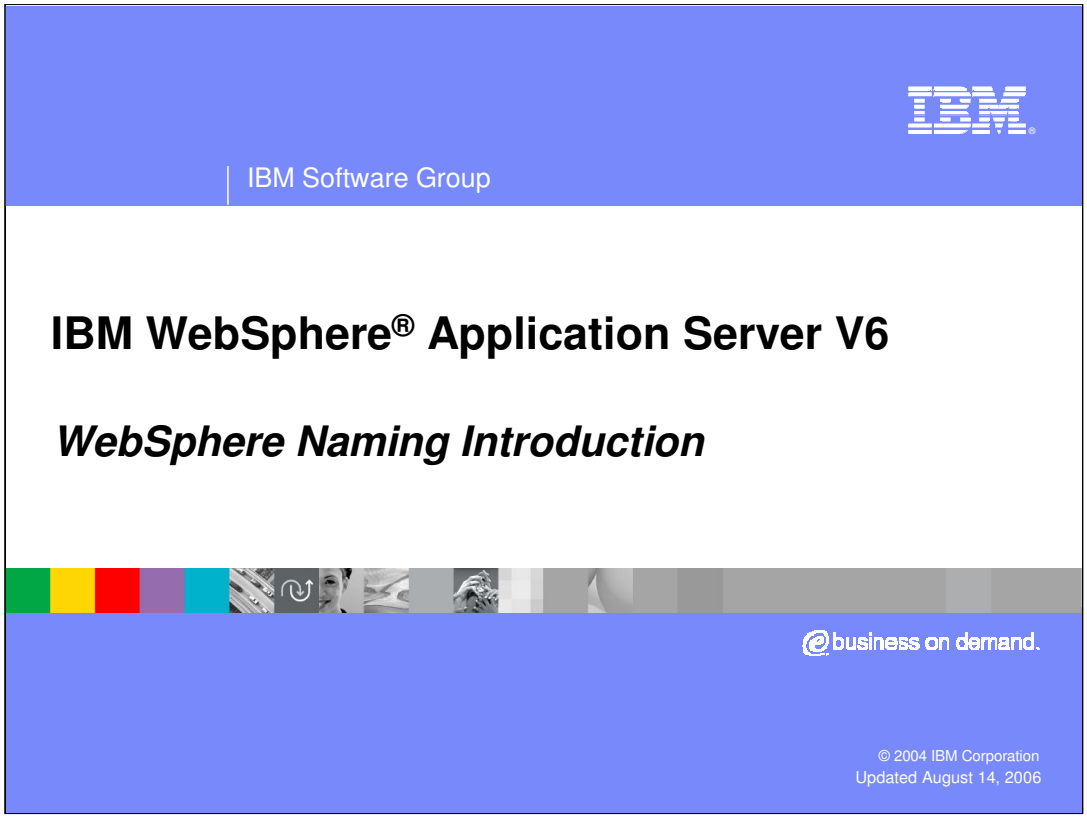
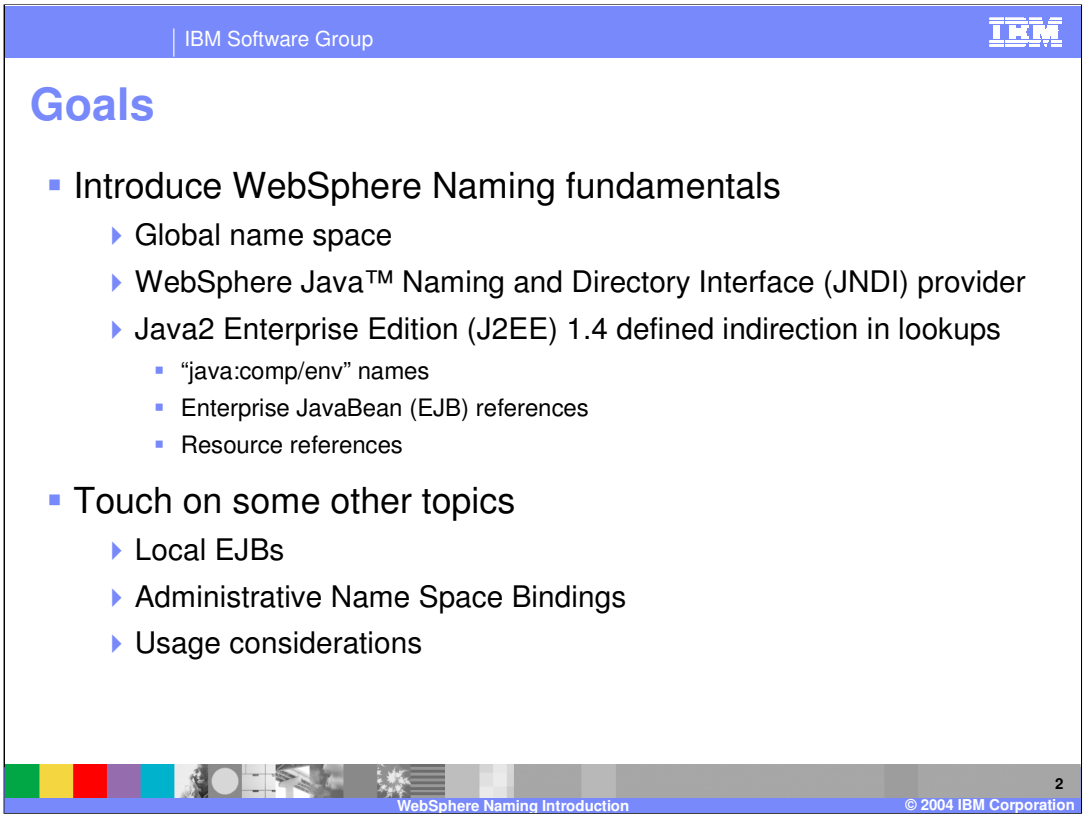


This presentation provides a high level overview of essential Naming functions in WebSphere Application Server Version 6. For those needing only a general understanding, this presentation is sufficient. For those who need a more in depth knowledge, this presentation is a prerequisite to the other presentations that cover Naming Basics, Advanced Topics, Debugging and Samples.

