parallel processing

- Parallel processing possible in two cases
  - ▶ Using asynchronous with callback, only possible when:
    - Mediation flow component invoked with asynchronous with callback
    - The operation is a request/response
    - Preferred interaction style of target is not “sync”
    - The invocation style property is not set to “sync”
    - The asynchronous with callback not allowed property is not selected
  - ▶ Parallel processing within an iterative aggregation
    - Fan out configured to allow parallel calls within the aggregation
    - Service invoke invocation style resolves to asynchronous with deferred response



There are two ways to make use of a service invoke primitive to obtain parallel processing of service calls. The first occurs when the invocation style of asynchronous with callback is used. However, this can only occur in very specific circumstances. First of all, the mediation flow component must have been called using the asynchronous with callback style and the flow must be for a request response operation. Additionally, neither the preferred target interaction style, nor the invocation style property can be set to the value sync. Finally, the property asynchronous with callback not allowed can not have been set. This last property setting implies the service invoke is not within an aggregation flow, because an aggregation flow between a fan out and fan in causes this property to be implicitly set.

The other way to obtain parallel processing is within an iterative aggregation between a fan out and fan in. This is one of the key scenarios for parallel processing, allowing a sequence of items within a message to be processed in parallel. To do this, the fan out must be configured to allow parallel service calls through specification of a property defining the maximum number of parallel calls. Also, the service invoke must resolve to an invocation style of asynchronous with deferred response.

## Error processing

- **MediationRuntimeException** thrown for:
  - ▶ **Property value is not valid:**
    - WebSphere® Integration Developer ensures valid values
    - Possible if promoted property value incorrectly set at runtime
- **Other possible exceptions you might see:**
  - ▶ **Shown in failInfo when fail terminal wired:**
    - NullPointerException – Reference not wired
    - IllegalArgumentException – Reference wired to import with no binding
  - ▶ **Same situations and fail terminal not wired:**
    - SIBXFireFailTerminalException



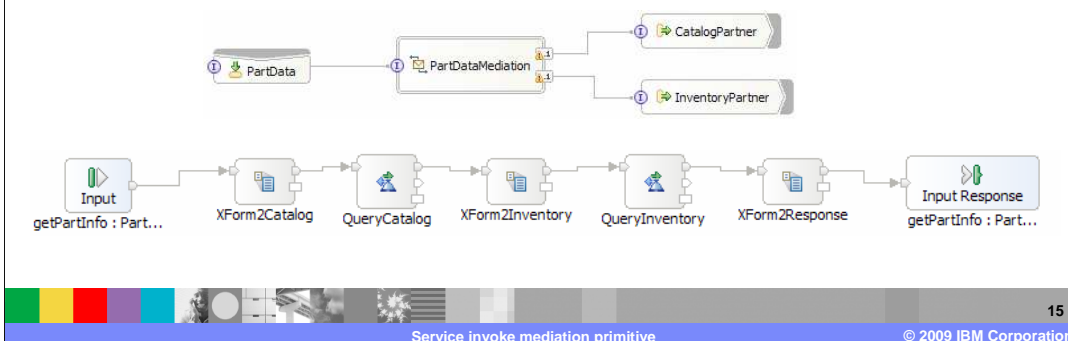
The error processing details and considerations are examined in this slide.

A `MediationRuntimeException` is thrown when a property value specified for the service invoke is not valid. Because WebSphere Integration Developer validates property values, this problem is only seen when the incorrect property value has been set administratively for a property that has been promoted. The retry on property is an example of one for which this can occur.

Other possible exceptions you might encounter involve problems with the assembly, and vary depending upon whether the fail terminal is wired. When the fail terminal is wired, the exceptions are seen in the `failInfo` section of the SMO. An unwired reference results in a `NullPointerException` and a reference wired to an import with no binding results in an `IllegalArgumentException`. For the case where the fail terminal is not wired, these same two situations both result in an `SIBXFireFailTerminalException`.

## Example usage

- Combine result of two services into one response
  - ▶ Query information for a specific part number
  - ▶ Returns a business object containing
    - Part number
    - Detailed description (from the catalog service)
    - Stock on hand (from the inventory service)



This slide illustrates one of the possible use cases for the service invoke primitive. In this scenario, the mediation is used to combine the results of calls to two services to build a response to the incoming request. The incoming request provides a part number, and the response is a business object containing the part number, a description of the part and the quantity currently in inventory. The description for the part is obtained from a catalog service and the quantity from an inventory service.

In the slide you can see the assembly diagram showing the export exposing the part data service, the part data mediation containing the flow and the two imports for the catalog and inventory services.

Below the assembly diagram is the mediation flow. Notice that there is no callout for this flow. Calls to the catalog service and inventory service are both required to build the response, so these are both done using the service invoke primitive. The response is then built within the request flow and returned to the caller using the input response node in the request flow.

In order to make use of the service invoke primitives, the message type of the SMO must be transformed to setup each request and to process each response. Therefore, looking at the flow, you see that the first primitive is an XSL transformation, called XForm2Catalog. It takes the part data request message received as input and transforms it into a catalog request message. It also saves the incoming part number in the transient context. The QueryCatalog service invoke primitive calls the catalog service and the resulting SMO is a catalog response message. The next XSL transformation, XForm2Inventory, saves the description from the response in the transient context and sets up the message to be an inventory request message. The QueryInventory service invoke primitive now calls the inventory service and the resulting SMO is an inventory response message. The final step is for the XSL transformation, XForm2Response, to build a part data response message. It gets the part number and description from the transient context and the inventory quantity from the inventory response message body. The input response node then returns this to the caller.

## Summary

- Examined the service invoke mediation primitive



Service invoke

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



In summary, this presentation provided details regarding the service invoke primitive, providing an overview of its function and information about the primitive's use of terminals and its properties. A discussion of invocation styles was presented to help you understand the behavior of a mediation flow containing a service invoke. Some error handling information was discussed and an example use case for the service invoke primitive was described.



## 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\\_ServiceInvokePrimitive.ppt](mailto:iea@us.ibm.com?subject=Feedback_about_WBPMv62_ServiceInvokePrimitive.ppt)

This module is also available in PDF format at: [..\\WBPMv62\\_ServiceInvokePrimitive.pdf](..\\WBPMv62_ServiceInvokePrimitive.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>

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.

© Copyright International Business Machines Corporation 2009. 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.