

IBM WebSphere eXtreme Scale
Version 8.6

Product Overview
June 2013



8.6 This edition applies to version 8, release 6, of WebSphere eXtreme Scale and to all subsequent releases and modifications until otherwise indicated in new editions.

© **Copyright IBM Corporation 2009, 2013.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures v

Tables vii

About the *Product Overview* ix

Chapter 1. Product overview 1

WebSphere eXtreme Scale overview	1
What's new in Version 8.6	4
Release notes	7
Notices	8
Privacy policy considerations	10
Hardware and software requirements	12
Directory conventions	13
WebSphere eXtreme Scale technical overview	15
Caching overview	16
Caching architecture: Maps, containers, clients, and catalogs	16
Enterprise data grid overview	22
IBM eXtremeMemory	25
Zones	26
Evictors	30
OSGi framework overview	33
Cache integration overview	34
Liberty profile	34
JPA level 2 (L2) cache plug-in	38
HTTP session management	45
Dynamic cache provider overview	47
Database integration: Write-behind, in-line, and side caching	54
Sparse and complete cache	55
Side cache	56
In-line cache	56
Write-behind caching	59
Loaders	61
Data pre-loading and warm-up	62
Database synchronization techniques	64
Data invalidation	65
Indexing	67
JPA Loaders	69
Serialization overview	71
Serialization using Java	73
ObjectTransformer plug-in	74
Serialization using the DataSerializer plug-ins	78
Scalability overview	79
Data grids, partitions, and shards	79
Partitioning	81
Placement and partitions	83
Single-partition and cross-data-grid transactions	86
Scaling in units or pods	92
Availability overview	93
High availability	93
Replicas and shards	106
Transaction processing overview	116
Transactions	117

CopyMode attribute	120
Locking strategies	122
Lock types	126
Deadlocks	128
Data access and transactions	131
Transaction isolation	133
Single-partition and cross-data-grid transactions	134
JMS for distributed transaction changes	140
Two-phase commit and error recovery	141
Security overview	143
REST data services overview	144

Chapter 2. Planning. 147

Planning the topology	147
Local in-memory cache	147
Peer-replicated local cache	149
Embedded cache	151
Distributed cache	152
Database integration: Write-behind, in-line, and side caching	154
Planning multiple data center topologies	169
Interoperability with other products	180

Chapter 3. Scenarios 183

Scenario: Configuring an enterprise data grid	183
Enterprise data grid overview	183
Configuring IBM eXtremeIO (XIO)	184
Configuring data grids to use eXtreme data format (XDF)	186
Developing enterprise data grid applications	187
Starting stand-alone servers (XIO)	192
Tuning IBM eXtremeIO (XIO)	192
Scenario: Securing your data grid in eXtreme Scale	193
Authenticating eXtreme Scale connections between servers	194
Authenticating requests from clients to servers	197
Authorizing access to the data grid	202
Authorizing access for special administrative operations	206
Securing data that flows between eXtreme Scale clients and servers with SSL encryption	209
Storing security artifacts for authorized users	214
Scenario: Using an OSGi environment to develop and run eXtreme Scale plug-ins	217
OSGi framework overview	217
Installing the Eclipse Equinox OSGi framework with Eclipse Gemini for clients and servers	219
Running eXtreme Scale containers with non-dynamic plug-ins in an OSGi environment	222
Administering eXtreme Scale servers and applications in an OSGi environment	223
Building and running eXtreme Scale dynamic plug-ins for use in an OSGi environment	224
Running eXtreme Scale containers with dynamic plug-ins in an OSGi environment	232

Scenario: Using JCA to connect transactional applications to eXtreme Scale clients	240
Transaction processing in Java EE applications	241
Installing an eXtreme Scale resource adapter	243
Configuring eXtreme Scale connection factories	245
Configuring Eclipse environments to use eXtreme Scale connection factories	246
Configuring applications to connect with eXtreme Scale	247
Securing J2C client connections	248
Developing eXtreme Scale client components to use transactions	249
Administering J2C client connections	254
Scenario: Configuring HTTP session failover in the Liberty profile	254
Enabling the eXtreme Scale web feature in the Liberty profile	255
Enabling the eXtreme Scale webGrid feature in the Liberty profile	255
Enabling the eXtreme Scale webApp feature in the Liberty profile	256
Configuring a web server plug-in to forward requests to multiple servers in the Liberty profile	257
Merging plug-in configuration files for deployment to the application server plug-in	258
Scenario: Running grid servers in the Liberty profile using Eclipse tools	259
Installing the Liberty profile developer tools for WebSphere eXtreme Scale	259
Setting up your development environment within Eclipse	260
Migrating a WebSphere Application Server memory-to-memory replication or database session to use WebSphere eXtreme Scale session management	262

Taking note of previous configuration settings in WebSphere Application Server administrative console	263
Creating the catalog service domain for WebSphere eXtreme Scale session management	264
Configuring WebSphere eXtreme Scale to use your previous configuration settings.	265
Scenario: Using WebSphere eXtreme Scale as a dynamic cache provider	267
Dynamic cache provider overview	267
Planning environment capacity	274
Configuring an Enterprise Data Grid in a stand-alone environment for dynamic caching	274
Configuring an Enterprise Data Grid for dynamic caching using a Liberty profile	277
Configuring dynamic cache instances	280

Chapter 4. Samples	281
Free trial	284
Sample properties files	284
Sample: xsadmin utility	284
Creating a configuration profile for the xsadmin utility	287
xsadmin utility reference	288
Verbose option for the xsadmin utility	292

Notices	295
--------------------------	------------

Trademarks	297
-----------------------------	------------

Index	299
------------------------	------------

Figures

1. High-level topology	2	38. The container for the primary shard fails	113
2. Catalog service	17	39. The synchronous replica shard on ObjectGrid container 2 becomes the primary shard	113
3. Container server	18	40. Machine B contains the primary shard. Depending on how automatic repair mode is set and the availability of the containers, a new synchronous replica shard might or might not be placed on a machine.	114
4. Partition	19	41. Microsoft WCF Data Services	145
5. Shard	19	42. WebSphere eXtreme Scale REST data service	145
6. ObjectGrid	20	43. Local in-memory cache scenario	148
7. Map	20	44. Peer-replicated cache with changes that are propagated with JMS	149
8. Map sets	20	45. Peer-replicated cache with changes that are propagated with the high availability manager	150
9. Possible topologies	22	46. Embedded cache	151
10. Enterprise data grid high-level overview	23	47. Distributed cache	153
11. Enterprise data grid object update flow	23	48. Near cache	153
12. Comparison of eXtremeMemory and Heap Storage Response Times	26	49. ObjectGrid as a database buffer	155
13. Primaries and replicas in zones	27	50. ObjectGrid as a side cache	155
14. JPA intra-domain topology	40	51. Side cache	156
15. JPA embedded topology	41	52. In-line cache	157
16. JPA embedded, partitioned topology	42	53. Read-through caching	158
17. JPA remote topology	44	54. Write-through caching	158
18. HTTP session management topology with a remote container configuration	46	55. Write-behind caching	159
19. ObjectGrid as a database buffer	54	56. Write-behind caching	160
20. ObjectGrid as a side cache	55	57. Loader	161
21. Side cache	56	58. Loader plug-in	163
22. In-line cache	57	59. Client loader	164
23. Read-through caching	58	60. Periodic refresh	165
24. Write-through caching	58	61. Enterprise data grid high-level overview	183
25. Write-behind caching	59	62. Enterprise data grid object update flow	184
26. Write-behind caching	60	63. Java example with ClassAlias and FieldAlias annotations	189
27. Loader	61	64. .NET example with ClassAlias and FieldAlias attributes	189
28. Loader plug-in	63	65. Eclipse Equinox process for including all configuration and metadata in an OSGi bundle	234
29. Client loader	64	66. Eclipse Equinox process for specify configuration and metadata outside of an OSGi bundle	235
30. Periodic refresh	65		
31. JPA Loader architecture	70		
32. Serialization methods that are available when you are running logic that interacts with data objects directly in the data grid shard	72		
33. Serialization methods when you are not directly interacting with the data grid shard.	73		
34. Catalog service domain	101		
35. Communication path between a primary shard and replica shards	107		
36. Placement of an ObjectGrid map set with a deployment policy of 3 partitions with a minSyncReplicas value of 1, a maxSyncReplicas value of 1, and a maxAsyncReplicas value of 1	109		
37. Example placement of an ObjectGrid map set for the partition0 partition. The deployment policy has a minSyncReplicas value of 1, a maxSyncReplicas value of 2, and a maxAsyncReplicas value of 1.	112		

Tables

1. Feature comparison	50	11. Configuration settings to update the splicer.properties file	264
2. Failure discovery and recovery summary	96	12. Configuration settings for the properties in the splicer.properties file	264
3. Status value and response.	98	13. Configuration settings for the properties in the splicer.properties file	264
4. Commit sequence on the primary	99	14. Feature comparison	270
5. Synchronous commit processing	99	15. Available samples	281
6. LockMode values and existing method equivalents	127	16. Available articles by feature.	283
7. Lock mode compatibility matrix	128	17. Arguments for the xsadmin utility.	289
8. Arbitration approaches	177		
9. Data type equivalents between Java and C#	191		
10. Custom properties for configuring connection factories	246		

About the *Product Overview*

The WebSphere® eXtreme Scale documentation set includes three volumes that provide the information necessary to use, program for, and administer the WebSphere eXtreme Scale product.

WebSphere eXtreme Scale library

The WebSphere eXtreme Scale library contains the following books:

- The *Product Overview* contains a high-level view of WebSphere eXtreme Scale concepts, including use case scenarios, and tutorials.
- The *Installation Guide* describes how to install common topologies of WebSphere eXtreme Scale.
- The *Administration Guide* contains the information necessary for system administrators, including how to plan application deployments, plan for capacity, install and configure the product, start and stop servers, monitor the environment, and secure the environment.
- The *Programming Guide* contains information for application developers on how to develop applications for WebSphere eXtreme Scale using the included API information.

To download the books, go to the WebSphere eXtreme Scale library page.

You can also access the same information in this library in the WebSphere eXtreme Scale Version 8.6 information center. .

Using the books offline

All of the books in the WebSphere eXtreme Scale library contain links to the information center, with the following root URL: <http://pic.dhe.ibm.com/infocenter/wxsinfo/v8r6>. These links take you directly to related information. However, if you are working offline and encounter one of these links, you can search for the title of the link in the other books in the library. The API documentation, glossary, and messages reference are not available in PDF books.

Who should use this book

This book is intended for anyone that is interested in learning about WebSphere eXtreme Scale.

Getting updates to this book

You can get updates to this book by downloading the most recent version from the WebSphere eXtreme Scale library page.

How to send your comments

Contact the documentation team. Did you find what you needed? Was it accurate and complete? Send your comments about this documentation by e-mail to wasdoc@us.ibm.com.

Chapter 1. Product overview



The WebSphere eXtreme Scale licensed program is an elastic, scalable, in-memory data grid. The data grid dynamically caches, partitions, replicates, and manages application data and business logic across multiple servers. WebSphere eXtreme Scale performs massive volumes of transaction processing with high efficiency and linear scalability. With WebSphere eXtreme Scale, you can also get qualities of service such as transactional integrity, high availability, and predictable response times.

WebSphere eXtreme Scale overview

The WebSphere eXtreme Scale licensed program is an elastic, scalable, in-memory data grid. The data grid dynamically caches, partitions, replicates, and manages application data and business logic across multiple servers. WebSphere eXtreme Scale performs massive volumes of transaction processing with high efficiency and linear scalability. With WebSphere eXtreme Scale, you can also get qualities of service such as transactional integrity, high availability, and predictable response times.

WebSphere eXtreme Scale can be used in different ways. You can use the product as a very powerful cache, as an in-memory database processing space to manage application state, or to build Extreme Transaction Processing (XTP) applications. These XTP capabilities include an application infrastructure to support your most demanding business-critical applications.

Elastic scalability

Elastic scalability is possible through the use of distributed object caching. With elastic scalability, the data grid monitors and manages itself. The data grid can add or remove servers from the topology, which increases or decreases memory, network throughput, and processing capacity as needed. When a scale-out process is initiated, capacity is added to the data grid while it is running without requiring a restart. Conversely, a scale-in process immediately removes capacity. The data grid is also self-healing by automatically recovering from failures.

WebSphere eXtreme Scale versus an in-memory database

WebSphere eXtreme Scale cannot be considered an actual in-memory database. An in-memory database is too simple to handle some of the complexities that WebSphere eXtreme Scale can manage. If an in-memory database has a server that fails, it cannot repair the issue. A failure can be disastrous if your entire environment is on that one server.

To tackle the problem of this type of failure, eXtreme Scale splits the given data set into partitions, which are equivalent to constrained tree schemas. Constrained tree schemas describe the relationship between entities. When you are using partitions, the entity relationships must model a tree data structure. In this structure, the head of the tree is the root entity and is the only entity that is partitioned. All other children of the root entity are stored in the same partition as the root entity. Each partition exists as a primary copy, or shard. A partition also contains replica shards for backing up the data. An in-memory database cannot provide this function because it is not structured and dynamic in this way. With an in-memory database,

you must implement the operations that WebSphere eXtreme Scale does automatically. You can run SQL operations on in-memory databases, improving the processing speed compared to databases that are not in memory. WebSphere eXtreme Scale has its own query language instead of SQL support. This query language is more elastic, enables partitioning of data, and provides dependable failure recovery.

WebSphere eXtreme Scale with databases

With the write-behind cache feature, WebSphere eXtreme Scale can serve as a front-end cache for a database. By using this front-end cache, throughput increases while reducing database load and contention. WebSphere eXtreme Scale provides predictable scaling in and scaling out at predictable processing cost.

The following image shows that in a distributed, coherent cache environment, the eXtreme Scale clients send and receive data from the data grid. The data grid can be automatically synchronized with a backend data store. The cache is coherent because all of the clients see the same data in the cache. Each piece of data is stored on exactly one writable server in the cache. Having one copy of each piece of data prevents wasteful copies of records that might contain different versions of the data. A coherent cache holds more data as more servers are added to the data grid, and scales linearly as the data grid grows in size. The data can also be optionally replicated for additional fault tolerance.

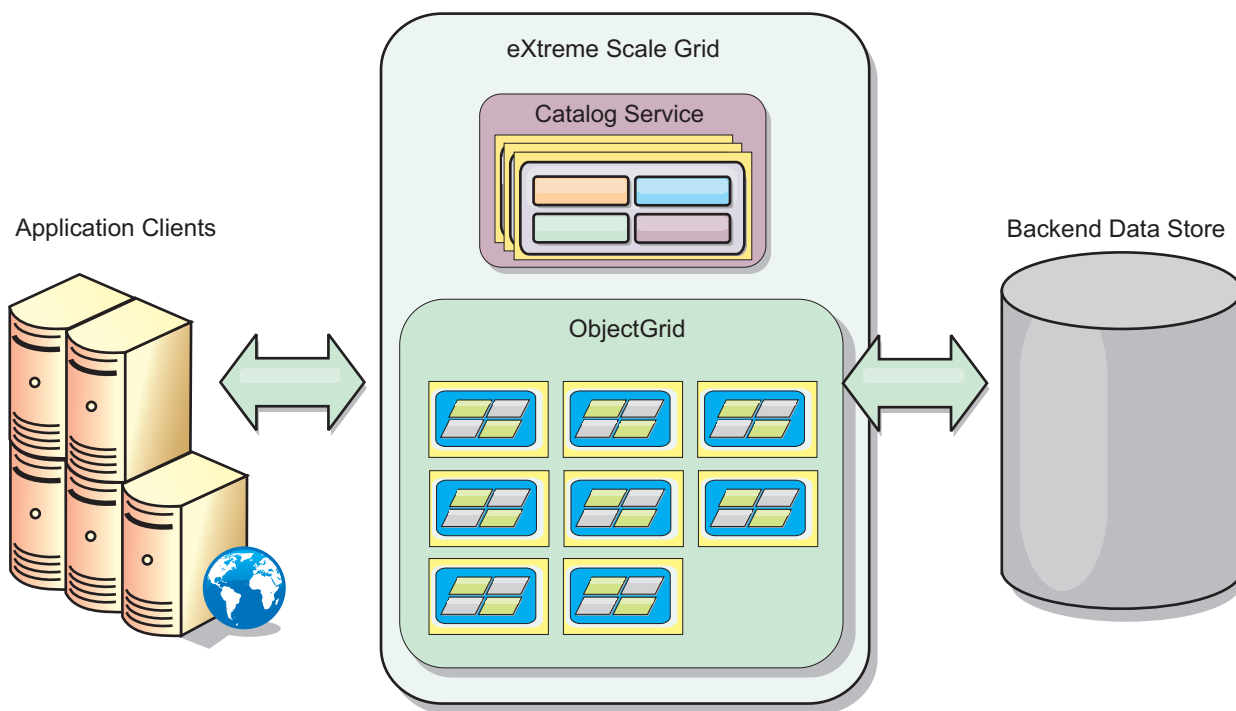


Figure 1. High-level topology

WebSphere eXtreme Scale has servers, called *container servers*, that provide its in-memory data grid. These servers can run inside WebSphere Application Server, or on simple Java™ Standard Edition (J2SE) Java virtual machines. More than one container server can run on a single physical server. As a result, the in-memory data grid can be large. The data grid is not limited by, and does not have an impact on, the memory or address space of the application or the application

server. The memory can be the sum of the memory of several hundred, or thousand, Java virtual machines, running on many different physical servers.