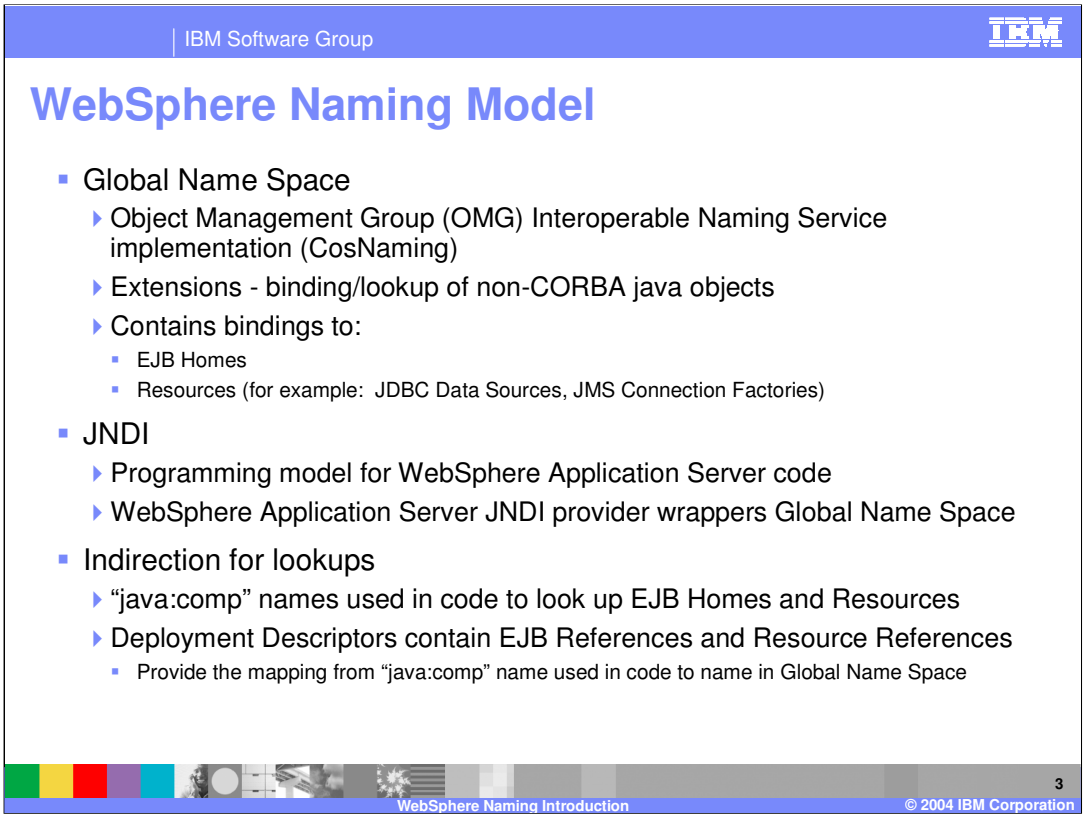


The primary purpose of Naming in a WebSphere Application Server environment is to provide a mechanism for J2EE applications to access EJB Homes and resources such as Data Sources or Java Message Service (JMS) Connection Factories. This presentation will provide a basic understanding of the components that are required to provide this functionality. These include the global name space where the EJB Homes and Resources are bound. The JNDI provider is used by clients of Naming as the programming model for accessing the name space. There are also EJB References and Resource References in Deployment Descriptors which enable a level of indirection between the special "java:comp/env" names used by the clients and the actual lookups that occur in the global name space.

This presentation then continues on to look at local EJBs, which follow a similar model as remote EJBs but without making use of the global name space.

A high level look is taken at Name Space Bindings, a WebSphere Application Server capability to add certain types of bindings to the name space through the administrative processes rather than through the use of Naming interfaces in runtime code.

Finally, an introduction is given to some of the usage considerations that apply when using Naming in a WebSphere Network Deployment environment.



The primary components used for Naming lookups of EJB Homes and Resources are the global name space, the WebSphere Application Server JNDI provider and the References in the Deployment Descriptors which are used to define a level of indirection for the lookups.

The global name space is an implementation of the Object Management Group's Interoperable Naming Service specification. This specification introduces a set of CosNaming interfaces which enable the binding and lookup of CORBA objects from NamingContext objects. EJB Homes in WebSphere Application Server are CORBA objects and therefore can be looked up from these NamingContext objects. The NamingContext objects can be put together to form a name space as a hierarchy or as a graph. Because Resources such as JMS Connection Factories are not CORBA objects, the global name space implementation has been extended to allow for the binding and lookup of serializable java objects in addition to CORBA objects.

The CosNaming interface and WebSphere extensions provided by the global name space are not used directly by user code in WebSphere. The Java Naming and Directory Interfaces, also known as JNDI, are used to access the name space. The JNDI interfaces provide a common set of interfaces which can be used to access any arbitrary Naming service or Directory Service through the use of a service-specific JNDI provider. WebSphere Application Server has a JNDI provider which has been implemented to know how to access the global name space with its combination of CosNaming interfaces and WebSphere Application Server specific extensions. The WebSphere Application Server JNDI provider, which is the user model in WebSphere Application Server for accessing the name space, uses names which are formatted as if they were CosNaming names, but can be used to access both CORBA and non-CORBA objects from the global name space.

