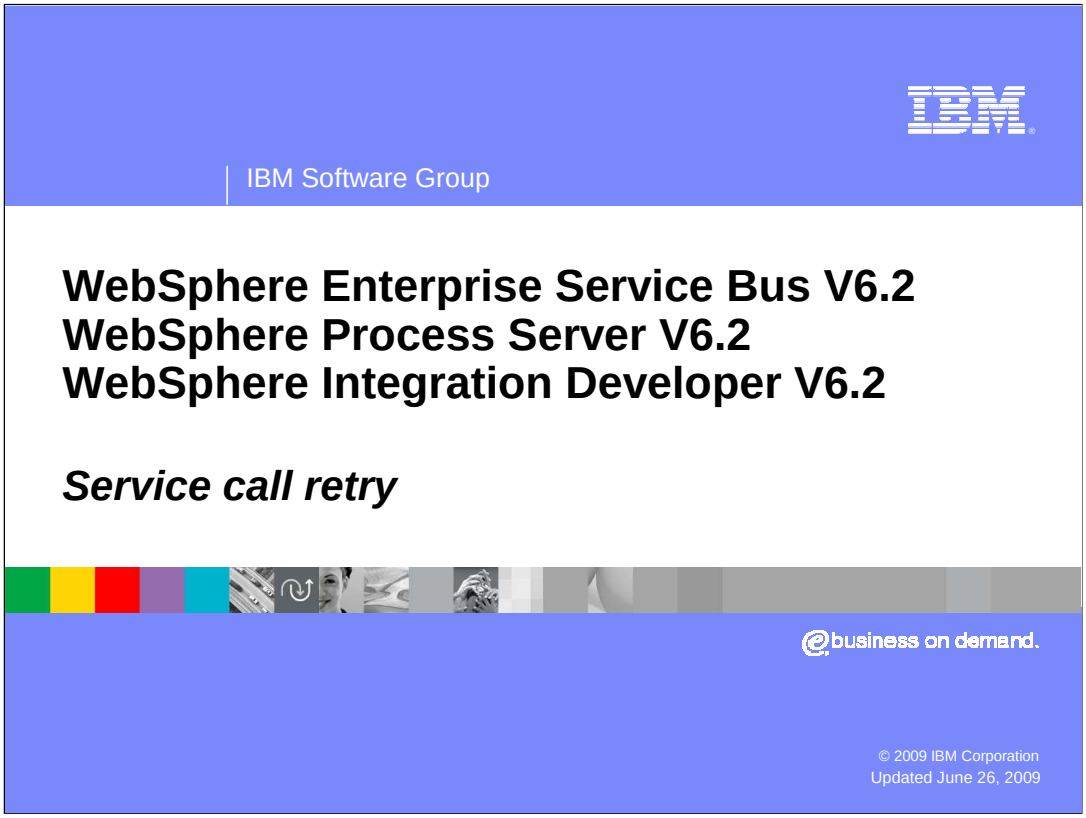


This presentation provides a detailed look at the service call retry capabilities for service invoke primitives and callout nodes.

Goals

- Introduce service call retry
  - Overview of the capabilities
  - Properties for configuration
  - Retry behavior
  - Retry with alternate endpoints
  - Example scenarios

The goal of this presentation is to provide you with a good understanding of the service call retry capabilities. The presentation starts out with an overview of what service call retry provides. It then looks at the properties that are used to configure service call retry and describes the behavior resulting from the various configuration settings. Finally, a couple of example scenarios of retry configuration are presented.