As an in-memory database processing space, WebSphere eXtreme Scale can be backed by disk, database, or both.

While eXtreme Scale provides several Java APIs, many use cases require no user programming, just configuration and deployment in your WebSphere infrastructure.

Data grid overview

The simplest eXtreme Scale programming interface is the `ObjectMap` interface, which is a simple map interface that includes: a `map.put(key,value)` method to put a value in the cache, and a `map.get(key)` method to later retrieve the value.

The fundamental data grid paradigm is a key-value pair, where the data grid stores values (Java objects), with an associated key (another Java object). The key is later used to retrieve the value. In eXtreme Scale, a map consists of entries of such key-value pairs.

WebSphere eXtreme Scale offers a number of data grid configurations, from a single, simple local cache, to a large distributed cache, using multiple Java virtual machines or servers.

In addition to storing simple Java objects, you can store objects with relationships. You can use a query language that is like SQL, with `SELECT ... FROM ... WHERE` statements to retrieve these objects. For example, an order object might have a customer object and multiple item objects associated with it. WebSphere eXtreme Scale supports one-to-one, one-to-many, many-to-one, and many-to-many relationships.

WebSphere eXtreme Scale also supports an `EntityManager` programming interface for storing entities in the cache. This programming interface is like entities in Java Enterprise Edition. Entity relationships can be automatically discovered from an entity descriptor XML file or annotations in the Java classes. You can retrieve an entity from the cache by primary key using the `find` method on the `EntityManager` interface. Entities can be persisted to or removed from the data grid within a transaction boundary.

Consider a distributed example where the key is a simple alphabetic name. The cache might be split into four partitions by key: partition 1 for keys starting with A-E, partition 2 for keys starting with F-L, and so on. For availability, a partition has a primary shard and a replica shard. Changes to the cache data are made to the primary shard, and replicated to the replica shard. You configure the number of servers that contain the data grid data, and eXtreme Scale distributes the data into shards over these server instances. For availability, replica shards are placed in separate physical servers from primary shards.

WebSphere eXtreme Scale uses a catalog service to locate the primary shard for each key. It handles moving shards among eXtreme Scale servers when the physical servers fail and later recover. For example, if the server containing a replica shard fails, eXtreme Scale allocates a new replica shard. If a server containing a primary shard fails, the replica shard is promoted to be the primary shard. As before, a new replica shard is constructed.

What's new in Version 8.6

WebSphere eXtreme Scale includes many new features in Version 8.6. Use this topic to learn about the latest product updates.

ASP.NET session state store provider

You can configure your ASP.NET applications to store session state in the data grid. Learn more....

Encode the credentialGeneratorProps property in client properties files for .NET

You can encode the value of the credentialGeneratorProps property in the Client.Net.properties file with the **FilePasswordEncoder** utility. Learn more...

Continuous query

When you develop client applications that interact with the data grid, you might require queries that retrieve automatic, real-time results when new entries are inserted or updated. You can use continuous query to be notified in your client Java virtual machine (JVM) when data is inserted or updated in the data grid. This feature makes grid and data management easier for developers, administrators or both. Learn more...

Disk overflow

You can use disk overflow to extend the data grid capacity by moving cache entries out of memory and into disk. When you enable disk overflow, entries that do not fit into the available memory capacity of the container servers are stored on disk. Learn more...

Display values of queried data

You can now display values of the keys in data queries that you create in the monitoring console or with the **xscmd** utility. Learn more...

Enterprise data grid

Enterprise data grids use the eXtremeIO transport mechanism and a new serialization format. With the new transport and serialization format, you can connect both Java and .NET clients to the same data grid. Learn more...

Global index

Global index extends the built-in HashIndex plug-in, and it runs on shards in a distributed, partitioned data grid. Global index tracks the location of indexed attributes in the data grid and provides efficient ways to find partitions, keys, values, or entries using attributes in large, partitioned data grid environment. Learn more...

Global index invalidation

You can optionally enable global index invalidation to improve invalidation efficiency in a large, partitioned environment; for example, more than 40 partitions. Learn more...

High Performance Extensible Logging (HPEL)

You can enable catalog and container server to use HPEL, an alternative to the basic log and trace facility. Learn more...

IBM® Support Assistant Data Collector

The IBM Support Assistant Data Collector is a tool you can run to gather data from your WebSphere eXtreme Scale environment for problem determination purposes. Learn more...

Inverse range index

You can configure an inverse range index using the built-in InverseRangeIndex plug-in. Learn more...

List of shard containers that are disabled for placement

When a problem occurs with placing shards on a particular shard container, the shard container is placed in a list that disables that shard container from receiving further placement requests. You can list the disabled shard containers and remove a shard container from the list with `xscmd` commands. Learn more...

Message center

The message center provides an aggregated view of event notifications for log and first-failure data capture (FFDC) messages. You can view these event notifications with the message center in the web console, the `xscmd` utility, or programmatically with MBeans. Learn more...

Multi-partition transactions

WebSphere eXtreme Scale Client now supports transaction updates to multiple partitions in a data grid. Learn more...

Near-cache invalidation

You can configure near cache invalidation to remove stale data from the near cache as quickly as possible. When an update, deletion, or invalidation operation is run against the remote data grid, an asynchronous invalidation gets triggered in the near cache. Learn more...

WebSphere eXtreme Scale Client for .NET

By installing the WebSphere eXtreme Scale Client for .NET, you can deploy .NET applications that access the data grid. Learn more...

New methods and APIs

- `upsert` method: The `upsert` and `upsertAll` methods replace the `ObjectMap` `put` and `putAll` methods. Learn more...
- `lock` method: When using pessimistic locking, you can use the `lock` method to lock data, or keys, without returning any data values. With the `lock` method, you can lock the key in the grid or lock the key and determine whether the value exists in the grid. In previous releases, you used the `get` and `getForUpdate` APIs to lock keys in the data grid. Learn more...

- `sessionIdOverrideClass`: This class implements the `com.ibm.websphere.xs.sessionmanager.SessionIDOverride` interface in order to override the default user ID retrieved from the `HttpSession.getId()` method. Learn more...

New `xscmd` utility commands and parameters

- `xscmd -c ShowSessionSize` command: Run this command to display the size of a specified session. Learn more...
- `xscmd -c getNotificationFilter` command: Run this command to display the current filters for new notifications from the message center. Learn more...
- `xscmd -c listenForNotifications` command: Run this command to listen for new notifications from the message center. Learn more...
- `xscmd -c setNotificationFilter` command: Run this command to create a filter for new notifications from the message center. Learn more...
- `xscmd -c showLinkedDomains` command: Run this command to check which catalog service domains are linked to your local catalog service domain. Learn more...
- `xscmd -c showNotificationHistory` command: Run this command to display the output of the event notification history in tabular format. Learn more...
- `-to` or `--timeout` parameter: Specify this parameter to reduce the timeout value to avoid waiting for operating system or other network timeouts during a network brown out or system loss. Learn more...
- `-hc` or `--linkHealthCheck` parameter: Use this parameter with the `xscmd -c showLinkedPrimaries` command to verify that the primary shards have the appropriate number of catalog service domain links. Learn more...
- `xscmd -c listDisabledForPlacement` command: Run this command to display a list of shard containers that have been disabled for shard placement. Learn more...
- `xscmd -c listIndoubts` command: Run this command to display a list of in-doubt transactions. Run this command when you want to resolve possible lock timeout exceptions on a partition. Learn more...
- `xscmd -c enableForPlacement -ct <shard_container>` command: Run this command to re-enable a shard containers that has been disabled for shard placement. Learn more...
- `xscmd -c showReplicationState` and `xscmd -c showDomainReplicationState` commands: Run these commands to see the state of the revisions across your catalog servers or catalog service domains. Learn more...
- `xscmd -c showTransport` command: Run these commands to display the transport type of the catalog service domain. Learn more...

Remote logging

You can enable remote logging to save log entries on a remote server. You must have a syslog server available to listen for and capture events. Learn more...

REST gateway support

You can use the Representational State Transfer (REST) gateway to access simple data grids that are hosted by a collective. This REST gateway is useful when you must access grid data from non-Java environments. Learn more...

Support for Java Servlets 3.0 Specification

WebSphere eXtreme Scale HTTP session management function now supports Java Servlets 3.0 specification. When you are writing applications for WebSphere eXtreme Scale in a stand-alone environment, only listeners specified explicitly in the `web.xml` are called back when sessions are invalidated using remote WebSphere eXtreme Scale container eviction.

eXtreme Data Format (XDF)

XDF is based on the `MapSerializerPlugin` plug-in, and is now the default serialization technology that is used when you are running IBM eXtremeIO (XIO) and have your map copy mode that is set to `COPY_TO_BYTES`. When you enable this feature, Java and C# objects can share data in the same data grid. Learn more...

WebApp feature

The Liberty profile `webApp` feature contains the capability to extend the Liberty profile web application. Add the `webApp` feature when you want to replicate HTTP session data for fault tolerance. Learn more...

WebGrid feature

With this Liberty profile feature, a Liberty profile server can host a data grid that caches data for applications to replicate HTTP session data for fault tolerance. Learn more...

Release notes

Links are provided to the product support Web site, to product documentation, and to last minute updates, limitations, and known problems for the product.

- “Accessing last-minute updates, limitations, and known problems”
- “Accessing system and software requirements”
- “Accessing product documentation”
- “Accessing the product support Web site” on page 8
- “Contacting IBM Software Support” on page 8

Accessing last-minute updates, limitations, and known problems

The release notes are available on the product support site as technotes. To see a list of all the technotes for WebSphere eXtreme Scale, go to the Support Web page. Clicking the links provided here will result in a search of the Support Web page for the relevant release notes, which will be returned as a list.

- To see a list of the release notes for Version 8.6, go to the Support Web page.

Accessing system and software requirements

The hardware and software requirements are documented on the following pages:

- Detailed system requirements

Accessing product documentation

For the entire information set, go to the Library page.

Accessing the product support Web site

To search for the latest technotes, downloads, fixes, and other support-related information, go to the Support Portal.

Contacting IBM Software Support

If you encounter a problem with the product, first try the following actions:

- Follow the steps described in the product documentation
- Look for related documentation in the online help
- Look up error messages in the message reference

If you cannot resolve your problem by any of the preceding methods, contact IBM Technical Support.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information about the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

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

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Mail Station P300
522 South Road
Poughkeepsie, NY 12601-5400
USA
Attention: Information Requests

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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 and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs.

© Copyright IBM Corp. _enter the year or years_. All rights reserved.

Programming interface information

This publication. primarily documents information that is NOT intended to be used as Programming Interfaces of WebSphere eXtreme Scale. This publication also documents intended Programming Interfaces that allow the customer to write programs to obtain the services of WebSphere eXtreme Scale. This information is identified where it occurs, either by an introductory statement to a chapter or section or by the following marking: Programming Interface information.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Privacy policy considerations

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software

Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

This Software Offering does not use cookies or other technologies to collect personally identifiable information.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's privacy policy at <http://www.ibm.com/privacy> and IBM's Online Privacy Statement at <http://www.ibm.com/privacy/details/us/en> sections entitled "Cookies, Web Beacons and Other Technologies" and "Software Products and Software-as-a Service".

Terms and conditions for information centers

Permissions for the use of these publications is granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at www.ibm.com/legal/copytrade.shtml.

Hardware and software requirements

Browse an overview of hardware and operating system requirements. Although you are not required to use a specific level of hardware or operating system for WebSphere eXtreme Scale, formally supported hardware and software options are available on the Systems Requirements page of the product support site. If a conflict exists between the information center and the System Requirements page, the information at the website takes precedence. Prerequisite information in the information center is provided as a convenience only.

See the System Requirements page for the official set of hardware and software requirements.

You can install and deploy the product in Java EE and Java SE environments. You can also bundle the client component with Java EE applications directly without integrating with WebSphere Application Server.

Hardware requirements

WebSphere eXtreme Scale does not require a specific level of hardware. The hardware requirements are dependent on the supported hardware for the Java Platform, Standard Edition installation that you use to run WebSphere eXtreme Scale. If you are using eXtreme Scale with WebSphere Application Server or another Java Platform, Enterprise Edition implementation, the hardware requirements of these platforms are sufficient for WebSphere eXtreme Scale.

Operating system requirements

.NET **8.6+** For details about the requirements for a .NET client environment, see Microsoft .NET considerations.

Java Each Java SE and Java EE implementation requires different operating system levels or fixes for problems that are discovered during the testing of the Java implementation. The levels required by these implementations are sufficient for eXtreme Scale.

Installation Manager requirements

Before you can install WebSphere eXtreme Scale, you must install Installation Manager. You can install Installation Manager using the product media, using a file obtained from the Passport Advantage® site, or using a file containing the most

current version of Installation Manager from the IBM Installation Manager download website. See Installing IBM Installation Manager and WebSphere eXtreme Scale product offerings for more information.

Web browser requirements

The web console supports the following Web browsers:

- Mozilla Firefox, version 3.5.x and later
- Microsoft Internet Explorer, version 7 and later

WebSphere Application Server requirements

8.6+

- WebSphere Application Server Version 7.0.0.21 or later
- WebSphere Application Server Version 8.0.0.2 or later

See the Recommended fixes for WebSphere Application Server for more information.

Java requirements

8.6+ Other Java EE implementations can use the eXtreme Scale run time as a local instance or as a client to eXtreme Scale servers. To implement Java SE, you must use Version 6 or later.

Directory conventions

The following directory conventions are used throughout the documentation to must reference special directories such as *wxs_install_root* and *wxs_home*. You access these directories during several different scenarios, including during installation and use of command-line tools.

wxs_install_root

The *wxs_install_root* directory is the root directory where WebSphere eXtreme Scale product files are installed. The *wxs_install_root* directory can be the directory in which the trial archive is extracted or the directory in which the WebSphere eXtreme Scale product is installed.

- Example when extracting the trial:

Example: /opt/IBM/WebSphere/eXtremeScale

- Example when WebSphere eXtreme Scale is installed to a stand-alone directory:

UNIX **Example:** /opt/IBM/eXtremeScale

Windows **Example:** C:\Program Files\IBM\WebSphere\extremeScale

- Example when WebSphere eXtreme Scale is integrated with WebSphere Application Server:

Example: /opt/IBM/WebSphere/AppServer

wxs_home

The *wxs_home* directory is the root directory of the WebSphere eXtreme Scale product libraries, samples, and components. This directory is the same as the *wxs_install_root* directory when the trial is extracted. For stand-alone installations, the *wxs_home* directory is the ObjectGrid subdirectory within the *wxs_install_root* directory. For installations that are integrated with

WebSphere Application Server, this directory is the optionalLibraries/ObjectGrid directory within the *wxs_install_root* directory.