Code that you write in a J2EE application does not look up EJB Homes or Resources using the name by which the object is bound into the global name space. Rather, the J2EE specifications describe a special kind of name to be used, which starts with the URL "java:" (java colon). EJB References and Resource References in an application module's

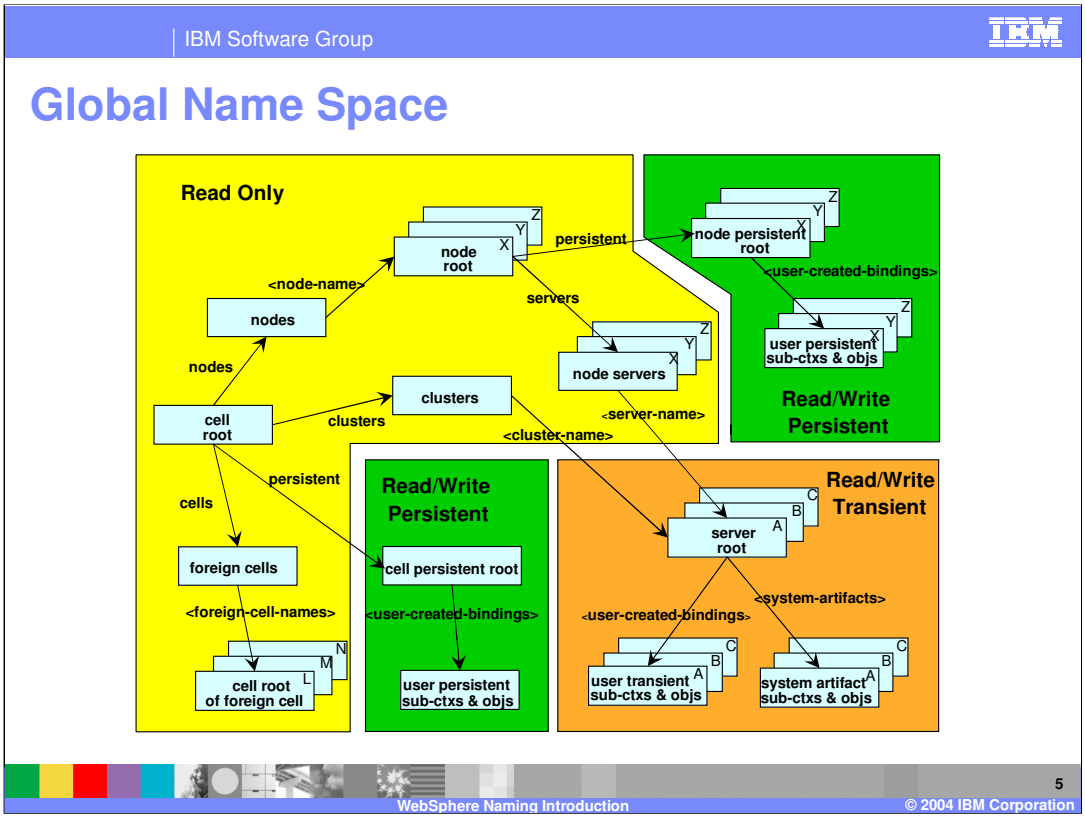
Global Name Space

- CosNaming function provided by all servers
  - Deployment Manager, Node Agents, Application Servers
  - No separate Name Server
- Single logical view of name space for entire cell
- Name space physically mapped across servers
  - Some parts replicated in all servers
  - Some parts replicated in a subset of servers
  - Some parts only in individual servers
- Name space built in memory during server startup
  - Configuration data defining cell topology
  - Resources defined in the configuration
  - EJBs contained in applications
  - Name space bindings in the configuration (administratively added by the user in the past)
  - Persistent bindings (programmatically added by the user in the past)
  - Transient bindings (programmatically added by user during startup)
- Can access the name space by connecting to any server
  - Multiple bootstrap host/port combinations available, one per server
  - Code running in a server defaults to name space in that server

This slide takes a more in depth look at the global name space, in particular how the CosNaming interfaces with WebSphere extensions are implemented within a WebSphere environment. To eliminate possible preconceived notions it should be stated that the global name space is not a typical CosNaming implementation. In many CosNaming implementations, a unique Name Server process is provided with the name space content being a reflection of a data store specifically designed to hold the name space contexts and bindings. However, this is not the case for the global name space in WebSphere Application Server.

In WebSphere Application Server, there is no separate Name Server process. Rather, every server in the environment provides the functionality for the global name space, including all application servers, the node agents and the deployment manager. From the perspective of a WebSphere Network Deployment cell, these servers work together to provide the user with a single logical view of the name space although the physical representation of the name space is not fully contained in any one server. The physical representation of the name space is partially replicated across the server processes and partially distributed across the server processes. More details on this will be provided in the subsequent slide.