Overview of retry capability

- Service call retry:
  - Enables automatic retry when a service calls return a fault
  - The fault is not returned to the mediation flow unless all retries are unsuccessful
  - You configure the retry behavior
- Service call retry capabilities are enabled for:
  - Service invoke primitives
  - Callout nodes
- Retries can be made for:
  - The same service at the same endpoint
  - The same service at different (alternate) endpoints

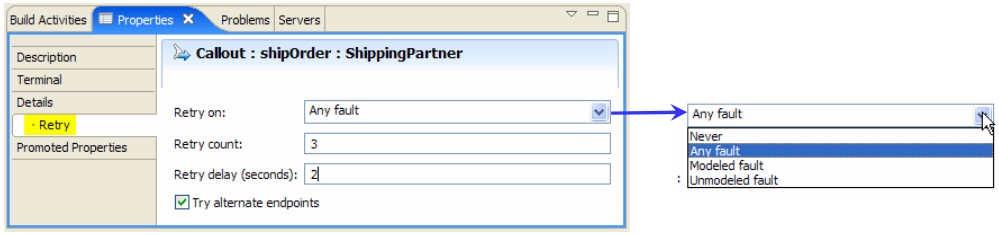
Service call retry provides an automatic mechanism to retry a service call after receiving a fault from the service. With retry, the fault is not returned to the mediation flow unless all retries have been attempted and the service call is still resulting in a fault. Properties are provided so that you can configure the retry behavior according to your application requirements. Both service invoke primitives and callout nodes provide for retry of service calls. When a retry is performed, configuration settings allow you to indicate if the same service endpoint should be used or if an alternate endpoint should be tried.

Overview of retry capability

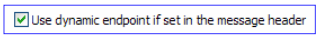
- For callout nodes and service invoke primitives:
  - The retry related properties are the same
  - The behavior is the same
- Configurable for:
  - Modeled versus unmodeled faults
  - Maximum number of retries to attempt
  - Delay between retries
  - Alternate endpoints
- Alternate endpoints
  - Dynamic endpoint must be enabled
  - Alternate endpoints passed in the SMOHeader

The retry properties used to configure a service invoke primitive or a callout node are the same, and the resulting behavior is also the same. The configuration properties include designation of the type of fault that should trigger a retry and the maximum number of times a retry should be attempted. You can also configure if there should be a delay before performing a retry and if the retry should be to the original service endpoint or to an alternate endpoint. When alternate endpoints are used the service invoke primitive, or callout node, must have been configured for the use of dynamic endpoints and the alternate endpoint addresses must be present in the SMOHeader.