- Example when extracting the trial:

Example: /opt/IBM/WebSphere/eXtremeScale

- Example when WebSphere eXtreme Scale is installed to a stand-alone directory:

UNIX **Example:** /opt/IBM/eXtremeScale/ObjectGrid

Windows **Example:** *wxs_install_root*\ObjectGrid

- Example when WebSphere eXtreme Scale is integrated with WebSphere Application Server:

Example: /opt/IBM/WebSphere/AppServer/optionalLibraries/ObjectGrid

was_root

The *was_root* directory is the root directory of a WebSphere Application Server installation:

Example: /opt/IBM/WebSphere/AppServer

.NET 8.6+ net_client_home

The *net_client_home* directory is the root directory of a .NET client installation.

Example: C:\Program Files\IBM\WebSphere\eXtreme Scale .NET Client

restservice_home

The *restservice_home* directory is the directory in which the WebSphere eXtreme Scale REST data service libraries and samples are located. This directory is named *restservice* and is a subdirectory under the *wxs_home* directory.

- Example for stand-alone deployments:

Example: /opt/IBM/WebSphere/eXtremeScale/ObjectGrid/restservice

Example: *wxs_home*\restservice

- Example for WebSphere Application Server integrated deployments:

Example: /opt/IBM/WebSphere/AppServer/optionalLibraries/ObjectGrid/restservice

tomcat_root

The *tomcat_root* is the root directory of the Apache Tomcat installation.

Example: /opt/tomcat5.5

wasce_root

The *wasce_root* is the root directory of the WebSphere Application Server Community Edition installation.

Example: /opt/IBM/WebSphere/AppServerCE

java_home

The *java_home* is the root directory of a Java Runtime Environment (JRE) installation.

UNIX **Example:** /opt/IBM/WebSphere/eXtremeScale/java

Windows **Example:** *wxs_install_root*\java

samples_home

The *samples_home* is the directory in which you extract the sample files that are used for tutorials.

UNIX Example: `wxs_home/samples`

Windows Example: `wxs_home\samples`

dvd_root

The `dvd_root` directory is the root directory of the DVD that contains the product.

Example: `dvd_root/docs/`

equinox_root

The `equinox_root` directory is the root directory of the Eclipse Equinox OSGi framework installation.

Example: `/opt/equinox`

user_home

The `user_home` directory is the location where user files are stored, such as security profiles.

Windows `c:\Documents and Settings\user_name`

UNIX `/home/user_name`

WebSphere eXtreme Scale technical overview

WebSphere eXtreme Scale is an elastic, scalable, in-memory data grid. It dynamically caches, partitions, replicates, and manages application data and business logic across multiple servers.

Because WebSphere eXtreme Scale is not an in-memory database, you must consider specific configuration requirements. The first step to deploying a data grid is to start a core group and catalog service. The catalog service acts as coordinator for all other Java virtual machines that are participating in the data grid and manages configuration information. WebSphere eXtreme Scale processes are started with commands that you issue on the command line.

The next step is to start container server processes for the data grid to store and retrieve data. As container servers are started, they automatically register themselves with the core group and catalog service. By registering, the catalog servers can cooperate in providing data grid services. More servers increase both data grid capacity and reliability.

A local data grid is a simple, single-instance grid where all the data is in the one data grid. To effectively use WebSphere eXtreme Scale as an in-memory database processing space, you can configure and deploy a distributed data grid. The data in the distributed grid is spread out over the various eXtreme Scale servers so that each server contains only some of the data. This portion of data is a partition.

A key distributed data grid configuration parameter is the number of partitions in the grid. The grid data is partitioned into this number of subsets, each of which is called a partition. The catalog service locates the partition for the data based on its key. The number of partitions directly affects the capacity and scalability of the data grid. A server can contain one or more data grid partitions. As a result, the memory space of the servers limits the size of a partition. Conversely, increasing the number of partitions increases the capacity of the data grid. The maximum capacity of a data grid is the number of partitions times the usable memory size of each server. A server can be a JVM, but you can define your container server to suit your deployment environment.

The data of a partition is stored in a shard. For availability, a data grid can be configured with replicas, which can be synchronous or asynchronous. Changes to the grid data are made to the primary shard, and replicated to the replica shards. The total memory that is used or required by a data grid can be calculated with the following equation: the size of the data grid times (1 (for the primary) + the number of replicas).

WebSphere eXtreme Scale distributes the shards of a data grid over the number of servers that are in the data grid. These servers might be on the same or different physical servers. For availability, replica shards are placed in separate physical servers from primary shards.

WebSphere eXtreme Scale monitors the status of its servers and moves shards during shard or physical server failure and recovery. For example, if the server that contains a replica shard fails, WebSphere eXtreme Scale allocates a new replica shard, and replicate data from the primary to the new replica. If a server that contains a primary shard fails, the replica shard is promoted to be the primary shard, and, a new replica shard is constructed. If you start an extra server for the data grid, the shards are balanced over all servers. This rebalancing is called scale-out. Similarly, for scale-in, you might stop one of the servers to reduce the resources that are used by a data grid. As a result, the shards are balanced over the remaining servers.

Caching overview

WebSphere eXtreme Scale can operate as an in-memory database processing space, which you can use to provide in-line caching for a database back-end or to serve as a side-cache. In-line caching uses eXtreme Scale as the primary means for interacting with the data. When eXtreme Scale is used as a side-cache, the back-end is used in conjunction with the data grid. This section describes various cache concepts and scenarios and discusses the available topologies for deploying a data grid.

Caching architecture: Maps, containers, clients, and catalogs

With WebSphere eXtreme Scale, your architecture can use local in-memory data caching or distributed client-server data caching.

WebSphere eXtreme Scale requires minimal additional infrastructure to operate. The infrastructure consists of scripts to install, start, and stop a Java Platform, Enterprise Edition application on a server. Cached data is stored in the eXtreme Scale server, and clients remotely connect to the server.

Distributed caches offer increased performance, availability and scalability and can be configured using dynamic topologies, in which servers are automatically balanced. You can also add additional servers without restarting your existing eXtreme Scale servers. You can create either simple deployments or large, terabyte-sized deployments in which thousands of servers are needed.

Catalog service

The catalog service controls placement of shards and discovers and monitors the health of container servers in the data grid. The catalog service hosts logic that should be idle and has little influence on scalability. It is built to service hundreds of container servers that become available simultaneously, and run services to manage the container servers.

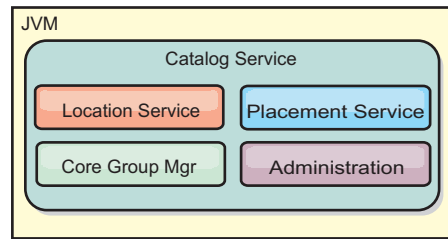


Figure 2. Catalog service

The catalog server responsibilities consist of the following services:

Location service

The location service runs on the data grid members to provide locality to clients and container servers. Container servers register with the location service to register the hosted applications. Clients can then use the location service to search for container servers to host applications.

Placement service

The catalog service manages the placement of shards across available container servers. The placement service is responsible for maintaining balance across physical resources and allocating individual shards to their host container server. The placement service runs as a One of N elected service in the cluster and in the data grid. This means that exactly one instance of the placement service is running. If an instance fails, another process is elected and takes over. For redundancy, the state of the catalog service is replicated across all the servers that are hosting the catalog service.

Core group manager

The core group manages peer grouping for availability monitoring, organizes container servers into small groups of servers, and automatically federates the groups of servers.

The catalog service uses the high availability manager (HA manager) to group processes together for availability monitoring. Each grouping of the processes is a core group. The core group manager dynamically groups the processes together. These processes are kept small to allow for scalability. Each core group elects a leader that is responsible for sending heartbeat messages to the core group manager. These messages detect if an individual member failed or is still available. The heartbeat mechanism is also used to detect if all the members of a group failed, which causes the communication with the leader to fail.

The core group manager is responsible for organizing containers into small groups of servers that are loosely federated to make a data grid. When a container server first contacts the catalog service, it waits to be assigned to either a new or existing group. An eXtreme Scale deployment consists of many such groups, and this grouping is a key scalability enabler. Each group consists of Java virtual machines. An elected leader uses the heartbeat mechanism to monitor the availability of the other groups. The leader relays availability information to the catalog service to allow for failure reaction by reallocation and route forwarding.

Administration

The catalog service is also the logical entry point for system administration.

The catalog service hosts a Managed Bean (MBean) and provides Java Management Extensions (JMX) URLs for any of the servers that the catalog service is managing.

For high availability, configure a catalog service domain. A catalog service domain consists of multiple Java virtual machines, including a master JVM and a number of backup Java virtual machines. For more information, see “High availability catalog service” on page 101.

Container servers, partitions, and shards

The container server stores application data for the data grid. This data is generally broken into parts, which are called partitions. Partitions are hosted across multiple shard containers. Each container server in turn hosts a subset of the complete data. A JVM might host one or more shard containers and each shard container can host multiple shards.

Remember: Plan out the heap size for the container servers, which host all of your data. Configure the heap settings accordingly.

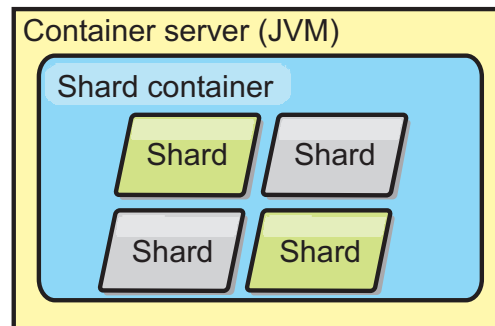


Figure 3. Container server

Partitions host a subset of the data in the grid. WebSphere eXtreme Scale automatically places multiple partitions in a single shard container and spreads the partitions out as more container servers become available.

Important: Choose the number of partitions carefully before final deployment because the number of partitions cannot be changed dynamically. A hash mechanism is used to locate partitions in the network and eXtreme Scale cannot rehash the entire data set after it has been deployed. As a general rule, you can overestimate the number of partitions

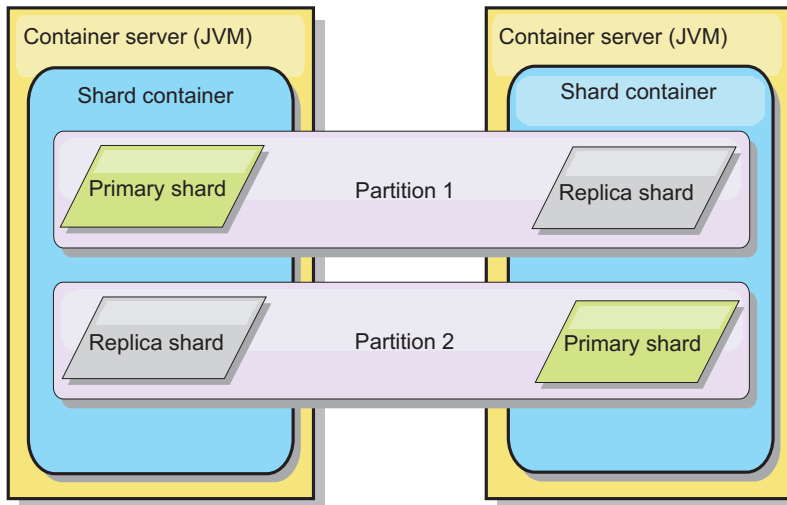


Figure 4. Partition

Shards are instances of partitions and have one of two roles: primary or replica. The primary shard and its replicas make up the physical manifestation of the partition. Every partition has several shards that each host all of the data contained in that partition. One shard is the primary, and the others are replicas, which are redundant copies of the data in the primary shard. A primary shard is the only partition instance that allows transactions to write to the cache. A replica shard is a "mirrored" instance of the partition. It receives updates synchronously or asynchronously from the primary shard. The replica shard only allows transactions to read from the cache. Replicas are never hosted in the same container server as the primary and are not normally hosted on the same machine as the primary.

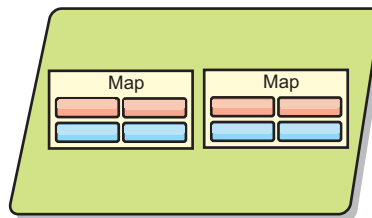


Figure 5. Shard

To increase the availability of the data, or increase persistence guarantees, replicate the data. However, replication adds cost to the transaction and trades performance in return for availability. With eXtreme Scale, you can control the cost as both synchronous and asynchronous replication is supported, as well as hybrid replication models using both synchronous and asynchronous replication modes. A synchronous replica shard receives updates as part of the transaction of the primary shard to guarantee data consistency. A synchronous replica can double the response time because the transaction has to commit on both the primary and the synchronous replica before the transaction is complete. An asynchronous replica shard receives updates after the transaction commits to limit impact on performance, but introduces the possibility of data loss as the asynchronous replica can be several transactions behind the primary.

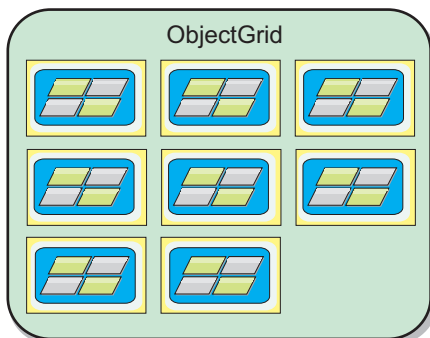


Figure 6. ObjectGrid

Maps

Java .NET

A map is a container for key-value pairs, which allows an application to store a value indexed by a key. Maps support indexes that can be added to index attributes on the key or value. These indexes are automatically used by the query runtime to determine the most efficient way to run a query.

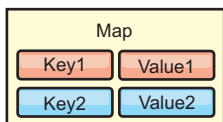


Figure 7. Map

A map set is a collection of maps with a common partitioning algorithm. The data within the maps are replicated based on the policy defined on the map set. A map set is only used for distributed topologies and is not needed for local topologies.

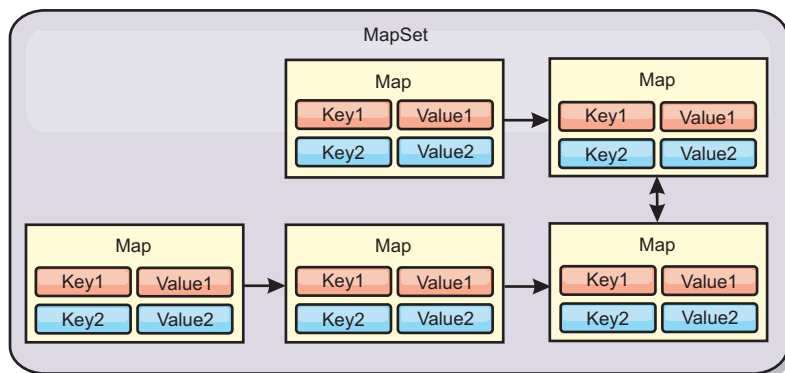


Figure 8. Map sets

A map set can have a schema associated with it. A schema is the metadata that describes the relationships between each map when using homogeneous Object types or entities.

WebSphere eXtreme Scale can store serializable Java objects or eXtreme data format (XDF) serialized objects in each of the maps using the ObjectMap API for Java clients or the IGridMapAutoTx/IGridMapPessimisticTx API for .NET clients. A schema can be defined over the maps to identify the relationship between the objects in the maps where each map holds objects of a single type. Defining a schema for maps is required to query the contents of the map objects. WebSphere eXtreme Scale can have multiple map schemas defined.

For more information about caching objects, see Getting started tutorial module 2: Create a client application.

WebSphere eXtreme Scale can also store entities using the EntityManager API. Each entity is associated with a map. The schema for an entity map set is automatically discovered using either an entity descriptor XML file or annotated Java classes. Each entity has a set of key attributes and set of non-key attributes. An entity can also have relationships to other entities. WebSphere eXtreme Scale supports one to one, one to many, many to one and many to many relationships. Each entity is physically mapped to a single map in the map set. Entities allow applications to easily have complex object graphs that span multiple Maps. A distributed topology can have multiple entity schemas.

For more information about caching objects with the EntityManager API,, see Caching objects and their relationships (EntityManager API).

Clients

Clients connect to a catalog service, retrieve a description of the server topology, and communicate directly to each server as needed. When the server topology changes because new servers are added or existing servers have failed, the dynamic catalog service routes the client to the appropriate server that is hosting the data. Clients must examine the keys of application data to determine which partition to route the request. Clients can read data from multiple partitions in a single transaction. However, clients can update only a single partition in a transaction. After the client updates some entries, the client transaction must use that partition for updates.