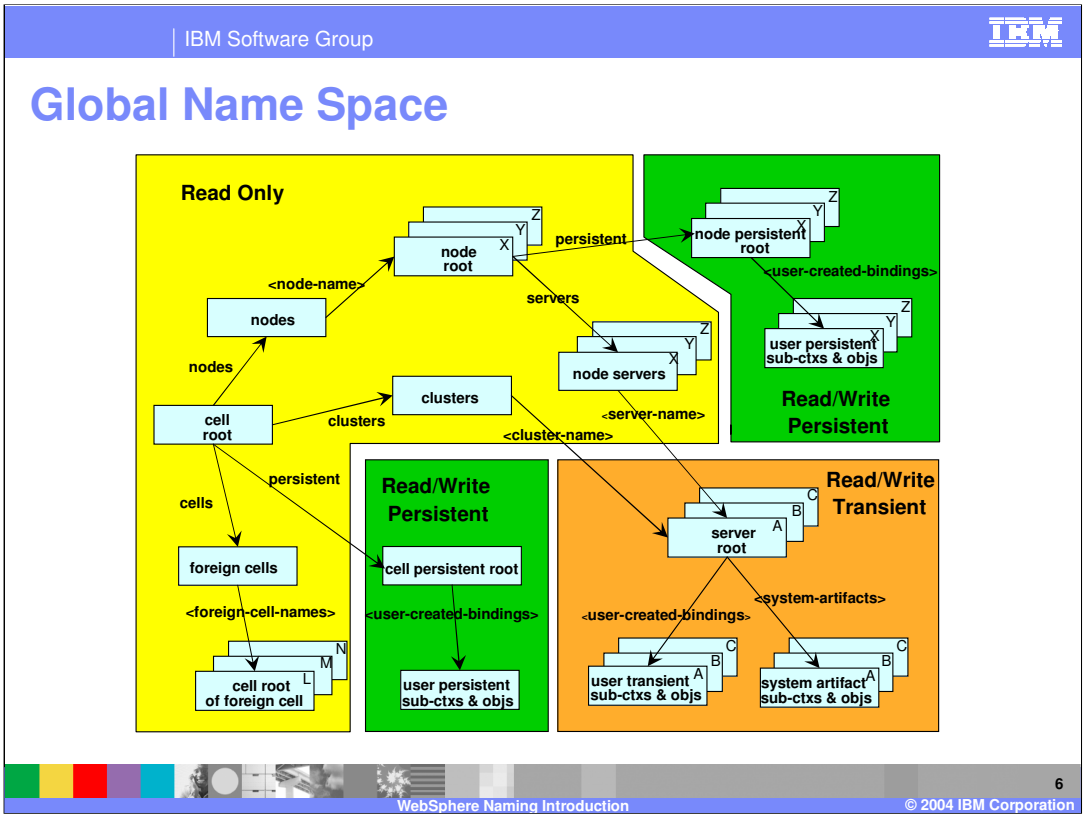
When a WebSphere Application Server starts, the subset of the logical name space which will be physically represented within that server is built in the server's memory. There are many sources of data that are used to build the name space within the server process. First is the configuration data from the system management repository which defines the topology of the cell. Also from the system management repository, all the resources which have been defined within the scope of that server are bound in. As applications are started, the EJB Homes in those applications are bound into the name space as well. The administratively configured Name Space Bindings from the system management repository are bound in, along with any persistent bindings which were added to the persistent portions of the name space prior to the server starting. Lastly, user code which is running in the server may bind objects into the transient portion of the name space. More details on the persistent and transient portions of the name space will be provided in the subsequent slide.



This is a pictorial representation of the global name space. First, this presentation will describe the logical name space structure, then talk about the characteristics of various portions of the name space and where those portions of the name space are physically distributed in the servers.

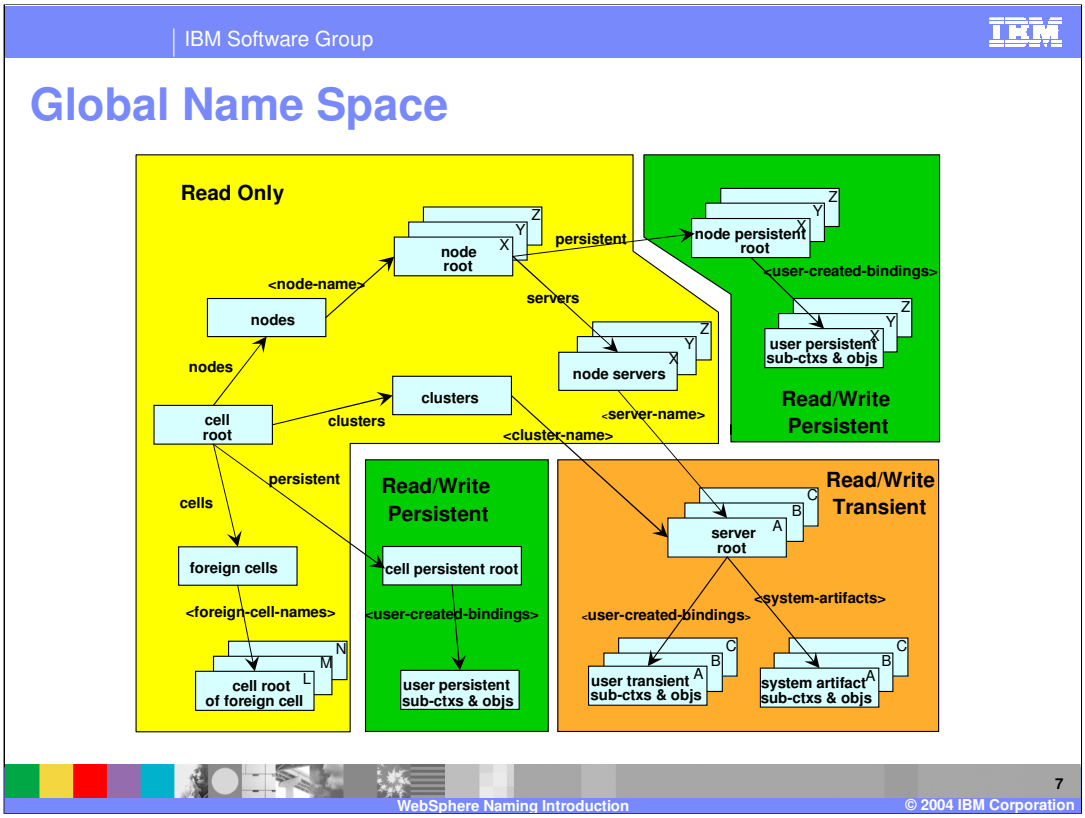
The light blue rectangles represent naming contexts in the name space. The text within a rectangle is a label which describes the purpose of the context. The arrows represent bindings in the name space and the text by an arrow is the name of the binding. This would be the actual name used when traversing the name space. The logical name space starts at the context in the center of the left-hand side of the picture, with the label “cell root”. Starting from the cell root context, the entire logical name space can be traversed. The structure of the name space is based on the topology of the WebSphere Application Server cell. There is a context for each node in the cell (the “node root” shown in the upper center). There is also a context for each server in the cell (the “server root” shown in the lower right). For example, if there was a “server1” on “nodeA”, from the cell root context the name “nodes/nodeA/servers/server1” would be used to traverse to the server root context for server1. Another example would be a cluster named “clusterX” with cluster members “x1” and “x2”. From the cell root context, a lookup of “clusters/clusterX” would be used to traverse to the server root context for one of the cluster members, either “x1” or “x2”.