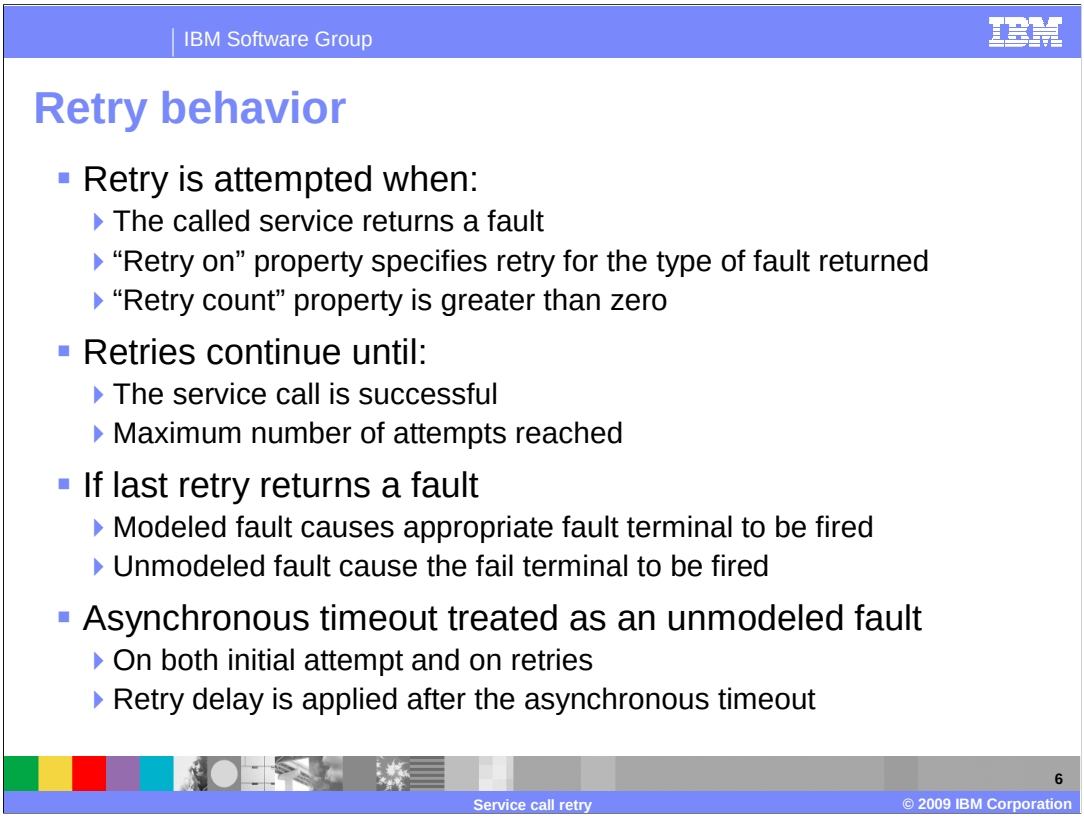
Properties for configuring retry



- Same panel used for service invoke primitives and callout nodes
- Retry on – which type of faults trigger a retry
- Retry count - how many times to retry after initial fault
- Retry delay - number of seconds to wait between retries
- Try alternate endpoints
  - Indicates to retry using alternate endpoints from the SMO
  - Only applies if the use dynamic endpoint property is also set on details panel