8.6+ You can use two types of clients: Java clients and .NET clients. You can use Java clients only, .NET clients only, or you can use both types to access the same catalog server and data grid.

Java clients

Java client applications run on Java virtual machines (JVM) and connect to the catalog service and container servers.

- A catalog service exists in its own data grid of Java virtual machines. A single catalog service can be used to manage multiple clients or container servers.
- A container server can be started in a JVM by itself or can be loaded into an arbitrary JVM with other containers for different data grids.
- A client can exist in any JVM and communicate with one or more data grids. A client can also exist in the same JVM as a container server.

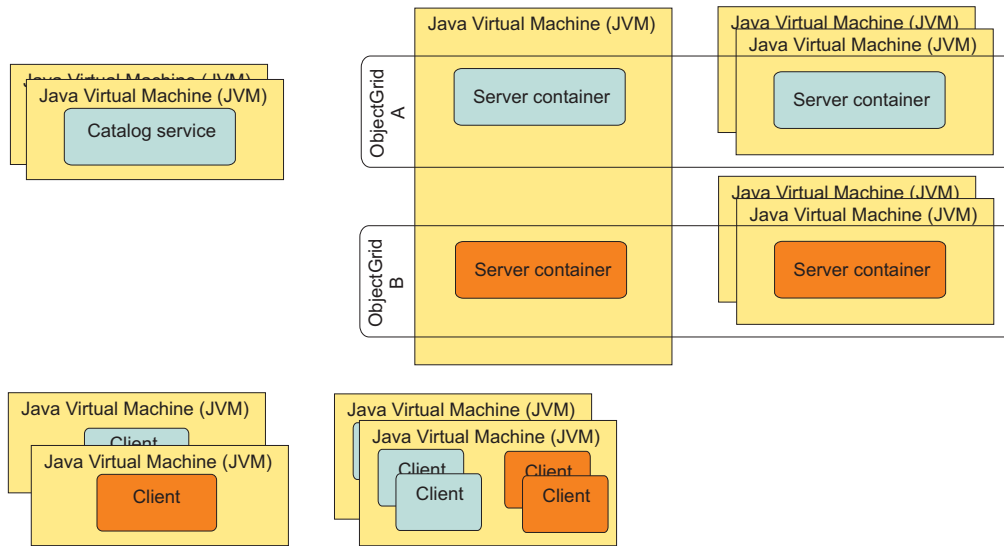


Figure 9. Possible topologies

8.6+ .NET clients

.NET clients work similarly to Java clients, but do not run in JVMs. .NET clients are installed remotely from your catalog and container servers. You connect to the catalog service from within the application. You can use a .NET client application to connect to the same data grid as your Java clients. For more information about using Java and .NET clients together, see “Scenario: Configuring an enterprise data grid” on page 183.

Enterprise data grid overview

Enterprise data grids use the eXtremeIO transport mechanism and a new serialization format. With the new transport and serialization format, you can connect both Java and .NET clients to the same data grid.

With the enterprise data grid, you can create multiple types of applications, written in different programming languages, to access the same objects in the data grid. In prior releases, data grid applications had to be written in the Java programming language only. With the enterprise data grid function, you can write .NET applications that can create, retrieve, update, and delete objects from the same data grid as the Java application.

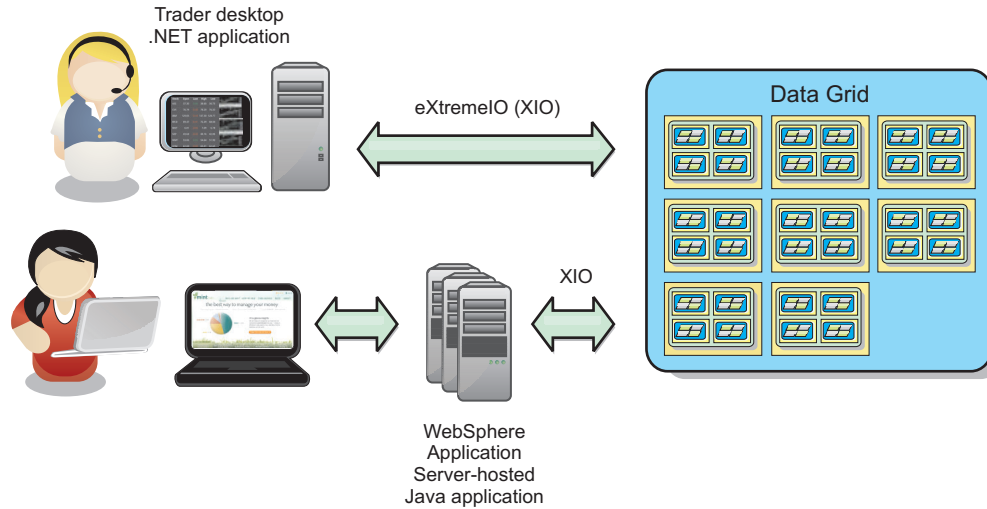


Figure 10. Enterprise data grid high-level overview

Object updates across different applications

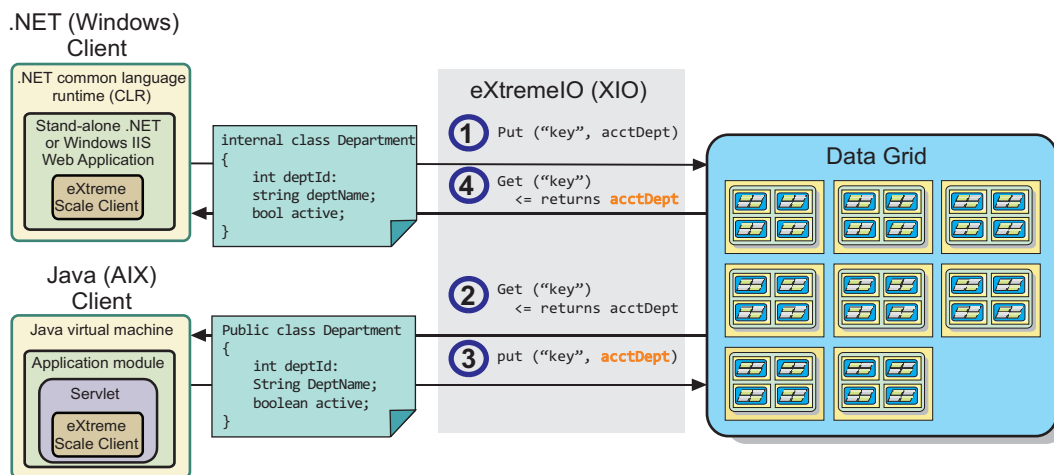


Figure 11. Enterprise data grid object update flow

1. The .NET client saves data in its format to the data grid.
2. The data is stored in a universal format, so that when the Java client requests this data it can be converted to Java format.
3. The Java client updates and re-saves the data.
4. The .NET client accesses the updated data, during which the data is converted to .NET format.

Transport mechanism

eXtremeIO (XIO) is a cross-platform transport protocol. XIO replaces the Java-bound Object Request Broker (ORB). With the ORB, WebSphere eXtreme Scale is bound to Java native client applications. XIO is a customized transport mechanism that is specifically targeted for data caching and enables client applications that are in different programming languages to connect to the data grid.

Serialization format

eXtreme data format (XDF) is a cross-platform serialization format. XDF replaces Java serialization on maps that have a copyMode attribute value of COPY_TO_BYTES in the ObjectGrid descriptor XML file. With XDF, performance is faster and data is more compact. In addition, the introduction of XDF enables client applications that are in different programming languages to connect to the same data grid.

Class evolution

eXtreme data format (XDF) allows for class evolution. With class evolution, you can evolve the class definitions that are used in the data grid without affecting older applications that are using previous versions of the class. These older classes are accessing data in the same map as the new applications.

Overview

Class evolution is a further extension of the identification of classes and fields that determine whether two types are compatible enough to function together. Classes can function together when one of the classes has fewer fields than the other class. The following user scenarios are designed into the XDF implementation :

Multiple versions of the same object class

In this scenario, you have a map in a sales application that is used tracking customers. This map has two different interfaces. One interface is for the web purchases. The second interface is for the phone purchases. In version 2 of this sales application, you decide to give discounts to web shoppers based on their purchasing habits. This discount is stored with the Customer object. The phone sales employees are still using version 1 of the application, which is unaware of the new discount field in the web version. You want Customer objects from version 2 of the application to work with Customer objects that were created with the version 1 application and vice versa.

Multiple versions of a different object class

In this scenario, you have a sales application that is written in Java that keeps a map of Customer objects. You also have another application that is written in C# and is used to manage the inventory in the warehouse and ship goods to customers. These classes are currently compatible based on the names of the classes, fields, and types. In your Java sales application, you want to add an option to the Customer record to associate the sales person with a customer account. However, you do not want to update the warehouse application to store this field because it is not needed in the warehouse.

Multiple incompatible versions of the same class

In this scenario, your sales and inventory applications both contain a Customer object. The inventory application uses an ID field that is a string and the sales application uses an ID field that is an integer. These types are not compatible. As a result, the objects are probably not stored in the same map. The objects must be handled by the XDF serialization and treated as two distinct types. While this scenario is not really class evolution, it is a consideration that must be part of your overall application design.

Determination for evolution

XDF attempts to evolve a class when the class names match and the field names do not have conflicting types. Using the ClassAlias and FieldAlias annotations are useful when you are trying to match classes between C# and Java applications

where the names of the classes or fields are slightly different. You can put these annotations on either the Java and C# application, or both. However, the lookup for the class in the Java application can be less efficient than defining the `ClassAlias` on the C# application. For more information about the `ClassAlias` and `FieldAlias` annotations, see “`ClassAlias` and `FieldAlias` annotations” on page 189

The effect of missing fields in serialized data

The constructor of the class is not invoked during deserialization, so any missing fields have a default that is assigned to it based on the language. The application that is adding new fields must be able to detect the missing fields and react when an older version of class is retrieved.

Updating the data is the only way for older applications to keep the newer fields

An application might run a fetch operation and update the map with an older version of the class that is missing some fields in the serialized value from the client. The server then merges the values on the server and determines whether any fields in the original version are merged into the new record. If an application runs a fetch operation, and then removes and inserts an entry, the fields from the original value are lost.

Merging capabilities

Objects within an array or collection are not merged by XDF. It is not always clear whether an update to an array or collection is intended to change the elements of that array or the type. If a merge occurs based on positioning, when an entry in the array is moved, XDF might merge fields that are not intended to be associated. As a result, XDF does not attempt to merge the contents of arrays or collections. However, if you add an array in a newer version of a class definition, the array gets merged back into the previous version of the class.

IBM eXtremeMemory

IBM eXtremeMemory enables objects to be stored in native memory instead of the Java heap. By moving objects off the Java heap, you can avoid garbage collection pauses, leading to more constant performance and predictable response times.

The Java virtual machine (JVM) relies on usage heuristics to collect, compact, and expand process memory. The garbage collector completes these operations. However, running garbage collection has an associated cost. The cost of running garbage collection increases as the size of the Java heap and number of objects in the data grid increase. The JVM provides different heuristics for different use cases and goals: optimum throughput, optimum pause time, generational, balanced, and real-time garbage collection. No heuristic is perfect. A single heuristic cannot suit all possible configurations.

WebSphere eXtreme Scale uses data caching, with distributed maps that have entries with a well-known lifecycle. This lifecycle includes the following operations: GET, INSERT, DELETE, and UPDATE. By using these well-known map lifecycles, eXtremeMemory can manage memory usage for data grid objects in container servers more efficiently than the standard JVM garbage collector.

The following diagram shows how using eXtremeMemory leads to more consistent relative response times in the environment. As the relative response times reach the

higher percentiles, the requests that are using eXtremeMemory have lower relative response times. The diagram shows the 95-100 percentiles.

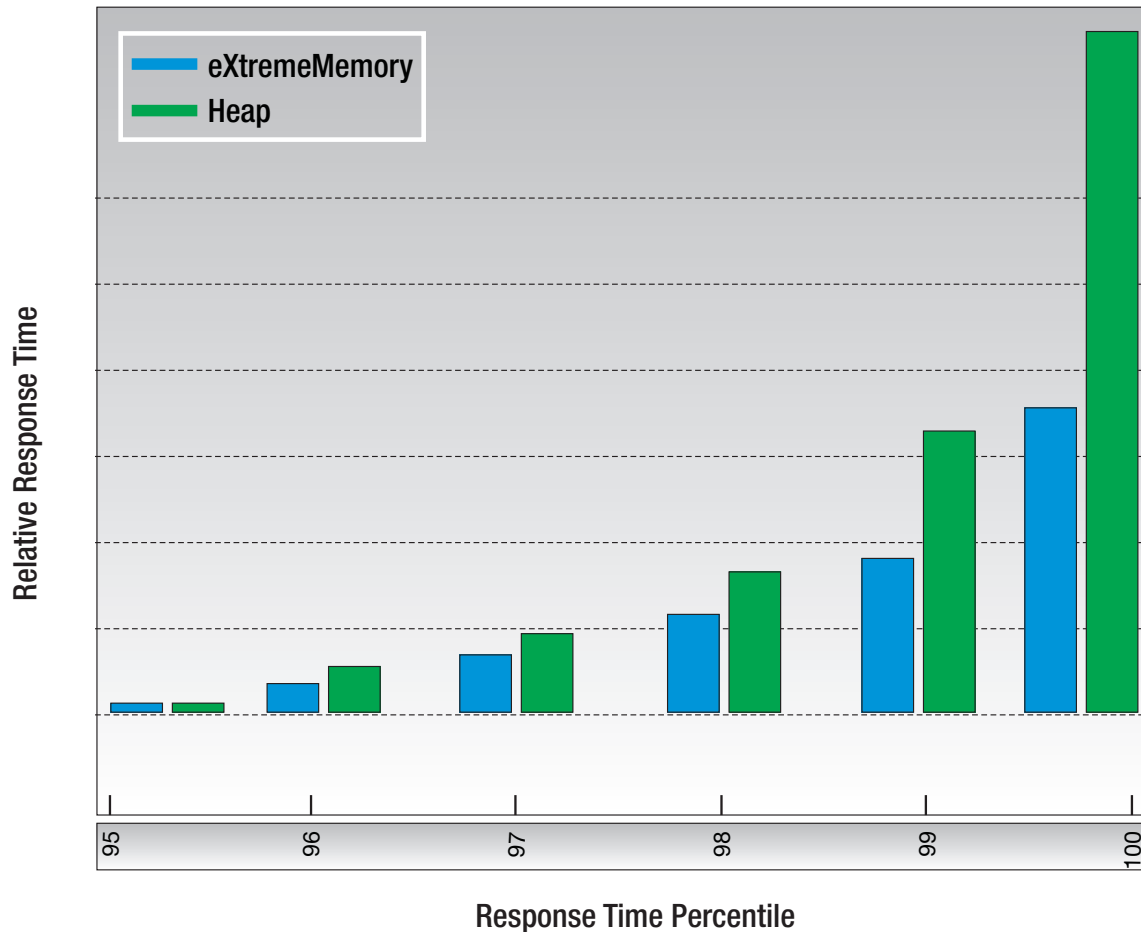


Figure 12. Comparison of eXtremeMemory and Heap Storage Response Times

Zones

Zones give you control over shard placement. Zones are user-defined logical groupings of physical servers. The following are examples of different types of zones: different blade servers, chassis of blade servers, floors of a building, buildings, or different geographical locations in a multiple data center environment. Another use case is in a virtualized environment where many server instances, each with a unique IP address, run on the same physical server.

Zones defined between data centers

The classic example and use case for zones is when you have two or more geographically dispersed data centers. Dispersed data centers spread your data grid over different locations for recovery from data center failure. For example, you might want to ensure that you have a full set of asynchronous replica shards for your data grid in a remote data center. With this strategy, you can recover from the

failure of the local data center transparently, with no loss of data. Data centers themselves have high speed, low latency networks. However, communication between one data center and another has higher latency. Synchronous replicas are used in each data center where the low latency minimizes the impact of replication on response times. Using asynchronous replication reduces impact on response time. The geographic distance provides availability in case of local data center failure.