Examine the portion of the name space within the yellow background. As you can see, the contexts within this area all pertain to the topology of the cell, reflecting information about the cell, its nodes, clusters and servers. This yellow part of the name space is replicated in every server in the cell and is built using configuration data from the system management repository.



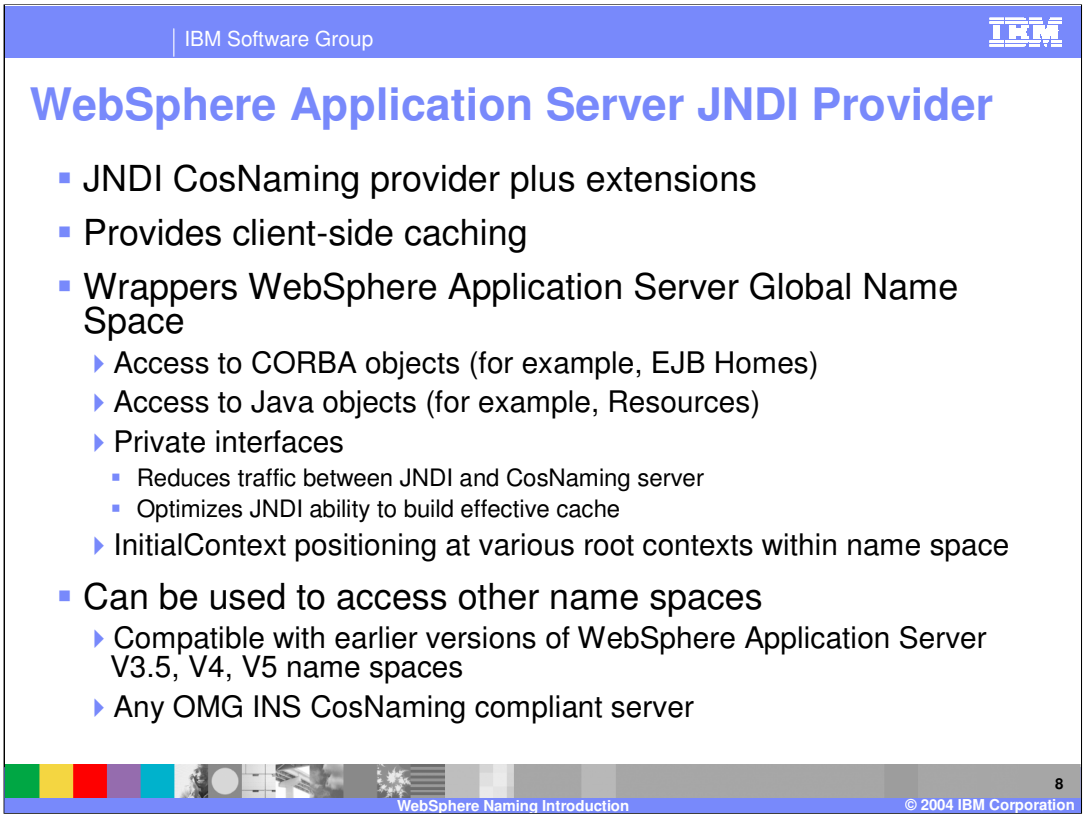
The next area to examine is the portion of the name space with the brown background in the lower right hand corner of the picture. This shows the “server root” context. There is a unique server root context for every server in the cell. The server root context is contained in the server that it represents, and it is not replicated in any other server. The server root context is read/write, so objects can be bound and unbound in this context. The context is transient, so an object bound into this context or any of its sub-contexts will not be there the next time the server starts. The context labeled “system artifact sub-ctxs & objs” refers to the EJB Homes and Resources for the server. These are bound into the server root every time the server starts, so the fact that the server root is transient is not a problem. In fact, it assures that the bindings in the server root actually represent the EJBs and Resources that are currently defined for that server and eliminates the possibility of there being stale, out of date bindings in the name space. The context labeled “user transient sub-ctxs & objs” refers to any bindings which user code added to the server root. Because the server root is transient, the only practical use of this would be where a user bound an object into this context, every time the server started, from code that ran within the server itself (for example, from a custom service).

Now notice the part of the name space with the green background in the lower center of the picture. This is the “cell persistent root”. This context is replicated in every server in the cell. This is provided as a place where users can add bindings to the name space that will be persistently maintained regardless of server restarts. The persistence is accomplished by updating an XML file in the system management repository. This then has to be pushed to all the nodes in the cell as all the servers share this context. The pushing of these updates throughout the cell involves some overhead and a timing window between when an object is bound and when that binding is reflected in all the servers. Because of this, it is recommended that this context only be used for bindings which have a long lifespan and it should not be used for bindings that are frequent and intended to represent real-time changes in the environment.



The “node persistent root” area of the name space is shown with the green background in the upper right hand portion of the picture. There is one of these per node, replicated in every server on the node. Servers on a different node have a different node root. These contexts are similar to the cell persistent root in terms of persistence and also being best for bindings with long life spans and infrequent updates.

In the picture, the traversing of a binding between contexts in different colored areas may involve two servers. For example, the “node servers” context contains a binding to the server root context for every server on that node. However, the node servers context exists on every server whereas the server root contexts are unique to the server they represent. For example, assume “nodeA” contains “server1” and “server2”. If connected to the name space for server1 at the cell root, the lookup of “nodes/nodeA/servers/server1” would be resolved all within server1. However, if connected to the name space for server2 at the cell root, the same lookup of “nodes/nodeA/servers/server1” would be resolved using both server1 and server2. From a logical view, the request started at the same logical place in the name space and looked up the identical name, but under the covers the operation handled the distributed nature of the name space.