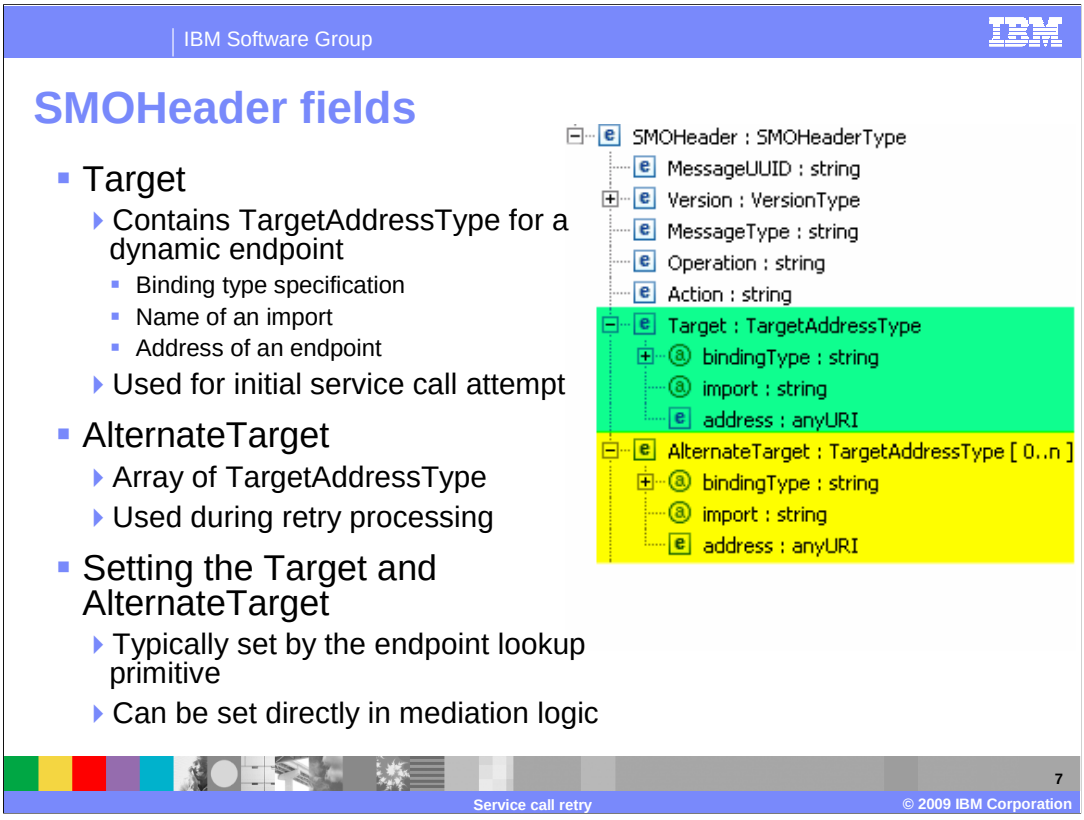


This slide shows a screen capture of the properties panel used to configure retry capabilities. This panel is identical for both service invoke primitives and for callout nodes. The property called retry on is used to specify the type of fault that will trigger a retry. The choices for this property are shown in a screen capture of the drop down box seen on the right side of the slide. As you can see the choices are never to try, retry on any fault, retry only for modeled faults or to retry only for unmodeled faults. The retry count specifies the maximum number of times to retry before returning the fault to the mediation flow. The retry delay allows you to specify a delay, in seconds, between the return of the fault and the retry of the service call. The try alternate endpoints indicates you want the retries to be performed using the endpoints specified in an alternate endpoints field in the SMO. When alternate endpoints is selected, the property on the details panel label “Use dynamic endpoint if set in the message header" must also be selected.