In the following example, primary shards for the Chicago zone have replicas in the London zone. Primary shards for the London zone have replicas in the Chicago zone.

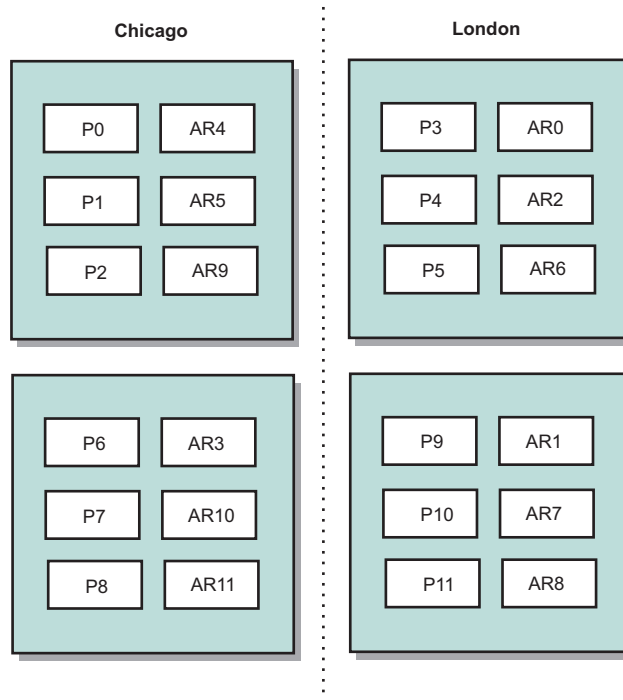


Figure 13. Primaries and replicas in zones

Three configuration settings in eXtreme Scale control shard placement:

- Set the deployment file
- Group containers
- Specify rules

The following sections explain the different options, presented loosely from least to most complicated.

Disable development mode

In your deployment XML file, set: `developmentMode="false"`.

This simple step activates the first eXtreme Scale shard placement policy.

For more information about the XML file, see Deployment policy descriptor XML file.

Policy 1: Shards for the same partition are placed in separate physical servers.

Consider a simple example of a data grid with one replica shard. With this policy, the primary and replica shards for each partition are on different physical servers. If a single physical server fails, no data is lost. The primary or replica shard for each partition are on different physical servers that did not fail, or both are on some other physical server that did not fail.

The high availability and simplicity of this policy make it the most efficient setting for all production environments. In many cases, applying this policy is the only step required for effectively controlling shard placement in your environment.

In applying this policy, a physical server is defined by an IP address. Shards are placed in container servers. Container servers have an IP address, for example, the **-listenerHost** parameter on the start server script. Multiple container servers can have the same IP address.

Since a physical server has multiple IP addresses, consider the next step for more control of your environment.

Define zones to group container servers

Container servers are assigned to zones with the **-zone** parameter on the start server script. In a WebSphere Application Server environment, zones are defined through node groups with a specific name format: `ReplicationZone<Zone>`. In this way, you choose the name and membership of your zones. For more information, see *Defining zones for container servers*.

Policy 2: Shards for the same partition are placed in separate zones.

Consider extending the example of a data grid with one replica shard by deploying it across two data centers. Define each data center as an independent zone. Use a zone name of DC1 for the container servers in the first data center, and DC2 for the container servers in the second data center. With this policy, the primary and replica shards for each partition would be in different data centers. If a data center fails, no data is lost. For each partition, either its primary or replica shard is in the other data center.

With this policy, you can control shard placement by defining zones. You choose your physical or logical boundary or grouping of interest. Then, choose a unique zone name for each group, and start the container servers in each of your zones with the name of the appropriate zone. Thus eXtreme Scale places shards so that shards for the same partition are placed in separate zones.

Specify zone rules

The finest level of control over shard placement is achieved using zone rules. Zone rules are specified in the `zoneMetadata` element of the eXtreme Scale deployment policy descriptor XML. A zone rule defines a set of zones in which shards are placed. A `shardMapping` element assigns a shard to a zone rule. The `shard` attribute of the `shardMapping` element specifies the shard type:

- P specifies the primary shard
- S specifies synchronous replica shards
- A specifies asynchronous replica shards.

If more than one synchronous or asynchronous replica exist, then you must provide `shardMapping` elements of the appropriate shard type. The

exclusivePlacement attribute of the zoneRule element determines the placement of shards in the same partition in zones. The exclusivePlacement attribute values are:

- true (a shard cannot be placed in the same zone as another shard from the same partition).

Remember: For the "true" case, you must have at least as many zones in the rule as you have shards using it. Doing so ensures that each shard can be in its own zone.

- false (shards from the same partition can be placed in the same zone).

The default setting is true.

For more information, see Example: Zone definitions in the deployment policy descriptor XML file.

Extended use cases

The following are various use cases for shard placement strategies:

Rolling upgrades

Consider a scenario in which you want to apply rolling upgrades to your physical servers, including maintenance that requires restarting your deployment. In this example, assume that you have a data grid spread across 20 physical servers, defined with one synchronous replica. You want to shut down 10 of the physical servers at a time for the maintenance.

When you shut down groups of 10 physical servers, no partition has both its primary and replica shards on the servers you are shutting down. Otherwise, you lose the data from that partition.

The easiest solution is to define a third zone. Instead of two zones of 10 physical servers each, use three zones, two with seven physical servers, and one with six. Spreading the data across more zones allows for better failover for availability.

Rather than defining another zone, the other approach is to add a replica.

Upgrading eXtreme Scale

When you are upgrading eXtreme Scale software in a rolling manner with data grids that contain live data, consider the following issues. The catalog service software version must be greater than or equal to the container server software versions. Upgrade all the catalog servers first with a rolling strategy. Read more about upgrading your deployment in the topicUpdating eXtreme Scale servers.

Changing data model

A related issue is how to change the data model or schema of objects that are stored in the data grid without causing downtime. It would be disruptive to change the data model by stopping the data grid and restarting with the updated data model classes in the container server classpath, and reloading the data grid. An alternative would be to start a new data grid with the new schema, copy the data from the old data grid to the new data grid, then shut down the old data grid.

Each of these processes are disruptive and result in downtime. To change the data model without downtime, store the object in one of these formats:

- Use XML as the value
- Use a blob made with Google protobuf
- Use JavaScript Object Notation (JSON)

Write serializers to go from plain old Java object (POJO) to one of these formats easily on the client side. Schema changes become easier.

Virtualization

Cloud computing and virtualization are popular emerging technologies. By default, eXtreme Scale insures that two shards for the same partition are never placed on the same IP address as described in Policy 1. When you are deploying on virtual images, such as VMware, many server instances, each with a unique IP address, can be run on the same physical server. To ensure that replicas can only be placed on separate physical servers, you can use zones to solve the problem. Group your physical servers into zones, and use zone placement rules to keep primary and replica shards in separate zones.

Zones for wide-area networks

You might want to deploy a single eXtreme Scale data grid over multiple buildings or data centers with slower network connections. Slower network connections lead to lower bandwidth and higher latency connections. The possibility of network partitions also increases in this mode due to network congestion and other factors.

To deal with these risks, the eXtreme Scale catalog service organizes container servers into core groups that exchange heartbeats to detect container server failure. These core groups do not span zones. A leader within each core group pushes membership information to the catalog service. The catalog service verifies any reported failures before responding to membership information by heartbeating the container server in question. If the catalog service sees a false failure detection, the catalog service takes no action. The core group partition heals quickly. The catalog service also heartbeats core group leaders periodically at a slow rate to handle the case of core group isolation.

Evictors

Java

Evictors remove data from the data grid. You can either set a time-based evictor or because evictors are associated with BackingMaps, use the BackingMap interface to specify the pluggable evictor.

Evictor types

A default TTL evictor is created with every dynamic backing map. The evictor removes entries based on a time to live concept.

None

Specifies that entries never expire and therefore are never removed from the map.

Creation time

Specifies that entries are evicted depending on when they were created.

If you are using the Creation time (`CREATION_TIME ttlType`) evictor, the evictor evicts an entry when its time from creation equals its TTL (TimeToLive attribute) value, which is set in milliseconds in your application configuration. If you set the TTL TTL (TimeToLive attribute) value to 10 seconds, the entry is automatically evicted ten seconds after it was inserted.

It is important to take caution when setting this value for the Creation time evictor type (`CREATION_TIME ttlType`). This evictor is best used when reasonably high amounts of additions to the cache exist that are only used for a set amount of time. With this strategy, anything that is created is removed after the set amount of time.

The Creation time evictor type (`CREATION_TIME ttlType`) is useful in scenarios such as refreshing stock quotes every 20 minutes or less. Suppose a Web application obtains stock quotes, and getting the most recent quotes is not critical. In this case, the stock quotes are cached in a data grid for 20 minutes. After 20 minutes, the map entries expire and are evicted. Every twenty minutes or so, the data grid uses the Loader plug-in to refresh the data with data from the database. The database is updated every 20 minutes with the most recent stock quotes.

Last access time

Specifies that entries are evicted depending upon when they were last accessed, whether they were read or updated.

Last update time

Specifies that entries are evicted depending upon when they were last updated.

If you are using the Last access time (`LAST_ACCESS_TIME`) or the Last update time evictor type (`LAST_UPDATE_TIME ttlType` attribute), set the TTL value (TimeToLive attribute) to a lower number than if you are using the Creation time evictor (`CREATION_TIME ttlType`). The entries TimeToLive attribute are reset every time it is accessed. In other words, if the TimeToLive attribute value is equal to 15 and an entry has existed for 14 seconds but then gets accessed, it does not expire again for another 15 seconds. If you set the TTL value to a relatively high number, many entries might never be evicted. However, if you set the value to something like 15 seconds, entries might be removed when they are not often accessed.

The Last access time (`LAST_ACCESS_TIME`) or Last update time evictor type (`LAST_UPDATE_TIME ttlType`) are useful in scenarios such as holding session data from a client, using a data grid map. Session data must be destroyed if the client does not use the session data for some period of time. For example, the session data times out after 30 minutes of no activity by the client. In this case, using an evictor type of Last access time (`LAST_ACCESS_TIME`) or Last update time (`LAST_UPDATE_TIME`) with the TTL value set to 30 minutes is appropriate for this application.

You may also write your own evictors: For more information, see [Writing a custom evictor](#).

Pluggable evictor

The default TTL evictor uses an eviction policy that is based on time, and the number of entries in the `BackingMap` has no affect on the expiration time of an

entry. You can use an optional pluggable evictor to evict entries based on the number of entries that exist instead of based on time.

The following optional pluggable evictors provide some commonly used algorithms for deciding which entries to evict when a BackingMap grows beyond some size limit.

- The LRUevictor evictor uses a least recently used (LRU) algorithm to decide which entries to evict when the BackingMap exceeds a maximum number of entries.
- The LFUEvictor evictor uses a least frequently used (LFU) algorithm to decide which entries to evict when the BackingMap exceeds a maximum number of entries.

The BackingMap informs an evictor as entries are created, modified, or removed in a transaction. The BackingMap keeps track of these entries and chooses when to evict one or more entries from the BackingMap instance.

A BackingMap instance has no configuration information for a maximum size. Instead, evictor properties are set to control the evictor behavior. Both the LRUevictor and the LFUEvictor have a maximum size property that is used to cause the evictor to begin to evict entries after the maximum size is exceeded. Like the TTL evictor, the LRU and LFU evictors might not immediately evict an entry when the maximum number of entries is reached to minimize impact on performance.

If the LRU or LFU eviction algorithm is not adequate for a particular application, you can write your own evictors to create your eviction strategy.

Memory-based eviction

Important: Memory-based eviction is only supported on Java Platform, Enterprise Edition Version 5 or later.

All built-in evictors support memory-based eviction that can be enabled on the BackingMap interface by setting the evictionTriggers attribute of BackingMap to MEMORY_USAGE_THRESHOLD. For more information about how to set the evictionTriggers attribute on BackingMap, see BackingMap interface and ObjectGrid descriptor XML file.

Memory-based eviction is based on heap usage threshold. When memory-based eviction is enabled on BackingMap and the BackingMap has any built-in evictor, the usage threshold is set to a default percentage of total memory if the threshold has not been previously set.

When you are using memory-based eviction, you should configure the garbage collection threshold to the same value as their target heap utilization. For example, if the memory-based eviction threshold is set at 50 percent and the garbage collection threshold is at the default 70 percent level, then the heap utilization can go as high as 70 percent. This heap utilization increase occurs because memory-based eviction is only triggered after a garbage collection cycle.

To change the default usage threshold percentage, set the memoryThresholdPercentage property on container and server property file for eXtreme Scale server process. To set the target usage threshold on a client process, you can use the MemoryPoolMXBean.

The memory-based eviction algorithm used by WebSphere eXtreme Scale is sensitive to the behavior of the garbage collection algorithm in use. The best algorithm for memory-based eviction is the IBM default throughput collector. Generation garbage collection algorithms can cause undesired behavior, and so you should not use these algorithms with memory-based eviction.

To change the usage threshold percentage, set the `memoryThresholdPercentage` property on the container and server property files for eXtreme Scale server processes.

During runtime, if the memory usage exceeds the target usage threshold, memory-based evictors start evicting entries and try to keep memory usage below the target usage threshold. However, no guarantee exists that the eviction speed is fast enough to avoid a potential out of memory error if the system runtime continues to quickly consume memory.

OSGi framework overview

OSGi defines a dynamic module system for Java. The OSGi service platform has a layered architecture, and is designed to run on various standard Java profiles. You can start WebSphere eXtreme Scale servers and clients in an OSGi container.

Benefits of running applications in the OSGi container

WebSphere eXtreme Scale OSGi support allows you to deploy the product in the Eclipse Equinox OSGi framework. Previously, if you wanted to update the plug-ins used by eXtreme Scale, you had to restart the Java virtual machine (JVM) to apply the new versions of the plug-ins. With the dynamic update capability that the OSGi framework provides, now you can update the plug-in classes without restarting the JVM. These plug-ins are exported by user bundles as services. WebSphere eXtreme Scale accesses the service or services by looking them up the OSGi registry.

eXtreme Scale containers can be configured to start more easily and dynamically using either the OSGi configuration admin service or with OSGi Blueprint. If you want to deploy a new data grid with its placement strategy, you can do so by creating an OSGi configuration or by deploying a bundle with eXtreme Scale descriptor XML files. With OSGi support, application bundles containing eXtreme Scale configuration data can be installed, started, stopped, updated, and uninstalled without restarting the whole system. With this capability, you can upgrade the application without disrupting the data grid.

Plug-in beans and services can be configured with custom shard scopes, allowing sophisticated integration options with other services running in the data grid. Each plug-in can use OSGi Blueprint rankings to verify that every instance of the plug-in is activated is at the correct version. An OSGi-managed bean (MBean) and `xscmd` utility are provided, which allow you to query the eXtreme Scale plug-in OSGi services and their rankings.

This capability allows administrators to quickly recognize potential configuration and administration errors and upgrade the plug-in service rankings in use by eXtreme Scale .

OSGi bundles

To interact with and deploy plug-ins in the OSGi framework, you must use *bundles*. In the OSGi service platform, a bundle is a Java archive (JAR) file that contains Java code, resources, and a manifest that describes the bundle and its dependencies. The bundle is the unit of deployment for an application. The eXtreme Scale product supports the following bundle types:

Server bundle

The server bundle is the `objectgrid.jar` file and is installed with the eXtreme Scale stand-alone server installation and is required for running eXtreme Scale servers and can also be used for running eXtreme Scale clients, or local, in-memory caches. The bundle ID for the `objectgrid.jar` file is `com.ibm.websphere.xs.server_<version>`, where the version is in the format: `<Version>.<Release>.<Modification>`. For example, the server bundle for eXtreme Scale version 7.1.1 is `com.ibm.websphere.xs.server_7.1.1`.

Client bundle

The client bundle is the `ogclient.jar` file and is installed with the eXtreme Scale stand-alone and client installations and is used to run eXtreme Scale clients or local, in-memory caches. The bundle ID for the `ogclient.jar` file is `com.ibm.websphere.xs.client_<version>`, where the version is in the format: `<Version>.<Release>.<Modification>`. For example, the client bundle for eXtreme Scale version 7.1.1 is `com.ibm.websphere.xs.client_7.1.1`.

