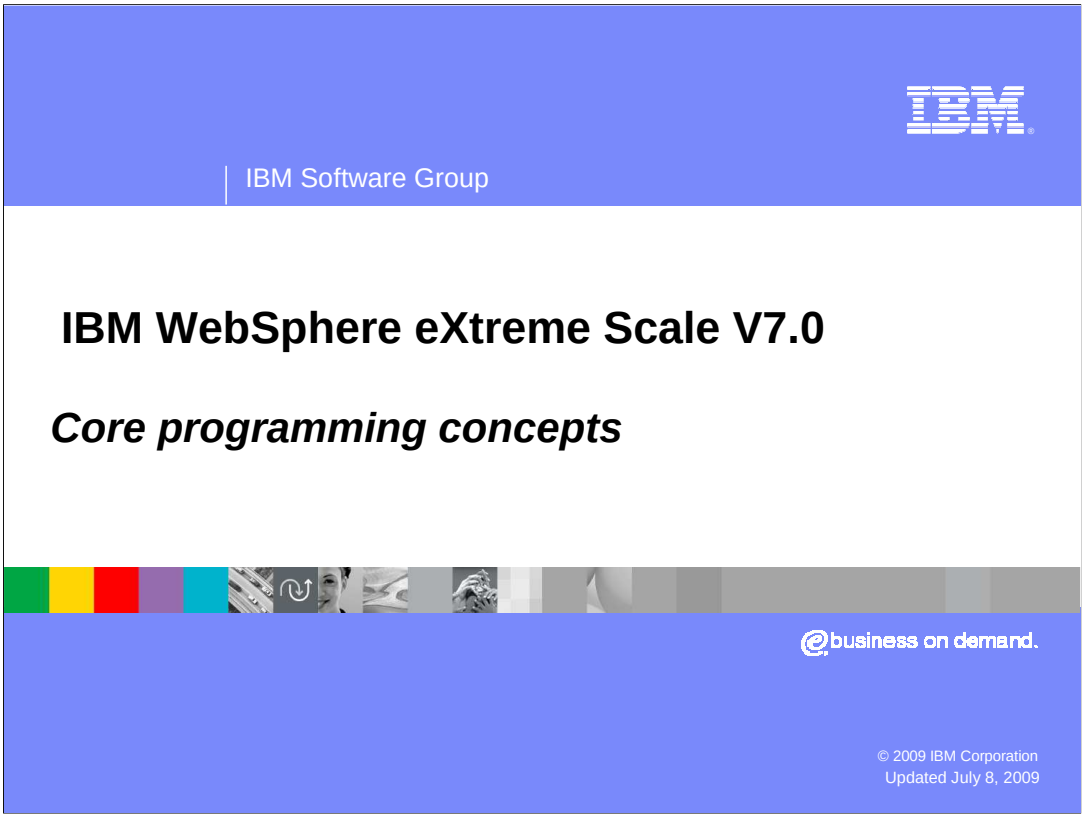


This presentation will introduce core programming concepts for WebSphere® eXtreme Scale version 7.0.

Agenda

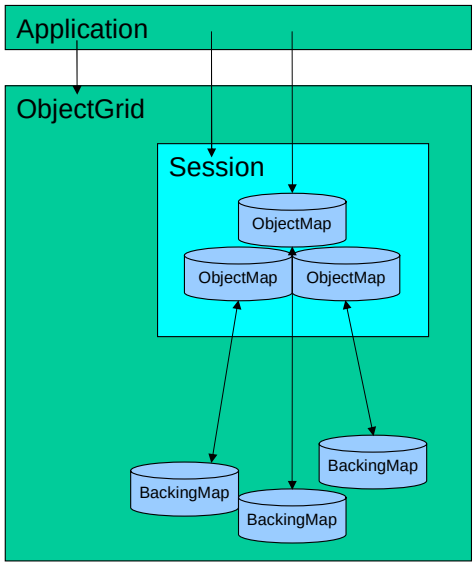
- Core programming model overview
- Accessing the grid
- Accessing the data
  - Local
  - Client-server
- Entity manager
- Managing the grid
- Embedded server support

WebSphere eXtreme Scale is a programming framework for storing and accessing data in a high performance, scalable cache. The core programming model allows you to perform fundamental interaction with the eXtreme Scale grid. This presentation will first cover basic methods for creating a grid instance or obtaining a reference to an existing grid from an eXtreme Scale grid server. Next it will cover basic data manipulation including adding data to, removing data from, and modifying data in the grid. Then it will introduce the entity manager, an alternate data storage and retrieval mechanism provided by WebSphere eXtreme Scale. Finally, the presentation will introduce several system level interfaces that are available that affect the operation of the grid, and show how you can combine grid client and server components with your own application code.