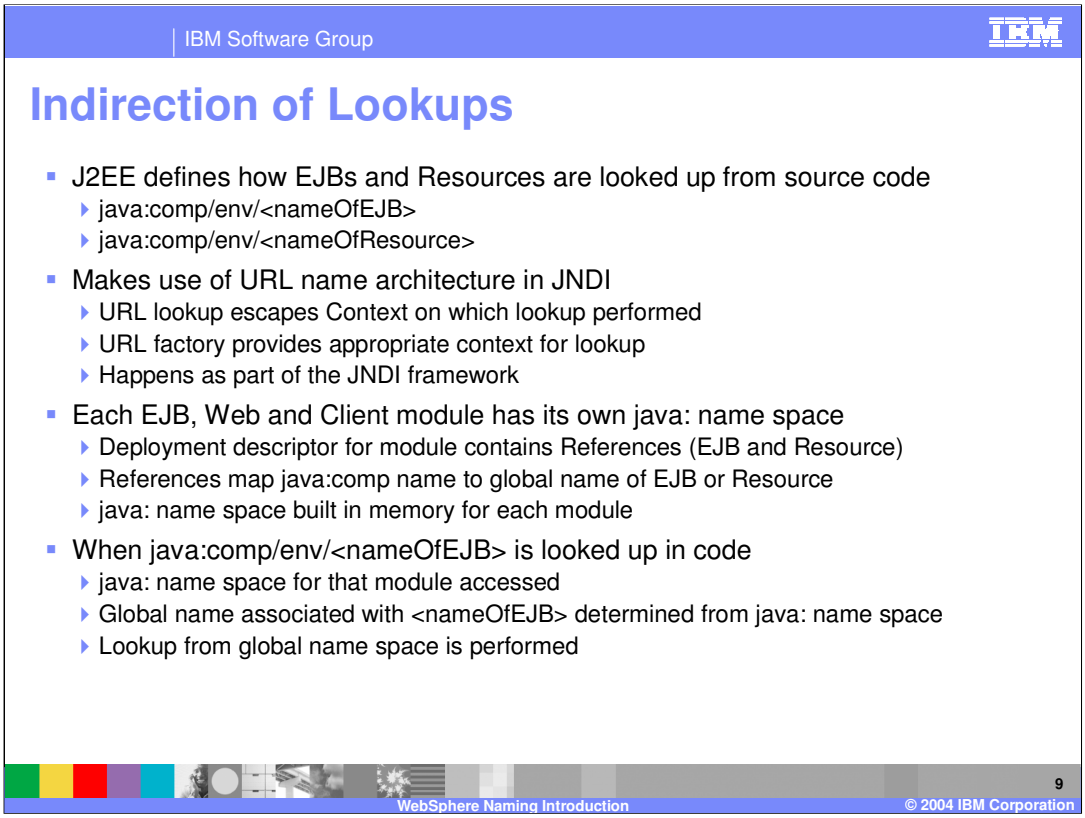


The WebSphere Application Server JNDI provider looks and behaves very much like a JNDI CosNaming provider, with the exception that it can bind and lookup non-CORBA objects as well as CORBA objects.

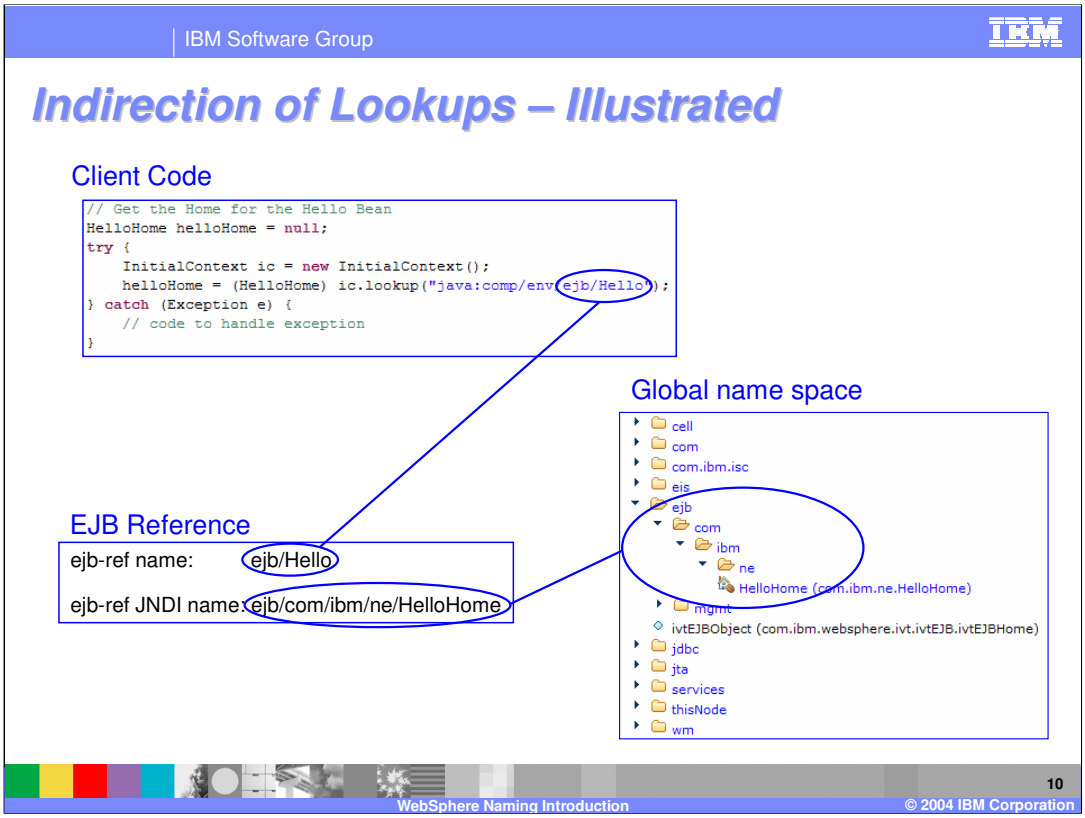
It provides a client-side caching capability which greatly improves performance when an object is looked up multiple times from within the same process. Caching is on by default, but can be turned off when not needed. There are also mechanisms to clear the cache when there is a possibility that data in the cache is out of date.