Limitations

You cannot restart the eXtreme Scale bundle because you cannot restart the object request broker (ORB) or eXtremeIO (XIO). To restart the eXtreme Scale server, you must restart the OSGi framework.

Cache integration overview

The crucial element that gives WebSphere eXtreme Scale the capability to perform with such versatility and reliability is its application of caching concepts to optimize the persistence and recollection of data in virtually any deployment environment.

Spring cache provider

Spring Framework Version 3.1 introduced a new cache abstraction. With this new abstraction, you can transparently add caching to an existing Spring application. You can use WebSphere eXtreme Scale as the cache provider for the cache abstraction.

Liberty profile

The Liberty profile is a highly composable, fast-to-start, dynamic application server runtime environment.

You install the Liberty profile when you install WebSphere eXtreme Scale with WebSphere Application Server Version 8.5. Because the Liberty profile does not include a Java runtime environment (JRE), you have to install a JRE provided by either Oracle or IBM.

For more information about supported Java environments and locations, see Minimum supported Java levels in the WebSphere Application Server Information Center.

This server supports two models of application deployment:

- Deploy an application by dropping it into the dropins directory.
- Deploy an application by adding it to the server configuration.

The Liberty profile supports a subset of the following parts of the full WebSphere Application Server programming model:

- Web applications
- OSGi applications
- Java Persistence API (JPA)

Associated services such as transactions and security are only supported as far as is required by these application types and by JPA.

Features are the units of capability by which you control the pieces of the runtime environment that are loaded into a particular server. The Liberty profile includes the following main features:

- Bean validation
- Blueprint
- Java API for RESTful Web Services
- Java Database Connectivity (JDBC)
- Java Naming and Directory Interface
- Java Persistence API (JPA)
- JavaServer Faces (JSF)
- JavaServer Pages (JSP)
- Lightweight Directory Access Protocol (LDAP)
- Local connector (for Java Management Extensions (JMX) clients)
- Monitoring
- OSGi JPA (JPA support for OSGi applications)
- Remote connector (for JMX clients)
- Secure Sockets Layer (SSL)
- Security
- Servlet
- Session persistence
- Transaction
- Web application bundle (WAB)
- z/OS[®] security
- z/OS transaction management

You can work with the runtime environment directly, or using the WebSphere Application Server Developer Tools for Eclipse.

On distributed platforms, the Liberty profile provides both a development and an operations environment. On the Mac, it provides a development environment.

Running the Liberty profile with a third-party JRE

When you use a JRE that Oracle provides, special considerations must be taken to run WebSphere eXtreme Scale with the Liberty profile.

Classloader deadlock

You might experience a classloader deadlock which has been worked around using the following JVM_ARGS settings. If you experience a deadlock in BundleLoader logic, add the following arguments:

```
export JVM_ARGS="$JVM_ARGS -XX:+UnlockDiagnosticVMOptions -XX:+UnsyncloadClass"
```

Data caching and the Liberty profile

You can use data caching products with the WebSphere Application Server Liberty profile to develop HTTP sessions, client-server connections through the REST gateway, and manage other cache integration scenarios.

In WebSphere eXtreme Scale, you can use the Liberty profile to connect to the data grid. For example, when you install WebSphere eXtreme Scale with the Liberty profile, you have access to features that you can use to manage HTTP session applications, Java client applications, and REST client applications that are installed in the Liberty profile.

The following features contain information about the main available features. Including a feature in the configuration might cause one or more features to be loaded automatically. Each feature includes a brief description and an example of how the feature is declared

Server feature

The server feature contains the capability for running an eXtreme Scale server, both catalog and container. Add the server feature when you want to run a catalog server in the Liberty profile or when you want to deploy a grid application into the Liberty profile.

wlp_install_root/usr/server/*server_name*/server.xml file

```
<server description="WebSphere eXtreme Scale Server">
  <featureManager>
    <feature>eXtremeScale.server-1.1</feature>
  </featureManager>
  <xsServer/>
</server>
```


Client feature

The client feature contains most of the programming model for eXtreme Scale. Add the client feature when you have an application that is running in the Liberty profile that is going to use eXtreme Scale APIs.

wlp_install_root/usr/server/*wlp_install_root*/server.xml file

```
<server description="WebSphere eXtreme Scale Client">
  <featureManager>
    <feature>eXtremeScale.client-1.1</feature>
  </featureManager>
</server>
```

Web feature

 The web feature is deprecated. Use the webapp feature when you want to replicate HTTP session data for fault tolerance.

The web feature contains the capability to extend the Liberty profile web application. Add the web feature when you want to replicate HTTP session data for fault tolerance.

```
wlp_install_root/usr/server/server_name/server.xml file
<server description="WebSphere eXtreme Scale enabled Web Server">
  <featureManager>
    <feature>eXtremeScale.web-1.1</feature>
  </featureManager>
  <xsWebAppV85/>
</server>
```

8.6+ WebApp feature

The webApp feature contains the capability to extend the Liberty profile web application. Add the webApp feature when you want to replicate HTTP session data for fault tolerance.

```
8.6+
wlp_install_root/usr/server/server_name/server.xml file
<server description="WebSphere eXtreme Scale enabled Web Server">
  <featureManager>
    <feature>eXtremeScale.webApp-1.1</feature>
  </featureManager>
  <xsWebApp/>
</server>
```

WebGrid feature

A Liberty profile server can host a data grid that caches data for applications to replicate HTTP session data for fault tolerance.

```
wlp_install_root/usr/server/server_name/server.xml file
<server description="WebSphere eXtreme Scale enabled Web Server">
  <featureManager>
    <feature>eXtremeScale.webGrid-1.1</feature>
  </featureManager>
  <xsWebGrid/>
</server>
```

8.6+ REST feature

Use the Representational State Transfer (REST) gateway to access simple data grids that are hosted by a collective in the Liberty profile.

8.6+

```
wlp_install_root/usr/server/server_name/server.xml file
<server description="WebSphere eXtreme Scale enabled Web Server">

  <featureManager>
    <feature>eXtremeScale.rest-1.1</feature>
  </featureManager>

  <xsRest/>
</server>
```

8.6+

Dynamic cache feature

A Liberty profile server can host a data grid that caches data for applications that have dynamic cache enabled.

8.6+

```
wlp_install_root/usr/server/server_name/server.xml file
<server description="WebSphere eXtreme Scale enabled Web Server">

  <featureManager>
    <feature>eXtremeScale.dynacacheGrid-1.1</feature>
  </featureManager>

  <xsDynacacheGrid/>
</server>
```

Dynamic cache app feature

The Liberty profile server can host WebSphere eXtreme Scale, which you can configure as a dynamic cache provider with this feature.

See the following example for using the default cache provider in the Liberty profile:

```
wlp_install_root/usr/server/server_name/server.xml file
<server description="WebSphere eXtreme Scale enabled Web Server">

  <featureManager>
    <feature>eXtremeScale.dynacacheapp-1.1</feature>
  </featureManager>

  <xsClientDomain default="production">
    <endpointConfig> production;localhost:2809 </endpointConfig>
  </xsClientDomain>

  <distributedMap id="baseCache" libraryRef="idgenerator" cacheProviderName="WebSphere eXtreme Scale">
    <xsDynacacheApp remoteDomain="production" />
  </distributedMap>

  <distributedMap id="cache01" jndiName="cache01" memorySizeInEntries="2000" createCacheAtServerStartu
    <xsDynacacheApp remoteDomain="production" />
  </distributedMap>
</server>
```

JPA level 2 (L2) cache plug-in

Java

WebSphere eXtreme Scale includes level 2 (L2) cache plug-ins for both OpenJPA and Hibernate Java Persistence API (JPA) providers. When you use one of these plug-ins, your application uses the JPA API. A data grid is introduced between the application and the database, improving response times.

Using eXtreme Scale as an L2 cache provider increases performance when you are reading and querying data and reduces load to the database. WebSphere eXtreme Scale has advantages over built-in cache implementations because the cache is automatically replicated between all processes. When one client caches a value, all other clients are able to use the cached value that is locally in-memory.

You can configure the topology and properties for the L2 cache provider in the `persistence.xml` file. For more information about configuring these properties, see JPA cache configuration properties for Hibernate Version 4.0.

Tip: The JPA L2 cache plug-in requires an application that uses the JPA APIs. If you want to use WebSphere eXtreme Scale APIs to access a JPA data source, use the JPA loader. For more information, see “JPA Loaders” on page 69.

JPA L2 cache topology considerations

The following factors affect which type of topology to configure:

1. How much data do you expect to be cached?

- If the data can fit into a single JVM heap, use the “Embedded topology” on page 41 or “Intra-domain topology” on page 40.
- If the data cannot fit into a single JVM heap, use the “Embedded, partitioned topology” on page 42, or “Remote topology” on page 43

2. What is the expected read-to-write ratio?

The read-to-write ratio affects the performance of the L2 cache. Each topology handles read and write operations differently.

- “Embedded topology” on page 41: local read, remote write
- “Intra-domain topology” on page 40: local read, local write
- “Embedded, partitioned topology” on page 42: Partitioned: remote read, remote write
- “Remote topology” on page 43: remote read, remote write.

Applications that are mostly read-only should use embedded and intra-domain topologies when possible. Applications that do more writing should use intra-domain topologies.

3. What is percentage of data is queried versus found by a key?

When enabled, query operations make use of the JPA query cache. Enable the JPA query cache for applications with high read to write ratios only, for example when you are approaching 99% read operations. If you use JPA query cache, you must use the “Embedded topology” on page 41 or “Intra-domain topology” on page 40.

The find-by-key operation fetches a target entity if the target entity does not have any relationship. If the target entity has relationships with the EAGER fetch type, these relationships are fetched along with the target entity. In JPA data cache, fetching these relationships causes a few cache hits to get all the relationship data.

4. What is the tolerated staleness level of the data?

In a system with few JVMs, data replication latency exists for write operations. The goal of the cache is to maintain an ultimate synchronized data view across

all JVMs. When you are using the intra-domain topology, a data replication delay exists for write operations. Applications using this topology must be able to tolerate stale reads and simultaneous writes that might overwrite data.

Intra-domain topology

With an intra-domain topology, primary shards are placed on every container server in the topology. These primary shards contain the full set of data for the partition. Any of these primary shards can also complete cache write operations. This configuration eliminates the bottleneck in the embedded topology where all the cache write operations must go through a single primary shard.

In an intra-domain topology, no replica shards are created, even if you have defined replicas in your configuration files. Each redundant primary shard contains a full copy of the data, so each primary shard can also be considered as a replica shard. This configuration uses a single partition, similar to the embedded topology.

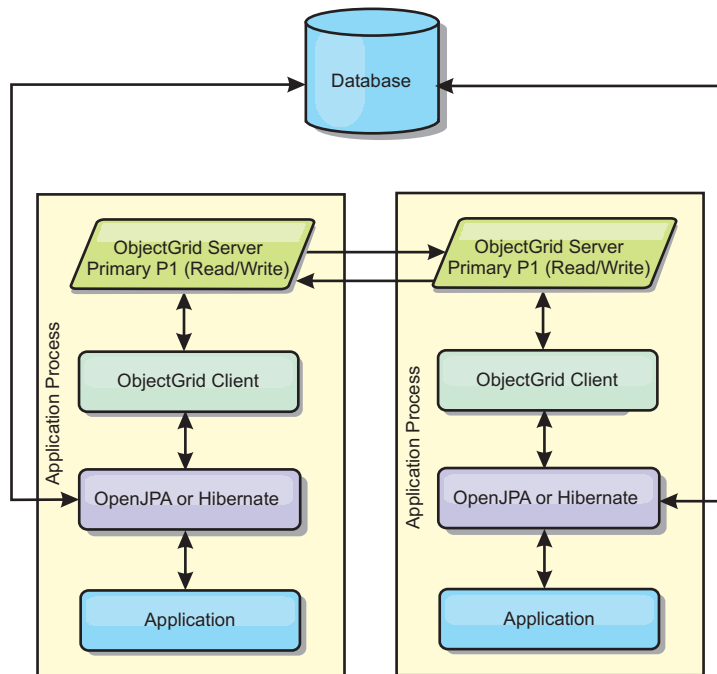


Figure 14. JPA intra-domain topology

Related JPA cache configuration properties for the intra-domain topology:

`ObjectGridName=objectgrid_name, ObjectGridType=EMBEDDED, PlacementScope=CONTAINER_SCOPE, PlacementScopeTopology=HUB | RING`

Advantages:

- Cache reads and updates are local.
- Simple to configure.

Limitations:

- This topology is best suited for when the container servers can contain the entire set of partition data.
- Replica shards, even if they are configured, are never placed because every container server hosts a primary shard. However, all the primary shards are replicating with the other primary shards, so these primary shards become replicas of each other.

Embedded topology

Tip: Consider using an intra-domain topology for the best performance.

An embedded topology creates a container server within the process space of each application. OpenJPA and Hibernate read the in-memory copy of the cache directly and write to all of the other copies. You can improve the write performance by using asynchronous replication. This default topology performs best when the amount of cached data is small enough to fit in a single process. With an embedded topology, create a single partition for the data.

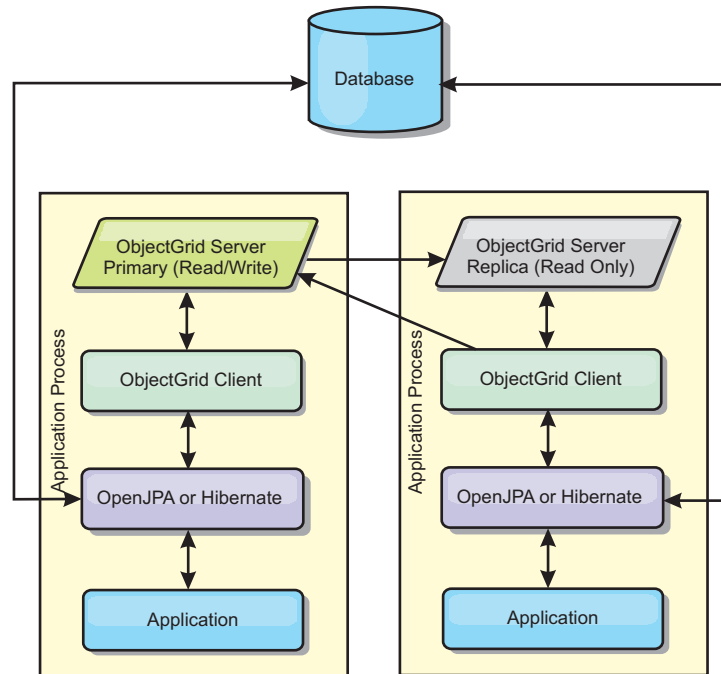


Figure 15. JPA embedded topology

Related JPA cache configuration properties for the embedded topology:

```
ObjectGridName=objectgrid_name,ObjectGridType=EMBEDDED,MaxNumberOfReplicas=num_replicas,ReplicaMode=SYNC | ASYNC | NONE
```

Advantages:

- All cache reads are fast, local accesses.
- Simple to configure.

Limitations:

- Amount of data is limited to the size of the process.
- All cache updates are sent through one primary shard, which creates a bottleneck.

Embedded, partitioned topology

Tip: Consider using an intra-domain topology for the best performance.

CAUTION:

Do not use the JPA query cache with an embedded partitioned topology. The query cache stores query results that are a collection of entity keys. The query cache fetches all entity data from the data cache. Because the data cache is divided up between multiple processes, these additional calls can negate the benefits of the L2 cache.

When the cached data is too large to fit in a single process, you can use the embedded, partitioned topology. This topology divides the data over multiple processes. The data is divided between the primary shards, so each primary shard contains a subset of the data. You can still use this option when database latency is high.