Basic programming model

Basic application flow

- Create ObjectGridManager
- Create ObjectGrid
- Define one or more BackingMaps
- Initialize the ObjectGrid
- Create a session
- Create one or more ObjectMaps
- Begin the session
- Insert objects into the ObjectMap
- Get objects from the ObjectMap
- Commit the session
- Remove (destroy) the ObjectGrid



This slide shows the primary artifacts you will deal with in an eXtreme Scale program, and provides a high-level overview of the interactions with the grid. The eXtreme Scale grid, also known as an ObjectGrid, is a container within an application that provides a runtime and programming framework for managing data stored within the container.

The backing maps contain the data managed by the ObjectGrid, and manages object life cycle and quality of service requirements. Backing maps can exist within the application or on a remote server.

The object maps provide the interface the application code uses to access data stored in the backing maps. All application interaction with the data is performed in the context of a session. The session manages the transactions contexts and manages how and when objects are copied between the backing map and object map.

The ObjectGrid is the central core of the eXtreme Scale grid framework, representing the overall grid cache. It is used for creating sessions, defining BackingMaps, managing system-level listeners and callbacks, and managing grid security settings. An application is responsible for establishing and maintaining the ObjectGrid. The ObjectGrid can be configured either programmatically or through an XML configuration file.

Program interaction with the grid includes several common steps. The application must first create an ObjectGridManager that it uses to create the ObjectGrid itself. The application can define one or more BackingMaps programmatically, or through XML configuration files. The application can then create a session and retrieve references to ObjectMaps. Through the ObjectMap, the application can create, retrieve, update, and remove objects from the grid in a transactional manner.

Creating a local grid instance

- To create an ObjectGrid using an XML configuration

```
ObjectGridManager myObjectGridManager =
  ObjectGridManagerFactory.getObjectGridManager();

ObjectGrid myObjectGrid =
  objectGridManager.createObjectGrid(objectgridNameStr,
  gridXml_URL);
```


To cache objects locally using eXtreme Scale, you must create an ObjectGrid instance within your application. The instance can be configured programmatically or created based on configuration data stored in an XML file. The code snippet shown here illustrates how to instantiate an ObjectGrid based on a configuration file, using the ObjectGridManager class. You can learn about ObjectGrid configuration files by exploring the samples provided in the product installation and in the eXtreme Scale information center.

Connecting to a grid server

- To connect to an ObjectGrid server:

```
ObjectGridManager myObjectGridManager =
  ObjectGridManagerFactory.getObjectGridManager();
clientClusterContext =
  myObjectGridManager.connect(catalogServerAddressStr,
  securityProps, overrideXmlUrl);
ObjectGrid myObjectGrid =
  myObjectGridManager.getObjectGrid(clientClusterContext,
  objectgridNameStr);
```


If your application is acting as a client accessing data from a remote eXtreme Scale server, you must create a connection to the server. This connection is controlled through a ClientClusterContext retrieved from the ObjectGridManager. When retrieving the ClientClusterContext you must specify the address of the catalog server. If you have a cluster of catalog servers you will provide a list of catalog servers. The catalog server keeps track of all of the eXtreme Scale servers within the grid configuration, and manages how data is partitioned among the servers. When you request an ObjectGrid from the ObjectGridManager, the manager communicates with the catalog server to find which servers host the requested ObjectGrid, and creates a connection to those servers.

Data access methods

- As needed
- Copy entire map
- Run logic on server (data grid)

An application interacts with the ObjectGrid the same way, whether the grid is local within the application's Java™ virtual machine or hosted in a remote eXtreme Scale server.

