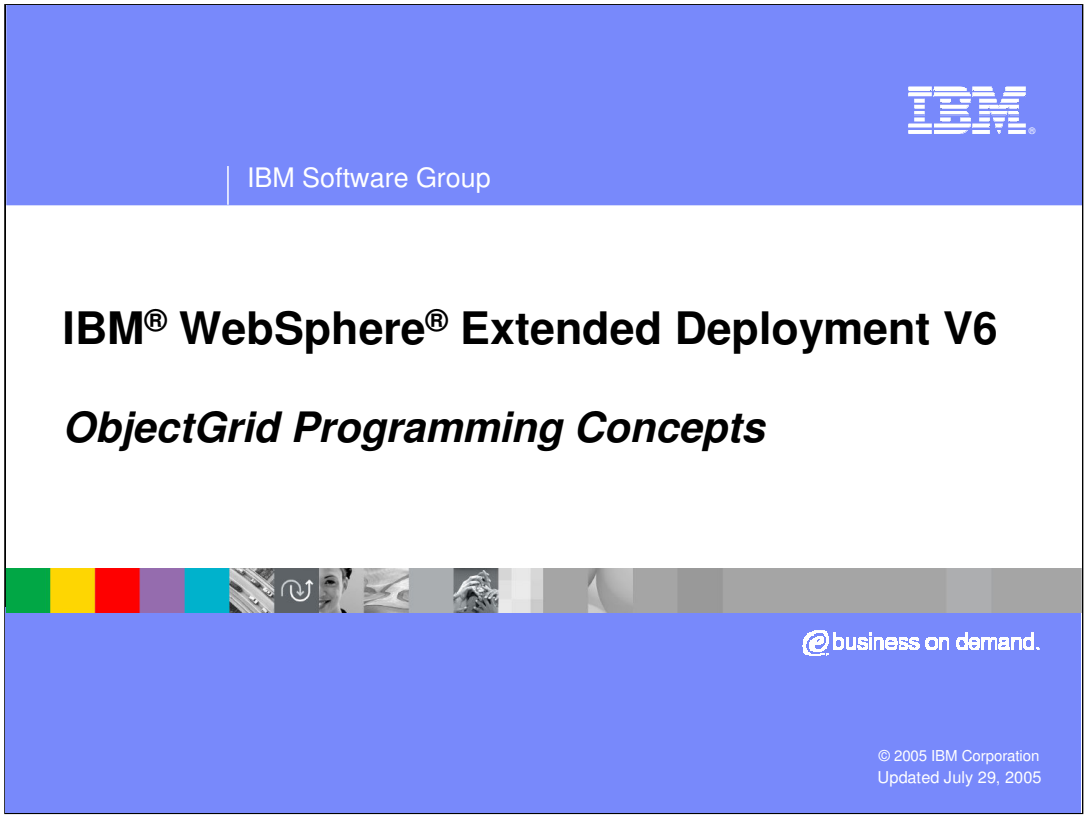


This presentation will cover basic ObjectGrid programming concepts.