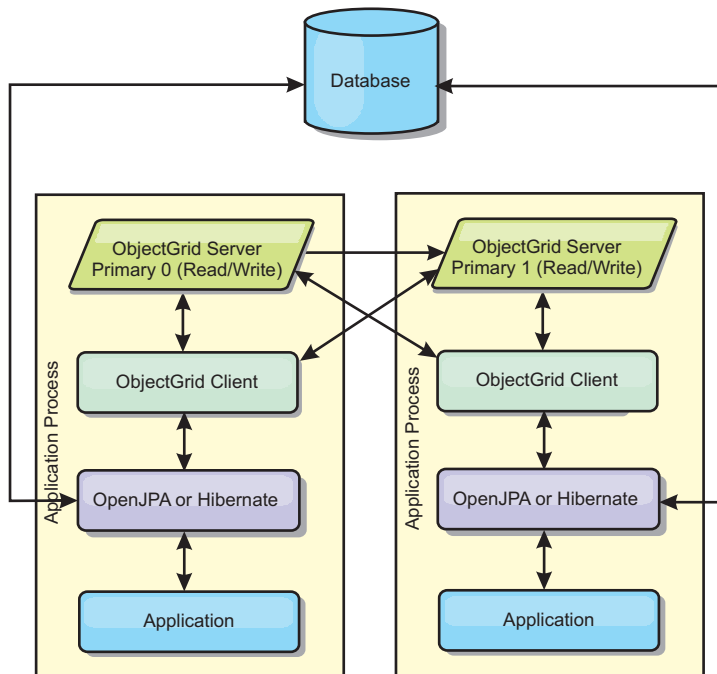


Figure 16. JPA embedded, partitioned topology

Related JPA cache configuration properties for the embedded, partitioned topology:

```
ObjectGridName=objectgrid_name,ObjectGridType=EMBEDDED_PARTITION,ReplicaMode=SYNC | ASYNC | NONE,  
NumberOfPartitions=num_partitions,ReplicaReadEnabled=TRUE | FALSE
```

Advantages:

- Stores large amounts of data.
- Simple to configure
- Cache updates are spread over multiple processes.

Limitation:

- Most cache reads and updates are remote.

For example, to cache 10 GB of data with a maximum of 1 GB per JVM, 10 Java virtual machines are required. The number of partitions must therefore be set to 10

or more. Ideally, the number of partitions must be set to a prime number where each shard stores a reasonable amount of memory. Usually, the `numberOfPartitions` setting is equal to the number of Java virtual machines. With this setting, each JVM stores one partition. If you enable replication, you must increase the number of Java virtual machines in the system. Otherwise, each JVM also stores one replica partition, which consumes as much memory as a primary partition.

Read about sizing memory and partition count calculation in the *Administration Guide* to maximize the performance of your chosen configuration.

For example, in a system with four Java virtual machines, and the `numberOfPartitions` setting value of 4, each JVM hosts a primary partition. A read operation has a 25 percent chance of fetching data from a locally available partition, which is much faster compared to getting data from a remote JVM. If a read operation, such as running a query, needs to fetch a collection of data that involves 4 partitions evenly, 75 percent of the calls are remote and 25 percent of the calls are local. If the `ReplicaMode` setting is set to either `SYNC` or `ASync` and the `ReplicaReadEnabled` setting is set to `true`, then four replica partitions are created and spread across four Java virtual machines. Each JVM hosts one primary partition and one replica partition. The chance that the read operation runs locally increases to 50 percent. The read operation that fetches a collection of data that involves four partitions evenly has 50 percent remote calls and 50 percent local calls. Local calls are much faster than remote calls. Whenever remote calls occur, the performance drops.

Remote topology

CAUTION:

Do not use the JPA query cache with a remote topology. The query cache stores query results that are a collection of entity keys. The query cache fetches all entity data from the data cache. Because the data cache is remote, these additional calls can negate the benefits of the L2 cache.

Tip: Consider using an intra-domain topology for the best performance.

A remote topology stores all of the cached data in one or more separate processes, reducing memory use of the application processes. You can take advantage of distributing your data over separate processes by deploying a partitioned, replicated eXtreme Scale data grid. As opposed to the embedded and embedded partitioned configurations described in the previous sections, if you want to manage the remote data grid, you must do so independent of the application and JPA provider.

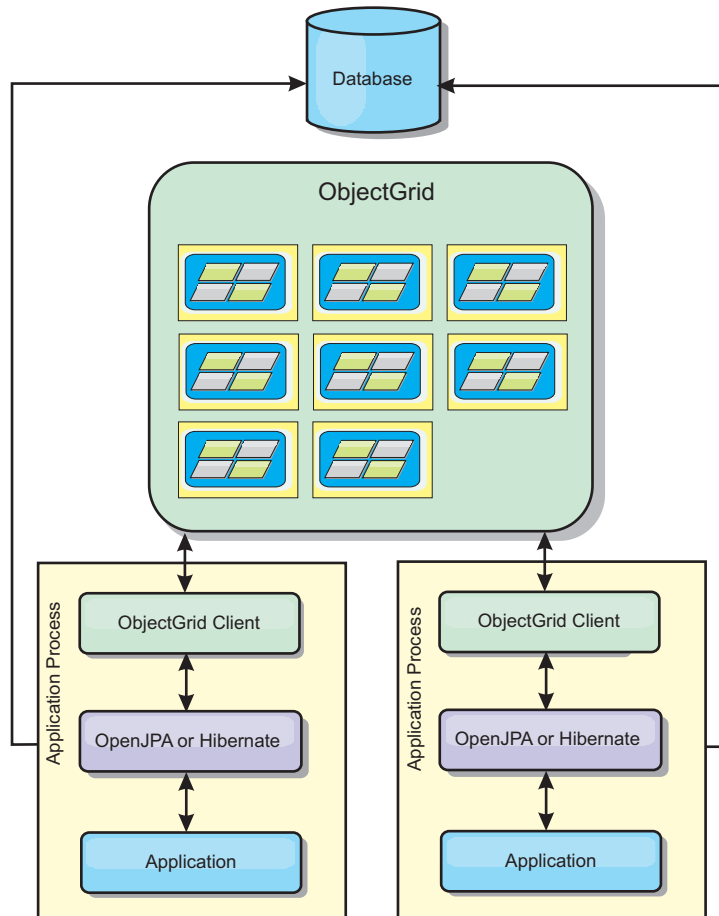


Figure 17. JPA remote topology

Related JPA cache configuration properties for the remote topology:

`ObjectGridName=objectgrid_name, ObjectGridType=REMOTE, AllowNearCache=TRUE`

Note: The `AllowNearCache` property is optional. If it is not included in the configuration, the default value is `FALSE`. This property is only used by a remote object grid type as long as the remote object grid server is also enabled for near caching as defined in the `ObjectGrid` descriptor XML file. To enable the L2 cache provider for near caching, set the property `AllowNearCache` is set to `TRUE`.

The `REMOTE` `ObjectGrid` type does not require any property settings because the `ObjectGrid` and deployment policy is defined separately from the JPA application. The JPA cache plug-in remotely connects to an existing remote `ObjectGrid`.

Because all interaction with the `ObjectGrid` is remote, this topology has the slowest performance among all `ObjectGrid` types.

Advantages:

- Stores large amounts of data.
- Application process is free of cached data.
- Cache updates are spread over multiple processes.
- Flexible configuration options.

Limitation:

- All cache reads and updates are remote.

HTTP session management

The session replication manager that is shipped with WebSphere eXtreme Scale can work with the default session manager in WebSphere Application Server. Session data is replicated from one process to another process to support user session data high availability.

Features

The session manager has been designed so that it can run in any Java Platform, Enterprise Edition Version 6 or later container. Because the session manager does not have any dependencies on WebSphere APIs, it can support various versions of WebSphere Application Server, as well as vendor application server environments.

The HTTP session manager provides session replication capabilities for an associated application. The session replication manager works with the session manager for the web container. Together, the session manager and web container create HTTP sessions and manage the life cycles of HTTP sessions that are associated with the application. These life cycle management activities include: the invalidation of sessions based on a timeout or an explicit servlet or JavaServer Pages (JSP) call and the invocation of session listeners that are associated with the session or the web application. The session manager persists its sessions in a fully replicated, clustered and partitioned data grid. The use of the WebSphere eXtreme Scale session manager enables the session manager to provide HTTP session failover support when application servers are shut down or end unexpectedly. The session manager can also work in environments that do not support affinity, when affinity is not enforced by a load balancer tier that sprays requests to the application server tier.

Usage scenarios

The session manager can be used in the following scenarios:

- In environments that use application servers at different versions of WebSphere Application Server, such as in a migration scenario.
- In deployments that use application servers from different vendors. For example, an application that is being developed on open source application servers and that is hosted on WebSphere Application Server. Another example is an application that is being promoted from staging to production. Seamless migration of these application server versions is possible while all HTTP sessions are live and being serviced.
- In environments that require the user to persist sessions with higher quality of service (QoS) levels. Session availability is better guaranteed during server failover than default WebSphere Application Server QoS levels.
- In an environment where session affinity cannot be guaranteed, or environments in which affinity is maintained by a vendor load balancer. With a vendor load balancer, the affinity mechanism must be customized to that load balancer.
- In any environment to offload the processing required for session management and storage to an external Java process.
- In multiple cells to enable session failover between cells.
- In multiple data centers or multiple zones.

How the session manager works

The session replication manager uses a session listener to listen on the changes of session data. The session replication manager persists the session data into an ObjectGrid instance either locally or remotely. You can add the session listener and servlet filter to every web module in your application with tooling that ships with WebSphere eXtreme Scale. You can also manually add these listeners and filters to the web deployment descriptor of your application.

This session replication manager works with each vendor web container session manager to replicate session data across Java virtual machines. When the original server dies, users can retrieve session data from other servers.

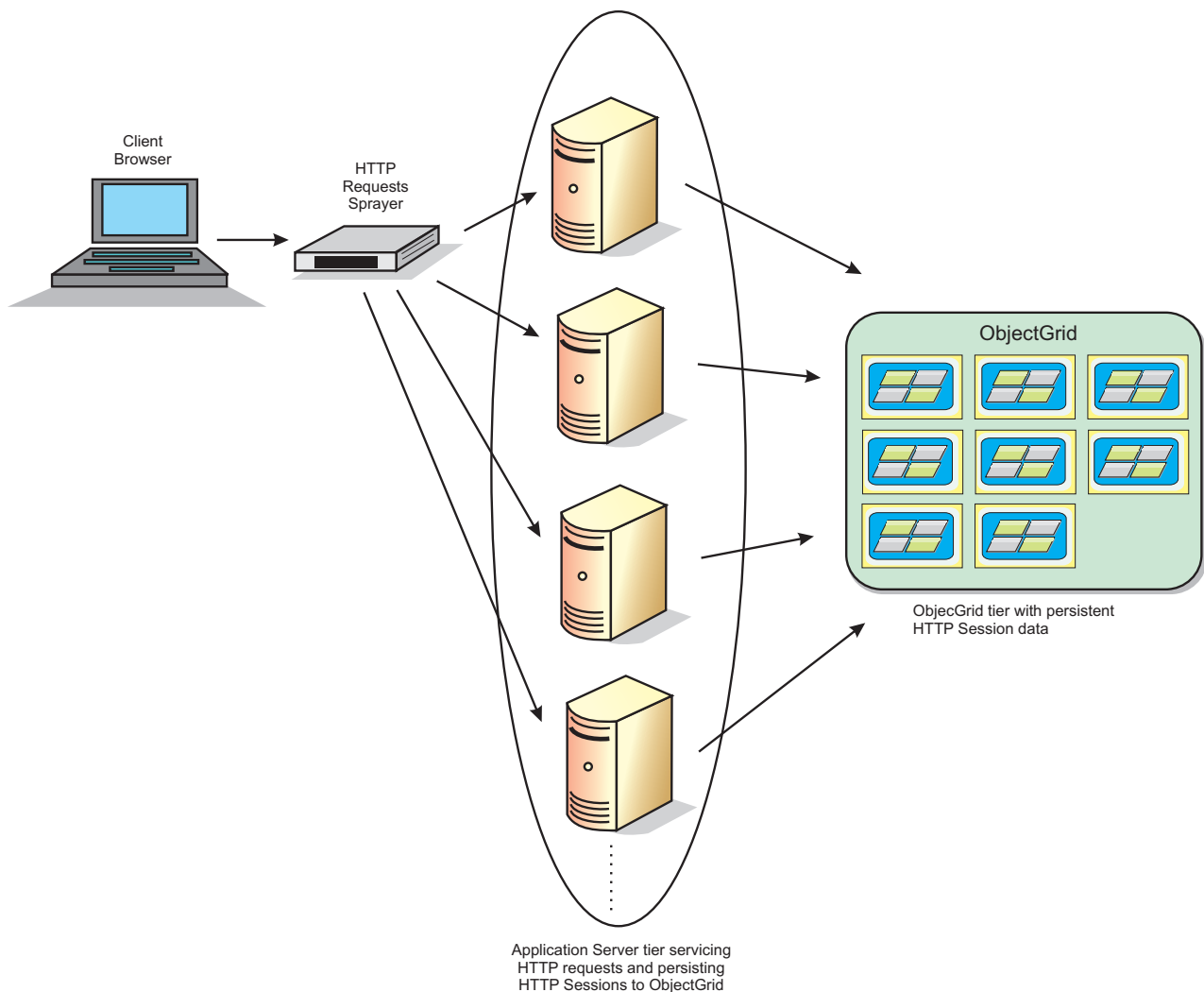


Figure 18. HTTP session management topology with a remote container configuration

Deployment topologies

The session manager can be configured using two different dynamic deployment scenarios:

Embedded, network attached eXtreme Scale container servers

In this scenario, the eXtreme Scale servers are collocated in the same processes as the servlets. The session manager can communicate directly to

the local ObjectGrid instance, avoiding costly network delays. This scenario is preferable when running with affinity and performance is critical.

Remote, network attached eXtreme Scale container servers

In this scenario, the eXtreme Scale servers run in external processes from the process in which the servlets run. The session manager communicates with a remote eXtreme Scale server grid. This scenario is preferable when the web container tier does not have the memory to store the session data. The session data is offloaded to a separate tier, which results in lower memory usage on the web container tier. Higher latency occurs because the data is in a remote location.

Generic embedded container startup

eXtreme Scale automatically starts an embedded ObjectGrid container inside any application-server process when the web container initializes the session listener or servlet filter, if the `objectGridType` property is set to `EMBEDDED`. See Servlet context initialization parameters for details.

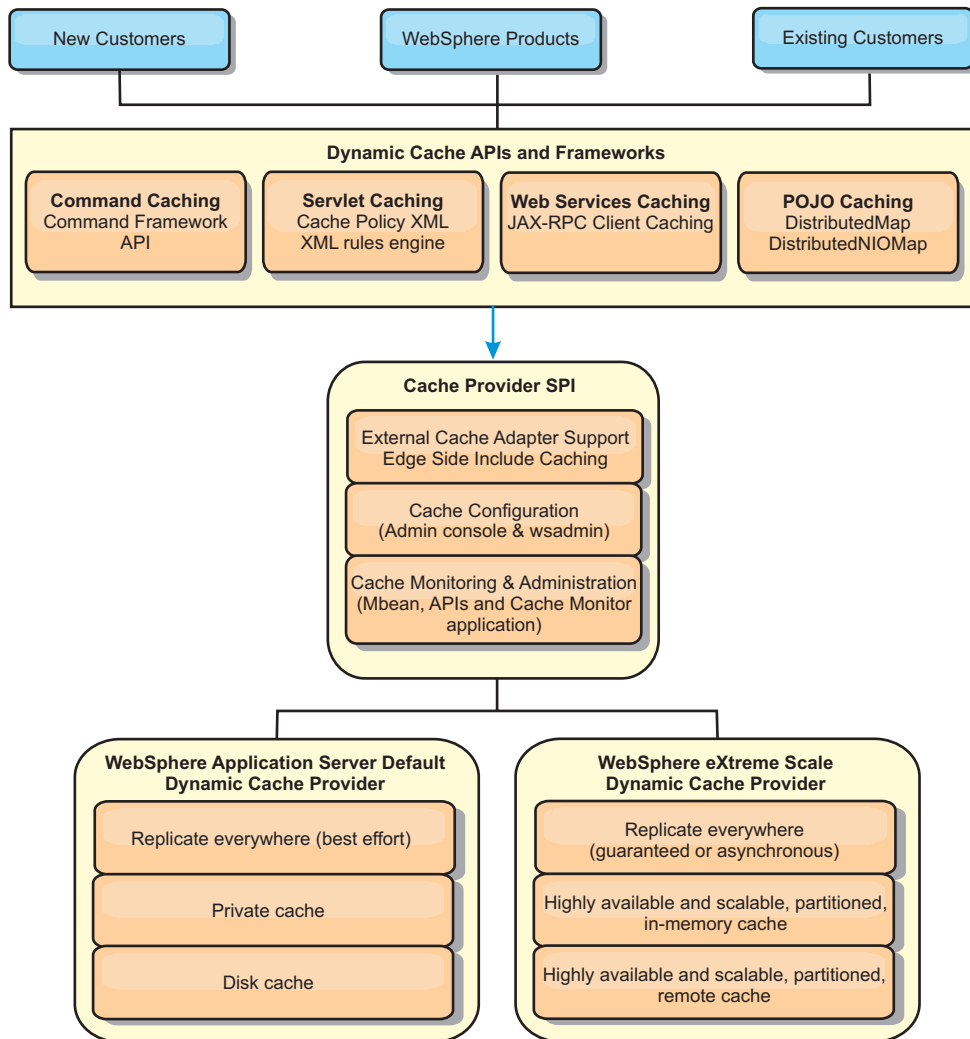
You are not required to package an `ObjectGrid.xml` file and `objectGridDeployment.xml` file into your web application WAR or EAR file. The default `ObjectGrid.xml` and `objectGridDeployment.xml` files are packaged in the product JAR file. Dynamic maps are created for various web application contexts by default. Static eXtreme Scale maps continue to be supported.