WebSphere eXtreme Scale provides several mechanisms for accessing data from a grid. The application can request individual objects or collections as needed. The application can request a local copy of an entire backing map. Or the application can send a command object to the servers to run within the server's Java virtual machine.

Working with eXtreme Scale data

- An ObjectGrid contains one or more ObjectMaps
  - Interface similar to java.util.Map
  - Supports all of the expected Map methods
    - put(), get()
    - Additional methods are also provided, such as insert(), update(), and not validate()
- Objects are stored as key/value pairs
  - Can be entered into the Map by the application
  - Can be loaded from an external source using custom loader objects
- Can be indexed on key or attribute values

Java objects are stored in an eXtreme Scale grid using key-value pairs within map objects called ObjectMaps. Data can be put into and retrieved from an ObjectMap using familiar “put” and “get” methods, similar to a Java map. Unlike a Java map, an ObjectMap provides transactional access to data in both local and remote maps. The map can be solely populated by the application, or it can be loaded from a back-end store by implementing a custom cache loader. The BackingMap definition can also include indexes that are automatically managed by the grid. These indexes are used by the query interface to improve performance.

Create a session

- Local workspace

```
com.ibm.websphere.objectgrid.Session mySession =
  myObjectGrid.getSession();
```

- Session contains methods to
  - Access maps
  - Manage transactions
    - Begin, Commit, and so on

Before accessing an ObjectMap, an application must first obtain a Session from the ObjectGrid object. An application obtains a Session instance through the ObjectGrid.getSession method.

An application uses a Session to begin, commit, or rollback transactions. After a transaction has started, any changes to ObjectMaps within that transaction scope are stored in the session until the transaction is committed. When a transaction is committed, the pending changes are applied to the BackingMaps and become visible to any other clients of that ObjectGrid.

ObjectGrid also supports implicit transactions, also known as auto-commit. When an ObjectMap operation is performed outside of the context of an active transaction, the session starts an implicit transaction before the operation and commits it before returning control to the application. This is not recommended as it is slower than using an explicit transaction.

A session must only be used by a single thread at a time.

Access map and data

- Access ObjectMap through Session
  - ObjectMap is local container for BackingMap

```
ObjectMap myEmpMap = mySession.getMap("employees");
```

- Map is defined in ObjectGrid configuration xml file
  - BackingMap entry
  - Defines map configuration
- Now access data
  - Data copied between BackingMap and ObjectMap

```
myEmpMap.get("000300"); // get employee with key “000300”
```


In addition to transactions, the Session also provides access to ObjectMaps through the getMap method. The ObjectMap refers to a named BackingMap defined in the ObjectGrid configuration file.

Once the application has a reference to an ObjectMap, the application can get data from the map and put data into the map.

Client scenario – ClientReplicableMap

- Provides a remote client with a full copy of a server map
- Client has local access to data
  - Server map is continuously replicated to clients asynchronously
  - Or client can retrieve a snapshot of server map
- Map can be used as a “scratchpad” to view or manipulate data locally on the client
- Can be used to offload server map when running computation over all entries in the map
- Not suitable for large maps as all data from all partitions of the server map is copied into the replica

If your client application accesses all or most of the data in a map, you can choose to copy the entire map to the client rather than requesting each piece of data individually. You can create a a static copy of the map, known as a snapshot map, at the time you make the request. Or you can request a dynamic copy of the map that is continuously replicated with data from the servers. Both types of local replica are configured using the BackingMap’s enableClientReplication method, inherited from ClientReplicableMap interface. After the replication is enabled, the server starts to replicate the map to the client. After the initial copy of the dynamic replica, the client automatically receives all updates to the server map. The client is only a few transactions behind the server at any point in time.

Data grid

- Can send a command object to run locally on the server
  - Co-located with the data
  - Can run on a single partition or all partitions
- MapGridAgent
  - Results must be aggregated by client
- ReduceGridAgent
  - Results aggregated by agent

WebSphere eXtreme Scale also provides the ability to create a custom command object that runs on the server. This can greatly improve efficiency as the processing occurs on the server, co-located with the data, and only the result is returned to the client. The command object must implement either the MapGridAgent or ReduceGridAgent interface.

A MapGridAgent is used to perform an operation on all or a specified subset of data in a server-side map. The MapGridAgent returns a Java Map holding the result for each processed key.