The JNDI provider is essentially a wrapper of the global name space provided by WebSphere Application Server, providing the capability to lookup EJB Homes and Resources which are bound into the global name space. There is a very tight coupling between the implementation of the global name space and the JNDI provider. This allows a high degree of optimization, reducing the traffic between the JNDI client and the server housing the global name space. It also is capable of flowing additional data on requests that enable the JNDI cache to be populated faster. The provider is also aware of the basic structure of the global name space, providing properties which allow the InitialContext to be positioned at different root contexts, for example, at the server root or the cell root.

The JNDI provider also is aware of previous versions of WebSphere Application Server and is capable of interacting with the global name space in any Application Server version from 3.5 on, including whatever optimizations exist in the global name space for those versions. In addition, any OMG-compliant Interoperable Naming Service implementation can be accessed using the WebSphere Application Server provider.



The J2EE specification defines JNDI as the mechanism for a J2EE application to access EJBs and Resources. It makes use of a mechanism available in JNDI, where a name that begins with a URL will not be processed by the naming context on which the lookup was performed, but will escape to a JNDI provider which is specific to the URL used in the name. J2EE defines the “java:” URL to be used for this. Further, J2EE specifies that each EJB and Web module will have it’s own unique environment where objects can be looked up from code in that module using the name “java:comp/env/<nameOfObject>”. It is here that the EJBs and Resources can be looked up. J2EE defines EJB References and Resource References in the Deployment Descriptor which tie the “<nameOfObject>” to an EJB or Resource. WebSphere Application Server has extended these EJB and Resource References to include a JNDI name, where that JNDI name is the name of the EJB or Reference as it is bound into the global name space. So in WebSphere Application Server, when code does a “java:comp/env/<nameOfObject>” lookup, the processing is to actually look up the object in the global name space using the JNDI name specified in the Reference in the Deployment Descriptor.

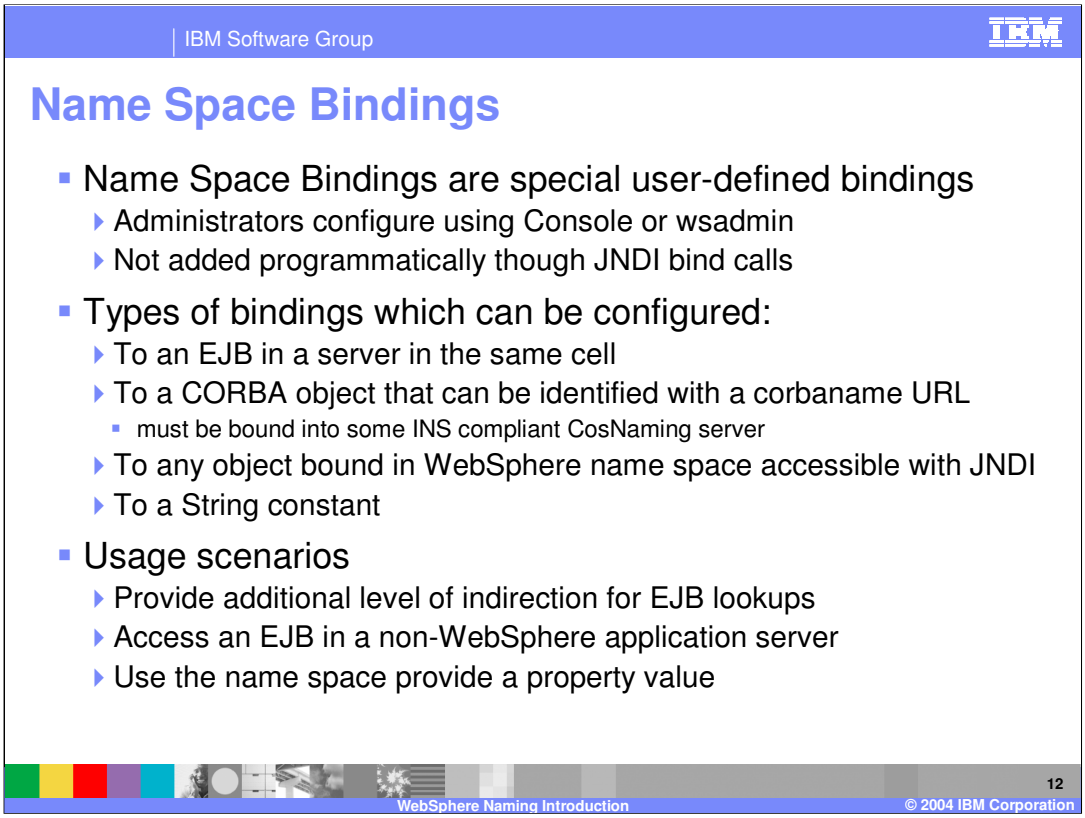


This slide illustrates what was just described on the previous slide. The client code does a lookup using the name “java:comp/env/ejb/Hello” which associates it with an EJB Reference named “ejb/Hello”. The EJB Reference contains the JNDI name “ejb/com/ibm/ne/HelloHome” which is used to lookup the EJB Home from the global name space.

Local EJBs

- EJBs can have local or remote interfaces, or both
  - Only remote EJB Homes bound in global name space
  - Local EJB Homes bound in a local: name space
- Local EJB Reference
  - Contained in Deployment Descriptor
  - Must be resolved to local EJB in same Application
- Same lookup model to access the Local EJB Home
  - java:comp/env/<nameOfEJB>
  - lookup goes indirect to the local: name space

EJBs can have local interfaces and remote interfaces. The descriptions up to this point have addressed accessing EJBs with remote interfaces. The mechanism for local interface EJBs is essentially the same with a few exceptions. The first is that the local interface EJBs are not bound into the global name space, but instead are bound into a special in-memory local: name space that the server uses just for local interface EJBs. The second difference is that rather than the Deployment Descriptor containing an EJB Reference, it contains an EJB Local Reference. This EJB Local Reference does not contain a JNDI name like the EJB Reference does. Lastly, the code using the local interface EJB and the local interface EJB itself must be packaged in the same J2EE application. It is this packaging that allows the java: name space to be built with the appropriate local: name for doing the indirect lookup of the EJB Home.