Agenda

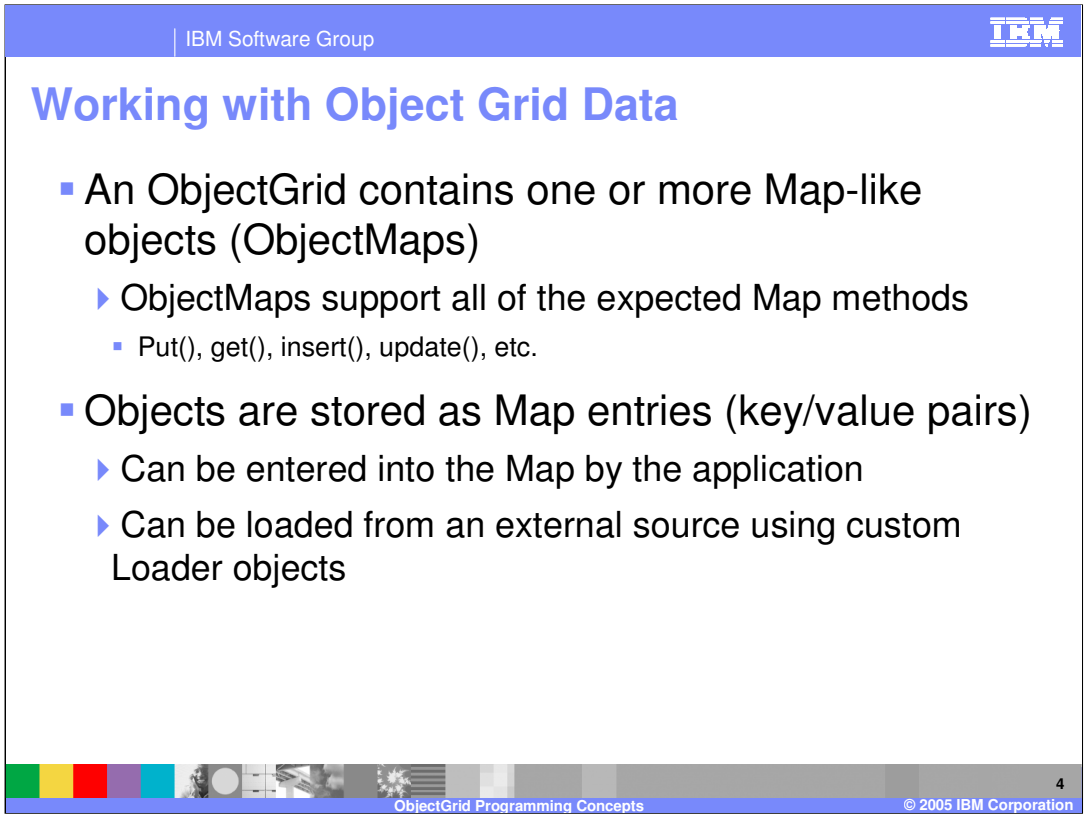
- Creating an ObjectGrid
- Working with an ObjectGrid

This presentation will first cover creating a simple ObjectGrid instance, followed by some of the extensible object types and ObjectGrid configuration options.

Section

Creating an ObjectGrid

This section will cover creating a simple ObjectGrid instance.



Java™ Objects are stored in an ObjectGrid using key-value pairs within Map objects called ObjectMaps. Data can be put into and retrieved from an ObjectMap using all of the usual Map-like methods, within the scope of a transaction. 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.

Instantiating an ObjectGrid

- ObjectGrid instances can be configured programmatically or using Extensible Markup Language (XML) files
  - Sample configuration files are provided with WebSphere XD installation
  - XML file defines the Java implementations that should be used and how they are associated
- com.ibm.websphere.objectgrid.* package contains classes needed to use ObjectGrid
- To create an ObjectGrid using an XML file:

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

ObjectGrid myObjectGrid = objectGridManager.createObjectGrid("newGrid",
  "newgrid.xml");
```


To cache objects using ObjectGrid, 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 “optionalLibraries” directory after installing WebSphere XD.

Section

Working with an ObjectGrid

This section will cover the basics of working with an ObjectGrid.

ObjectGrid Sessions

- ObjectGrid operations can be performed within the scope of a one-phase commit transaction using an ObjectGrid session object
- Simple example:

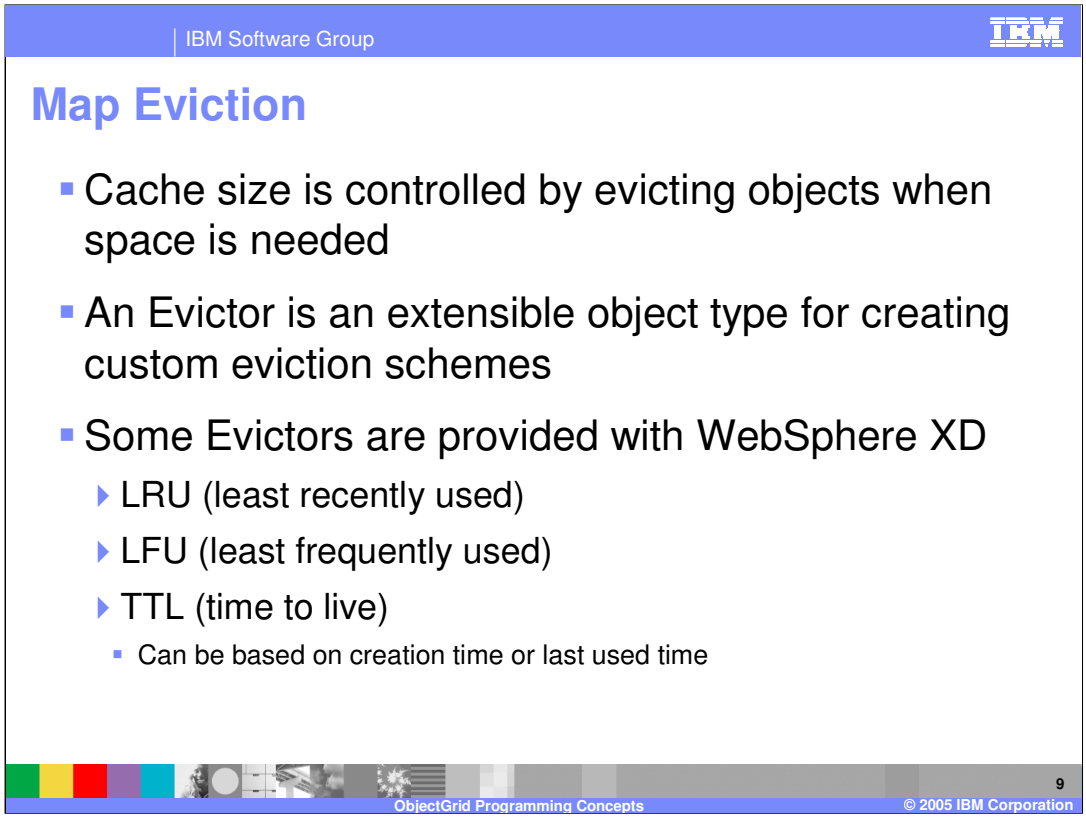
```
BackingMap m = myObjectGrid.defineMap("testMap");
Session s = myObjectGrid.getSession();
ObjectMap testMap = s.getMap("testMap");
s.begin();
testMap.insert("Joe Employee", employeeRecord);
s.commit();
```


ObjectGrid supports accessing ObjectMaps within the scope of a transaction. To do so, get a session object from the ObjectGrid, then get access to the ObjectMap within the context of the session, as shown here. You can then perform actions against the map in between calls to the “begin” and “commit” methods on the session. This basic example puts an employeeRecord object into the map using the string “Joe Employee” as a key. In this case the key is a simple string, but keys can be any type of object.

Loaders

- A Loader is an extensible object type used for associating an ObjectMap with a backing data store
- When requested data is not in the Map, the request is passed to the data store using the Loader
  - Maps can also be configured to preload data
- A Map also uses the Loader class to persist data back into the data store
  - An explicit call to flush() pushes the data to the data store, but does not commit a transaction

The cache loader interface allows you to implement a custom Java class to load cache data from a back-end data store, and also to persist changed values back to the hardened store, independent of the state of a transaction. Data can optionally be preloaded from the data store at server startup.



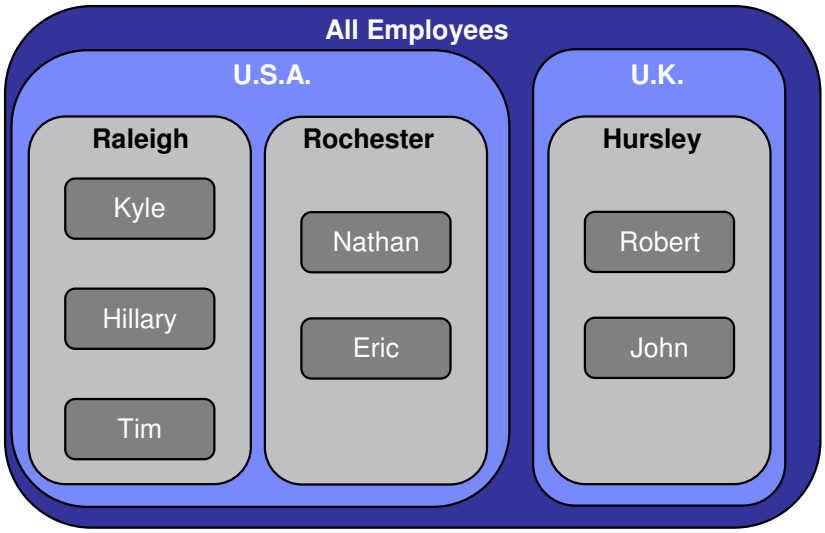
Cache size control is also customizable. WebSphere XD provides Evictor classes that can remove objects from the cache using least-frequently used or least-recently used policies, when the cache reaches a certain size, and also a time-to-live based evictor for invalidating entries that have existed for longer than a set period of time. You can also write your own evictor class, to manage the cache size based on custom criteria.

Keyword Based Eviction

- Map entries (key, value pairs) can be associated with one or more keywords
  - Simple mechanism for grouping entries
- Keywords can be associated with other keywords
  - Implicitly associates map entries with the associated keyword
  - Useful for creating nested groupings
- Application can evict all entries associated with a keyword
  - Includes nested keywords

Objects stored in an ObjectMap can be associated with one or more keywords, enabling simple grouping of entries within the cache. Keywords can also be associated with other keywords to create nested groups. These groups are useful because you can invalidate objects in the cache based on keyword. You can choose to invalidate all items associated with a particular keyword, which includes all nested keywords as well.

Nested Keyword Example



As an example, you might be storing employee records in an ObjectMap. You could group all of the employees according to the site at which they work, and then group the sites by state or country (or both). In the example shown here, choosing to invalidate all records associated with the keyword “U.S.A.” would invalidate five employees, because all of the employees at both the Raleigh and Rochester sites would be invalidated.

Locking Strategies

- Optimistic locking
  - Locks are only acquired during the actual update action