A ReduceGridAgent is used to process a set of entries and reduce them to a single result. The ReduceGridAgent returns a single Java Object.

An agent is submitted using an ObjectMap's AgentManager and is run on the server ObjectGrid instances that host the data in the map. The agent is routed to one or more server instances based on the partitions hosting the specified keys.

Entity manager

- Alternative mechanism for interacting with distributed cache
- Uses JPA-like APIs (Entities)
- Persist POJO's, along with any relationships it has, in a transactional manner
- Retrieve/find POJO's using the EntityManager APIs
- Keep track of changes in any attributes
- Loose coupling of data and metadata to allow heterogeneous server and client application versions

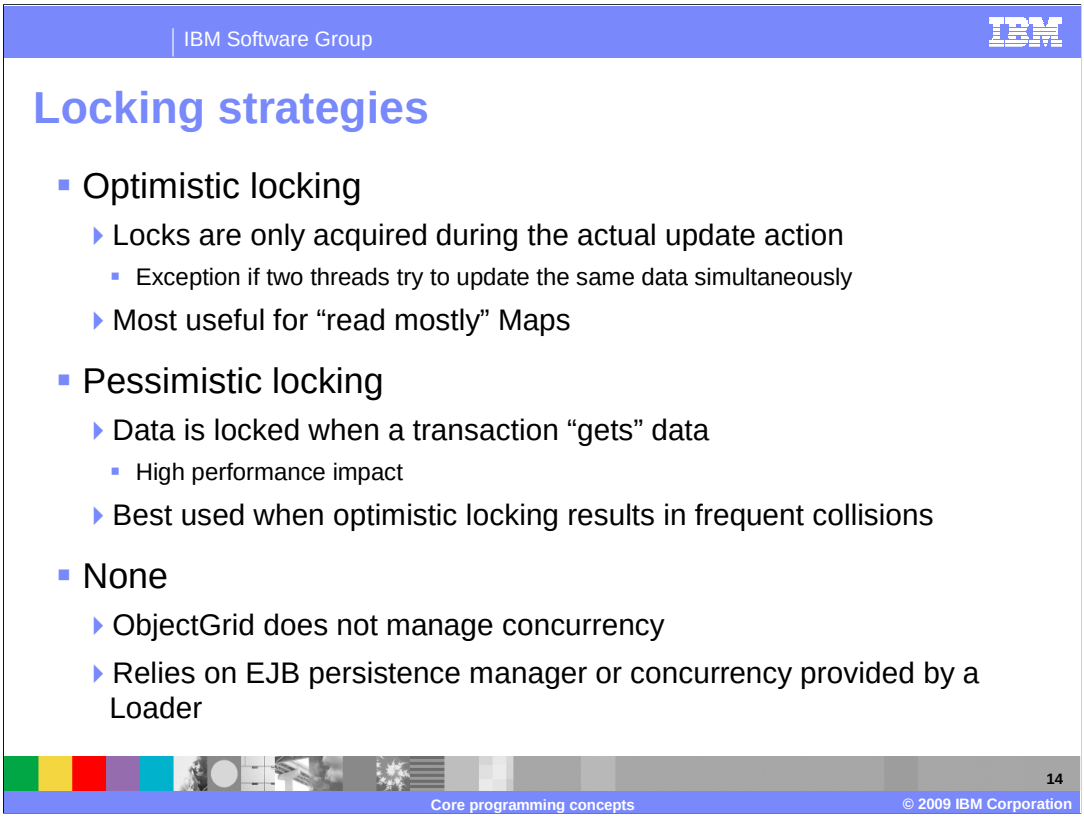
WebSphere eXtreme Scale also provides the ability to store and manage objects as JPA-like entities. Entities use two 'Tuple' objects, one for the key and another for the value. Each Tuple is an array of attributes and foreign keys. Each entity is physically mapped to a single Map in the grid. Entities can have relationships to other Entities. eXtreme Scale supports one to one, one to many, many to one and many to many relationships. These relationships are maintained automatically by the entity manager. Entities allow applications to easily manage complex object graphs that span multiple Maps.

The entity programming model is covered in more detail in a separate presentation.

Section

Managing the grid

This section will cover common interfaces used to manage object and grid life cycles.