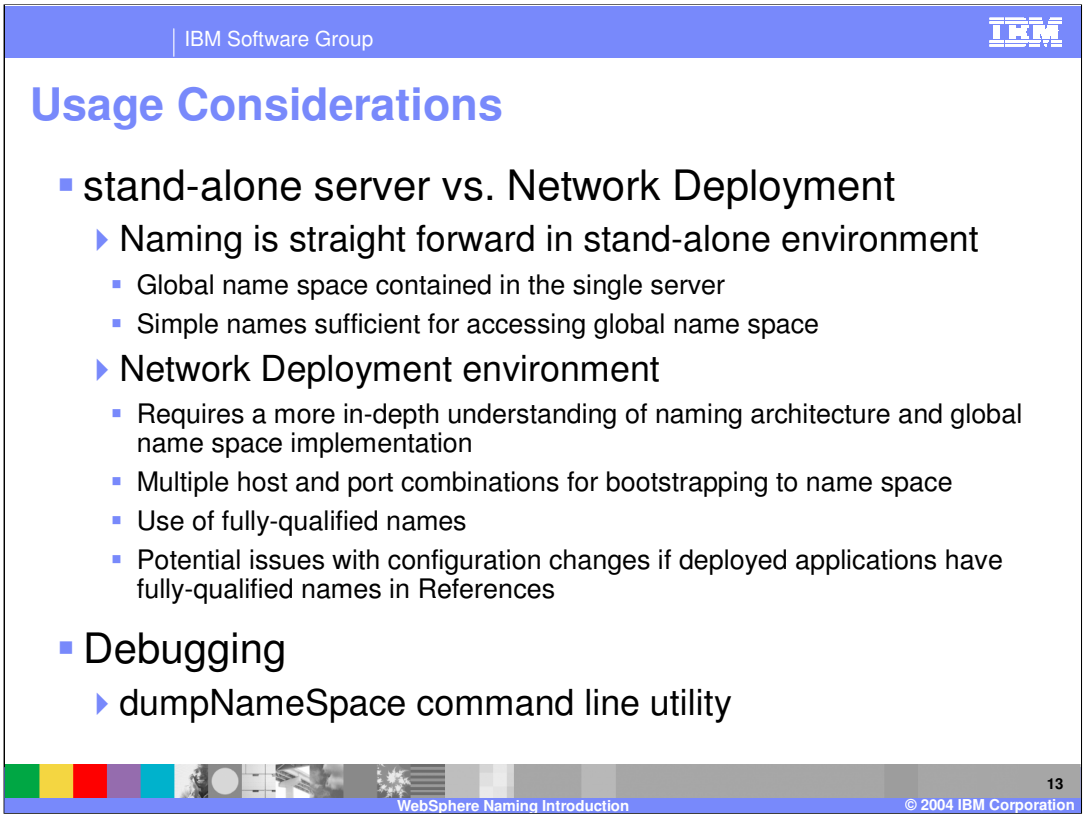
WebSphere Application Server provides a way for users to add bindings to the name space without having to write and run JNDI code to perform the bind of the object into the name space. These special bindings are called “Name Space Bindings” and they can be configured by administrators using the Administrative Console or by using wsadmin scripts.

There are different types of these Name Space Bindings with different usage scenarios for the different types of bindings.

There is an EJB binding which allows you to administratively create a binding in the name space which serves as an alias for the name of an EJB somewhere in the cell. Essentially they can be used to create a double level of indirection for looking up EJBs which makes life easier when changing the topology of the cell. In the Advanced Naming presentation there is a discussion about using these bindings as a solution to a problem associated with using fully-qualified names which contain topology information. Note that this is an important topic for anyone planning to deploy applications in a Network Deployment environment.

There is a CORBA object binding which is configured with a “corbaname” URL string. This can be helpful when wanting to access an EJB in a different WebSphere Application Server cell or from a non-WebSphere Application Server. The binding serves as an alias for that EJB in the other environment.

There is a String Constant binding. This can be used similar to property values to control processing within an application. The application can lookup some predefined name, and the value of the string bound at that name can then be examined by the application to make processing choices.



Naming in the stand-alone server environment is relatively straight forward. The entire name space is contained within the server, and looking up EJBs and Resources can be done using simple names which are relative to the server root context. The Basic Naming presentation addresses this environment.

In a Network Deployment environment, there are additional considerations because the name space is distributed across multiple servers and because look up of EJBs and Resources may have to happen across servers rather than strictly within a single server. This brings in usage of fully-qualified names and decisions about host and port for bootstrapping into the name space. These are addressed in the Advanced Naming presentation.

There is a utility called dumpNameSpace which can be an important tool for debugging naming related problems. The Debugging Naming presentation provides details on this tool.

Summary

- Described basic components of WebSphere Application Server Naming
  - Global name space
  - WebSphere Application Server JNDI provider
  - EJB and Resource References
  - java:comp/env names
  - Handling of local EJBs
  - Administrative Name Space Bindings
- Usage considerations

This presentation introduced the fundamental architecture and several of the important concepts related to Naming in WebSphere Application Server. It addressed the global name space structure and implementation, the JNDI provider used to wrapper the global name space, and the use of “java:comp/env” names and References to provide a level of indirection for lookups. It also looked at how local EJBs are handled and presented basic information about the different types of administratively created Name Space Bindings.

Template Revision: 11/02/2004 5:50 PM

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
AIX	DB2 Universal Database	Lotus	pSeries	zSeries

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

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

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

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

Linux is a registered trademark of Linus Torvalds.

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

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

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

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

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

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

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.