The behavior of service call retry is described here. A retry is attempted only when the service call returns a fault whose type matches the types specified in the retry on property and the retry count is greater than zero. The retries will continue until either there is a successful service call, or the maximum number of retries specified in the retry count property has been reached. In the case where the maximum retry count has been reached and the last retry returned a fault, the fault is returned to the mediation flow. In the case of a modeled fault, the terminal for that specific modeled fault is fired on either the service invoke primitive or the callout response node. Likewise, if it is an unmodeled fault, the fail terminal is fired. For a detailed explanation of handling of faults in a mediation flow, see the unmodeled fault presentation for which a reference is provided at the end of this presentation.

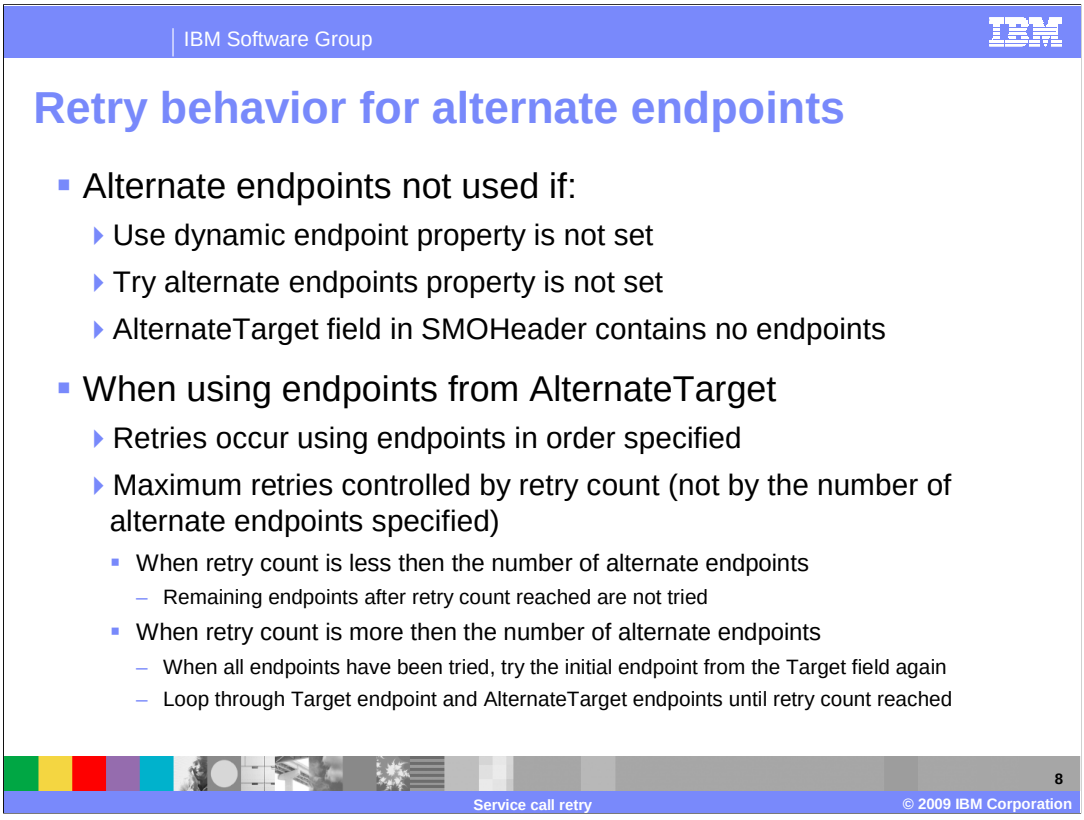
Finally, in the event that a service does not respond, an asynchronous timeout occurs, which is treated as an unmodeled fault. If there is a retry delay configured, that delay is applied after the asynchronous timeout. Therefore, the total time between the initial call and the retry is the sum of the timeout value and delay value.