Since the eXtreme Scale grid supports access by multiple threads within a process or multiple process in a client-server environment, the runtime provides locking mechanisms to control concurrent access to data. An ObjectGrid BackingMap supports several locking strategies to maintain cache entry consistency.

The default lock strategy is **OPTIMISTIC**. When this locking strategy is specified, locks are held in the backing map only while data is being read from the cache and copied to the local session. When the session cache is synchronized with the main cache, any objects that have been updated locally are checked against the version in the backing map. If the check fails, due perhaps to another client changing the object, then the transaction is rolled back and an exception is thrown. Optimistic locking is best used when data changes infrequently.

The **PESSIMISTIC** lock strategy acquires locks for cache entries any time a cache entry is read. This lock is conditionally held until the transaction completes. You can tune the duration locks that are held using transaction isolation levels for the session. Pessimistic locks can have a large performance impact, so should be used when data is changed frequently and frequent collisions occur with other locking strategies..

If your application does not need the grid to manage concurrency, you can specify a locking strategy of **NONE**. You might choose this strategy if the data is never updated or is only updated during quiet periods. This strategy is very fast because a lock manager is not required. The NONE lock strategy is ideal for look-up tables or read-only maps.

Copy modes

- Three different copy modes are supported
- Differing performance and data integrity traits:
  - COPY_ON_READ_AND_COMMIT
    - A copy of data is made on every read and commit action
    - Safest copy mode: thread never has direct reference to objects in map
  - COPY_ON_READ
    - Copy is not made when commit() is called: applications must not reuse objects after commit() is called to ensure data integrity
    - Better performance than COPY_ON_READ_AND_COMMIT
  - COPY_ON_WRITE
    - Minimizes copying in read-most scenarios for best performance
    - To maintain data integrity, data must be accessed through a dynamic proxy

The copy mode setting determines if and when objects are copied between the backing map and object map and given to the application code, and when objects are passed by reference. Since copying objects can be a very expensive operation, the copy mode can have a great impact on overall application performance.

COPY_ON_READ_AND_COMMIT is the default copy mode. When an application first requests an object from an ObjectMap within a transaction, the grid runtime copies the object from the BackingMap to the ObjectMap. When the transaction is committed, any objects changed by the application are copied back into the BackingMap. The application can only modify its copy of the object, ensuring that it cannot inadvertently corrupt the data cached in the BackingMap. This mode provides the best data integrity, but is the slowest, because it makes a copy of the object every time a read or commit is performed.

The **COPY_ON_READ** mode provides better performance, because data is not copied when a commit operation is performed. As with COPY_ON_READ_AND_COMMIT, a copy of the object is created the first time the object is requested within a transaction. When the transaction commits, the object in the BackingMap is replaced with a reference to the copy.

To ensure data integrity, the application must destroy object references after a transaction is committed. If the application does not destroy the committed map entry references, it can directly access data in the BackingMap, causing the cached data to become corrupted.

The **COPY_ON_WRITE** mode improves performance over the COPY_ON_READ mode by eliminating the copy that occurs when an object is first requested within transaction. Instead, the grid returns a proxy to the value. The proxy ensures that a copy of the value is not made unless the application calls a set method on the value interface. When a transaction commits, the BackingMap examines the proxy to determine if a copy was made as a result of a set method being called. If a copy was made, then the reference to that copy is stored in the BackingMap as with COPY_ON_READ. This mode provides the best performance in read-most scenarios (which are very common) while also ensuring data integrity.

Evicting objects from the map

- Cache size is controlled by evicting objects when space is needed
- An Evictor is an extensible object type for creating custom eviction schemes
- Some Evictors are provided
  - TTL (time to live) eviction is built into ObjectGrid
    - Can be based on creation time or last used time
  - Plug-in Evictors:
    - LRU (least recently used)
    - LFU (least frequently used)

Cache size control is also customizable. ObjectGrid calls Evictors when the cache reaches a certain size in order to make room for new objects. WebSphere eXtreme Scale provides predefined Evictor classes that can remove objects from the cache using least-frequently used or least-recently used policies. You can also write your own evictor class to manage the cache size based on custom criteria.