    - Will throw an exception if two threads try to update the same data simultaneously
  - Most useful for “read mostly” Maps
- Pessimistic locking
  - Data is locked when a transaction “gets” data
    - High performance impact
  - Best used when optimistic locking results in frequent collisions
- None
  - ObjectGrid does not manage concurrency
  - Relies on EJB persistence manager or concurrency provided by a Loader

Like most cache frameworks, you have a choice of data locking strategies when working with ObjectGrid. Optimistic locking is the most common, and only acquires exclusive locks when writing data to the map. If two threads try to get the same lock simultaneously, an exception will be thrown, and they will have to try again. For maps that only update data occasionally, this is the preferred mode. Pessimistic locking, on the other hand, assumes that data may be updated each time it is accessed. This means that data is locked for updating each time it is accessed. While this method is generally slower, it can be better than optimistic locking for write-heavy applications that generate collisions frequently. You can also choose to have ObjectGrid ignore concurrency issues entirely, and rely on a custom Loader or the EJB persistence manager for concurrency.

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

A copy mode is the setting that determines if and when copies of objects are made and given to the application code, as opposed to when objects are passed by reference. The ‘COPY_ON_READ_AND_COMMIT’ 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, ensuring that the worker thread never has a direct reference to an object in the map. The ‘COPY_ON_READ’ mode provides better performance, because data is not copies when the a commit operation is performed. To ensure data integrity, this mode requires the application to guarantee that objects will not be reused after a transaction is committed. The ‘COPY_ON_WRITE’ mode requires an application to access objects indirectly using a Java dynamic proxy, but provides the best performance in read-most scenarios (which are very common) and also ensures data integrity.

Further Exploration

- ObjectGrid samples
  - optionalLibraries/ObjectGrid/objectGridSamples.jar
  - optionalLibraries/ObjectGrid/SamplesGuide.htm
- ObjectGrid Javadoc
  - web/xd/apidocs
- ObjectGrid programming guide
  - Not yet available, but will be posted to the WebSphere XD documentation library
    - http://www.ibm.com/software/webservers/appserv/extend/library/

To continue learning about writing ObjectGrid applications you should explore the ObjectGrid samples, which can be found in the “optionalLibraries” directory after installing WebSphere XD. The included Samples Guide will walk you through the provided sample code, which is well documented. Javadoc is also provided for the ObjectGrid interfaces in the “web” directory after installing WebSphere XD. Though it is not yet available, a comprehensive ObjectGrid programming guide will be published to the WebSphere XD Web site in the near future.

Summary

- ObjectGrid instances are created programmatically and can be configured programmatically or using XML files
- ObjectMaps can be used like a standard map, with transaction support
- ObjectGrid features can be customized by implementing custom Java classes
  - Cache loading, invalidation, and more are extensible

In summary, ObjectGrid instances are created within your application code, optionally based on XML configuration files. ObjectMaps are used to hold cached objects, and work like a standard map, with the added benefit of transaction support. Many ObjectGrid features can be customized; this presentation only scratches the surface by introducing some of the more commonly customized components. For more information, consult the resources cited on the previous slide.

Template Revision: 3/09/2005 9:40 AM

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.