The screen capture in the upper right shows the SMOHeader, contained in the header portion of the service message object. The Target field contains a TargetAddressType describing a dynamic endpoint to be used as the endpoint for the initial service call. The dynamic endpoint is described with a TargetAddressType, composed of some combination of a binding type, import name and a URI address for an endpoint. For more information on the use of the TargetAddressType and dynamic endpoints, see the dynamic invocation presentation referenced at the end of this presentation.

The next field is the AlternateTarget which is an array of TargetAddressType used during retry processing.

The setting of the Target and AlternateTarget addresses is typically done using the endpoint lookup primitive which accesses the WebSphere® Service Registry and Repository to obtain endpoint addresses. However, any primitives in the mediation flow can be used to set the Target and AlternateTarget addresses in the SMOHeader.



When doing a retry, alternate endpoints are not used unless all of these are true. First, the use dynamic endpoints property, which controls the use of the target address in the SMO for the initial service call, must be selected. Then the try alternate endpoints property must also be selected. If both these are selected and the TargetAddress field in the SMO contains addresses, then these alternate target addresses are used for the retry. If not, the retry is done using the same endpoint as the initial attempt.

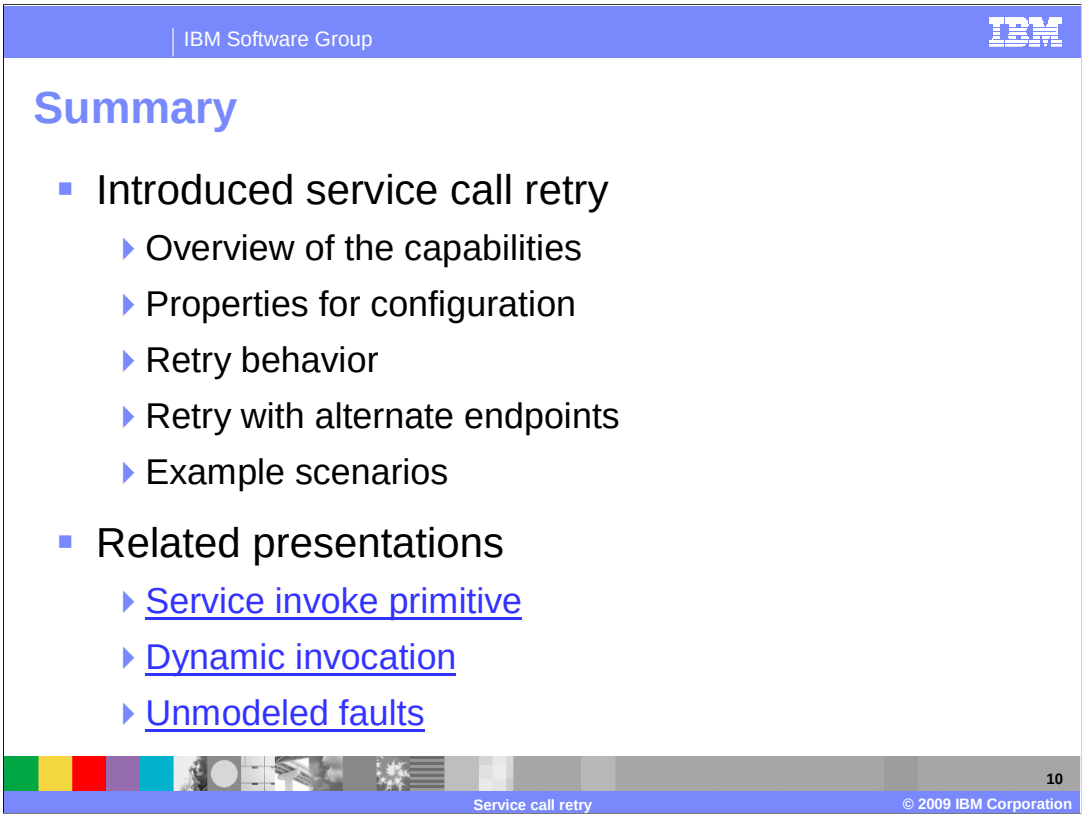
When the alternate target endpoints are used for retry, the attempts are made to the endpoints in the same order that they appear in the array. The number of retries that occur is still controlled by the retry count property and not by the number of alternate endpoints in the array. If the retry count is less than the number of alternate endpoints, then the endpoints in the array beyond the maximum retry count are not attempted. In the case where the retry count is greater than the number of alternate endpoints, the initial endpoint is retried again after the alternates are exhausted. Retries will continue, looping through the endpoints in the Target field and the AlternateTarget array until the maximum retry count is reached.

Example scenarios

- Endpoint with high level of availability
  - Service endpoint automatically restarted if terminated
  - Quality of service guarantees outage never longer than ten seconds
  - Configure:
    - Retry count of five
    - Retry delay of two seconds
- Group of endpoints that back each other up
  - Quality of service guarantees always a backup endpoint available
  - Service outage of a single endpoint might not be resolved quickly
  - Configure:
    - Retry with alternate endpoints
    - Retry count equal to the number of alternate endpoints
    - Retry delay of zero seconds

The use of service call retry is dictated by your particular application requirements. This slide provides a couple of examples to illustrate the application of retry. In the first scenario, there is a service with a single endpoint that automatically restarts if it goes down. Also, it provides a quality of service guarantee that it will never be down for longer than 10 seconds. In this case, you can configure your service invoke primitive or callout node to have a retry count of five with a delay of two seconds. Assuming the service meets its quality of service guarantee, the retry will eventually be successful.

In the second scenario, there are several endpoints for the same service which provide backup for each other. The quality of service guarantee provided by the service is that there will always be at least one of the endpoints available, but any single endpoint might be unavailable for an extended period of time. In this case, you configure you retry to use alternate endpoints with a retry count equal to the number of service endpoints provided, and the retry delay can be set to zero.



In this presentation you were introduced to service call retry capabilities for service invoke primitives and callout nodes; starting with an overview of the function. The properties used for configuration and the resulting behavior were described. Finally, a couple of example usage scenarios were provided.

To fully understand retry capabilities in the context of mediation flows, you might find these presentations useful. In the mediation primitives section is a presentation on the service invoke primitive. In this section, mediation flows, there are presentations describing dynamic invocation and unmodeled faults. The latter not only addresses the handling of unmodeled faults, but modeled faults as well.

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_RetryServiceCalls.ppt

This module is also available in PDF format at: ../WBPMv62_RetryServiceCalls.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.