Section

Embedded server support

WebSphere eXtreme Scale provides scripts to start stand-alone catalog and grid servers. It also provides mechanisms to allow any Java virtual machine to host the eXtreme Scale runtime.

Embedded server support

- ServerFactory.getInstance
  - Allows any JVM to host the ObjectGrid server run time
  - Returns the server singleton and starts the server runtime
  - You must set the server properties before you issue the getInstance method
    - Otherwise, any modifications that you make after you issue the method do not affect the run time
- ServerFactory.getServerProperties
  - Retrieve properties to configure the ObjectGrid server instance
  - All properties set before the first invocation of getInstance are used to initialize the server

The ServerFactory getInstance method returns an eXtreme Scale server singleton and starts the server run time. This allows any JVM to act as an eXtreme Scale server. Since the getInstance method both instantiates and starts the server, you must set the servers properties before you call getInstance. Modifications that you make after you issue the method do not affect the run time.

You can set several properties to configure the eXtreme Scale server instance, such as server name, catalog server bootstrap host and port, and ORB listener host and port. Retrieve the server properties singleton with the ServerFactory.getServerProperties method. All properties set at the first invocation of getInstance are used to initialize the server.

Embedded catalog server

- CatalogServerProperties.setCatalogServer
  - Indicates JVM can host the catalog service
  - Instructs ObjectGrid server runtime to instantiate the catalog service when the server is started.

```
CatalogServerProperties catalogServerProperties =
  ServerFactory.getCatalogProperties();
catalogServerProperties.setCatalogServer(true);
Server server = ServerFactory.getInstance();
```


Any Java virtual machine can also host a catalog server by retrieving the catalog server properties singleton and setting setCatalogServer property to “true”. This indicates to the eXtreme Scale server run time to instantiate the catalog service when the server is started.

Embedding the eXtreme Scale grid container

- Server.createContainer
  - Instantiates an grid container capable of hosting shards for the ObjectGrids specified in the policy
  - Allows JVM to host multiple grid containers

```
Server server = ServerFactory.getInstance();
DeploymentPolicy policy =
  DeploymentPolicyFactory.createDeploymentPolicy(
       deploymentPolicyXML_URL, objectGridXML_URL);
Container container = server.createContainer(policy);
```


Once you have started the embedded server instance, you can create an eXtreme Scale grid container by calling the server's createContainer method. The grid container holds the BackingMaps defined in the ObjectGrid XML configuration file.

Self-contained server process

- Start all ObjectGrid services in same JVM
  - Catalog
  - Grid containers
  - Client connection logic
- Useful for development; practical in production.

```
CatalogServerProperties catalogServerProperties =
  ServerFactory.getCatalogProperties();
catalogServerProperties.setCatalogServer(true);
Server server = ServerFactory.getInstance();
DeploymentPolicy policy =
  DeploymentPolicyFactory.createDeploymentPolicy(
        deploymentPolicyXML_URL, objectGridXML_URL);
Container container = server.createContainer(policy);
```


You can start all the services together, which is useful for development and practical in production. By starting the services together, a single process starts the catalog service, some set of grid containers, and runs client connection logic. This provides an easy way to sort out programming issues before deploying in a distributed environment or to create a highly customized production environment.

Summary

- Accessing the grid
  - Local: ObjectGridManager.createObjectGrid
  - Remote: ObjectGridManager.connect
- Simple data access
  - Local
  - Client-server
- Grid management interfaces
- Embedded server support

WebSphere eXtreme Scale is a programming framework for storing and accessing data in a high performance, scalable cache. The core programming model allows you to perform fundamental interaction with the eXtreme Scale grid. Java objects are stored in an eXtreme Scale grid using key-value pairs within map objects called ObjectMaps. Data can be put into and retrieved from an ObjectMap using familiar “put” and “get” methods, similar to a Java map. These methods are the same whether the grid is local to the application’s Java virtual machine, or spread across a cluster of eXtreme Scale servers. eXtreme Scale also provides several system interfaces allowing fine grained control over object and cache life cycles. Finally, interfaces are provided that allow you to combine grid client and server components with your own application code.

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_WXS70_ProgrammingConcepts.ppt

This module is also available in PDF format at: ../WXS70_ProgrammingConcepts.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

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

EJB, Java, JVM, 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.

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.