This approach for starting embedded ObjectGrid containers applies to any type of application server. The approaches involving a WebSphere Application Server component or WebSphere Application Server Community Edition GBean are deprecated.

Dynamic cache provider overview

The WebSphere Application Server provides a dynamic cache service that is available to deployed Java EE applications. This service is used to cache data such as output from servlet, JSP, or commands, and object data programmatically specified within an enterprise application with the `DistributedMap` APIs. .

Initially, the only service provider for the dynamic cache service was the default dynamic cache engine that is built into WebSphere Application Server. You can also specify WebSphere eXtreme Scale to be the cache provider for any cache instance. By setting up this capability, you can enable applications that use the dynamic cache service, to use the features and performance capabilities of WebSphere eXtreme Scale.



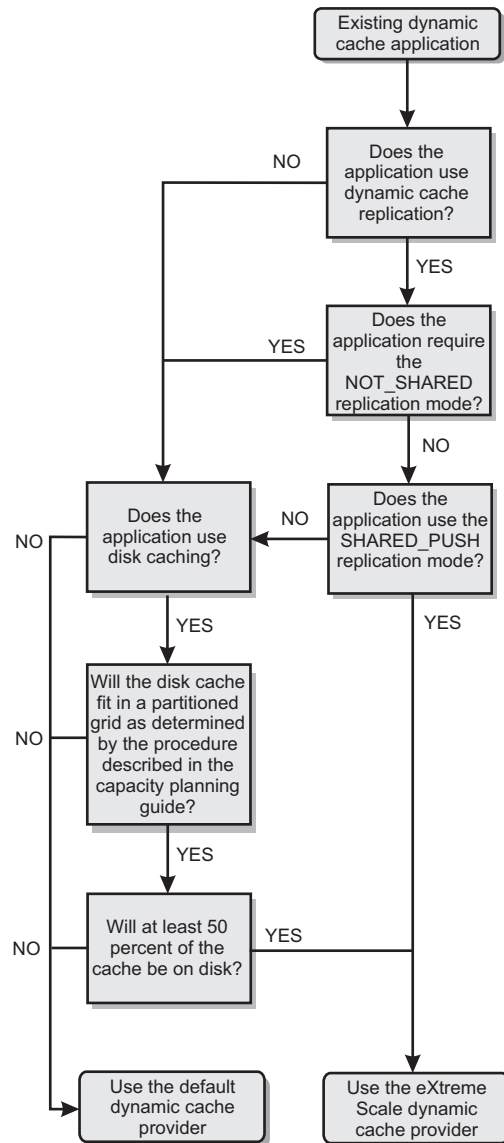
You can install and configure the dynamic cache provider as described in [Configuring the default dynamic cache instance \(baseCache\)](#).

Deciding how to use WebSphere eXtreme Scale

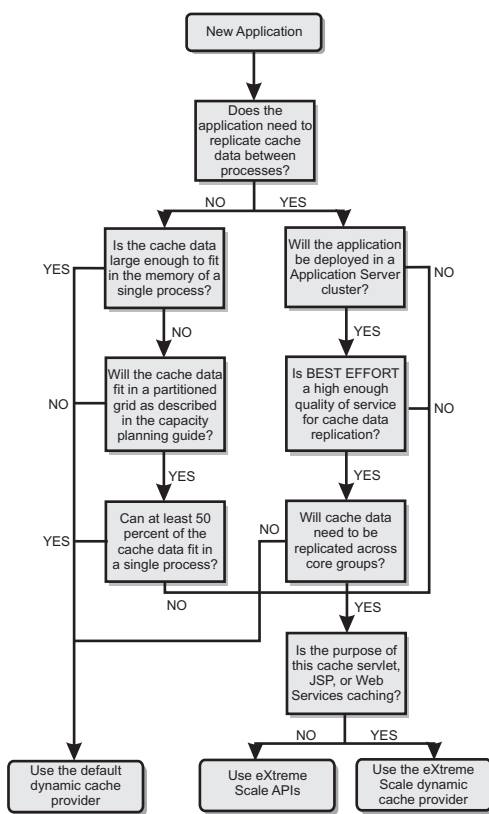
The available features in WebSphere eXtreme Scale significantly increase the distributed capabilities of the dynamic cache service beyond what is offered by the default dynamic cache provider and data replication service. With eXtreme Scale, you can create caches that are truly distributed between multiple servers, rather than just replicated and synchronized between the servers. Also, eXtreme Scale caches are transactional and highly available, ensuring that each server sees the same contents for the dynamic cache service. WebSphere eXtreme Scale offers a higher quality of service for cache replication provided via DRS.

However, these advantages do not mean that the eXtreme Scale dynamic cache provider is the right choice for every application. Use the decision trees and feature comparison matrix below to determine what technology fits your application best.

Decision tree for migrating existing dynamic cache applications



Decision tree for choosing a cache provider for new applications



Feature comparison

Table 1. Feature comparison

Cache features	Default provider	eXtreme Scale provider	eXtreme Scale API
Local, in-memory caching	Yes	via Near-cache capability	via Near-cache capability
Distributed caching	via DRS	Yes	Yes
Linearly scalable	No	Yes	Yes
Reliable replication (synchronous)	No	Yes	Yes
Disk overflow	Yes	N/A	N/A
Eviction	LRU/TTL/heap-based	LRU/TTL (per partition)	LRU/TTL (per partition)
Invalidation	Yes	Yes	Yes
Relationships	Dependency / template ID relationships	Yes	No (other relationships are possible)
Non-key lookups	No	No	via Query and index
Back-end integration	No	No	via Loaders
Transactional	No	Yes	Yes
Key-based storage	Yes	Yes	Yes

Table 1. Feature comparison (continued)

Cache features	Default provider	eXtreme Scale provider	eXtreme Scale API
Events and listeners	Yes	No	Yes
WebSphere Application Server integration	Single cell only	Multiple cell	Cell independent
Java Standard Edition support	No	Yes	Yes
Monitoring and statistics	Yes	Yes	Yes
Security	Yes	Yes	Yes

For a more detailed description on how eXtreme Scale distributed caches work, see “Planning the topology” on page 147.

Note: An eXtreme Scale distributed cache can only store entries where the key and the value both implement the `java.io.Serializable` interface.

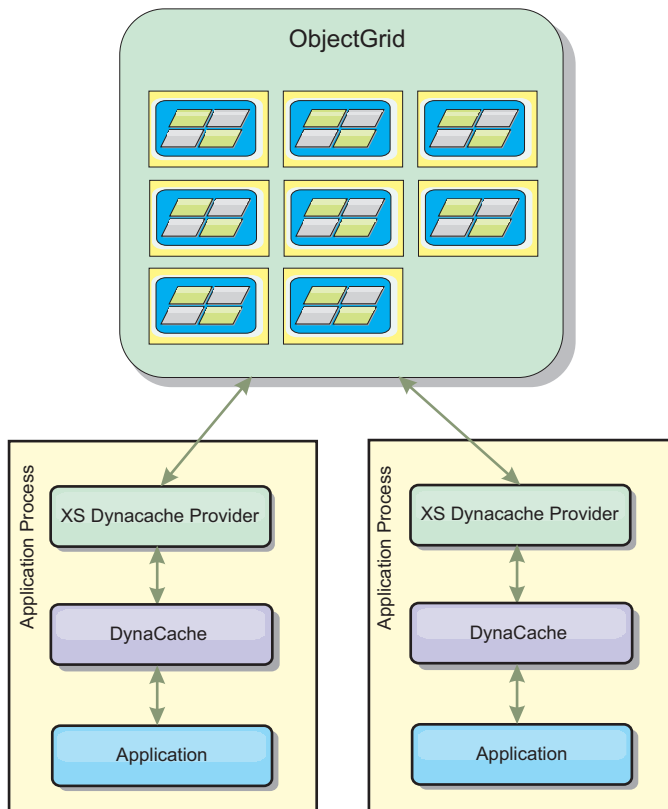
Topology

Deprecated:  **8.6+** The local, embedded, and embedded-partitioned topology types are deprecated.

A dynamic cache service that is created with eXtreme Scale as the provider can be deployed in a remote topology.

Remote topology

The remote topology eliminates the need for a disk cache. All of the cache data is stored outside of WebSphere Application Server processes. WebSphere eXtreme Scale supports standalone container processes for cache data. These container processes have a lower overhead than a WebSphere Application Server process and are also not limited to using a particular Java Virtual Machine (JVM). For example, the data for a dynamic cache service being accessed by a 32-bit WebSphere Application Server process could be located in an eXtreme Scale container process running on a 64-bit JVM. This allows users to use the increased memory capacity of 64-bit processes for caching, without incurring the additional overhead of 64-bit for application server processes. The remote topology is shown in the following image:



Dynamic cache engine and eXtreme Scale functional differences

Users should not notice a functional difference between the two caches except that the WebSphere eXtreme Scale backed caches do not support disk offload or statistics and operations related to the size of the cache in memory.

No appreciable difference exists in the results returned by most dynamic cache API calls, regardless of whether you are using the default dynamic cache provider or the eXtreme Scale cache provider. For some operations, you cannot emulate the behavior of the dynamic cache engine with eXtreme Scale.

Dynamic cache statistics

You can retrieve statistical data for a WebSphere eXtreme Scale dynamic cache with eXtreme Scale monitoring tooling. For more information, see [Monitoring](#).

MBean calls

The WebSphere eXtreme Scale dynamic cache provider does not support disk caching. Any MBean calls relating to disk caching do not work.

Dynamic cache replication policy mapping

The eXtreme Scale dynamic cache provider's remote topology supports a replication policy that most closely matches the SHARED_PULL and SHARED_PUSH_PULL policy (using the terminology used by the default WebSphere Application Server dynamic cache provider). In an eXtreme Scale dynamic cache, the distributed state of the cache is consistent between all the servers.

8.6+ Global index invalidation

You can use a global index to improve invalidation efficiency in large partitioned environments; for example, more than 40 partitions. Without the global index feature, the dynamic cache template and dependency invalidation processing must send remote agent requests to all partitions, which results in slower performance. When you configure a global index, invalidation agents are sent only to applicable partitions that contain cache entries that are related to the Template or Dependency ID. The potential performance improvement is greater in environments with large numbers of partitions configured. You can configure a global index with the Dependency ID and Template ID indexes, which are available in the example dynamic cache objectGrid descriptor XML files. For more information, see “Configuring an Enterprise Data Grid in a stand-alone environment for dynamic caching” on page 274.

Near cache

You can configure a dynamic cache instance to create and maintain a near cache, which resides locally within the application server JVM. The near cache contains a subset of the entries that are contained within the remote dynamic cache instance. You can configure a near cache instance with a `dynacache-nearCache-ObjectGrid.xml` file. For more information, see “Configuring an Enterprise Data Grid in a stand-alone environment for dynamic caching” on page 274. There are also custom properties for tuning the near-cache. For more information, see Dynamic cache custom properties.

Multi-master replication

A dynamic cache instance can be configured to support a multi-master replication topology. Collision arbitration is important for any replication topology. For more information, see “Design considerations for multi-master replication” on page 176. The sample `objectgrid.xml` files that are delivered for the dynamic cache grid configuration are configured with a default collision arbiter: `<bean`

```
id="CollisionArbiter"
```

```
className="com.ibm.ws.objectgrid.dynacache.arbiters.DynacacheCollisionArbiter"/>.
```

The arbiter is invoked to resolve collisions during replication. It first resolves collisions that result from remove and invalidation events, applying these actions over any other event. For all other events, the changes from the lexically lowest named catalog service domain will be applied. For more information, see “Planning multiple data center topologies” on page 169.

Note: Dynamic cache grid users of WebSphere Portal Server or WebSphere Commerce Server may have defined multiple cache instances within their WebSphere Application Server configuration. If you decide to enable multi-master replication for the eXtreme Scale servers, then this configuration will only affect those cache instances that are defined to use the eXtreme Scale servers as the dynamic cache provider, and will not affect the cache instances defined to use the default WebSphere Application Server dynamic cache provider.

Additional information

- Dynamic cache Redbook
- Dynamic cache documentation
 - WebSphere Application Server 7.0
- DRS documentation

Database integration: Write-behind, in-line, and side caching

WebSphere eXtreme Scale is used to front a traditional database and eliminate read activity that is normally pushed to the database. A coherent cache can be used with an application directly or indirectly using an object relational mapper. The coherent cache can then offload the database or backend from reads. In a slightly more complex scenario, such as transactional access to a data set where only some of the data requires traditional persistence guarantees, filtering can be used to offload even write transactions.

You can configure WebSphere eXtreme Scale to function as a highly flexible in-memory database processing space. However, WebSphere eXtreme Scale is not an object relational mapper (ORM). It does not know where the data in the data grid came from. An application or an ORM can place data in an eXtreme Scale server. It is the responsibility of the source of the data to make sure that it stays consistent with the database where data originated. This means eXtreme Scale cannot invalidate data that is pulled from a database automatically. The application or mapper must provide this function and manage the data stored in eXtreme Scale.

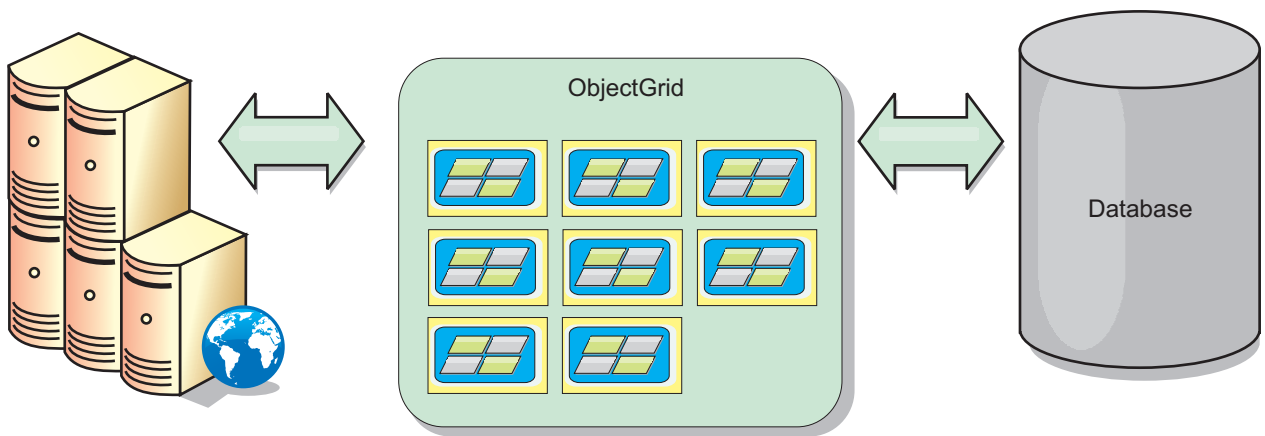


Figure 19. ObjectGrid as a database buffer

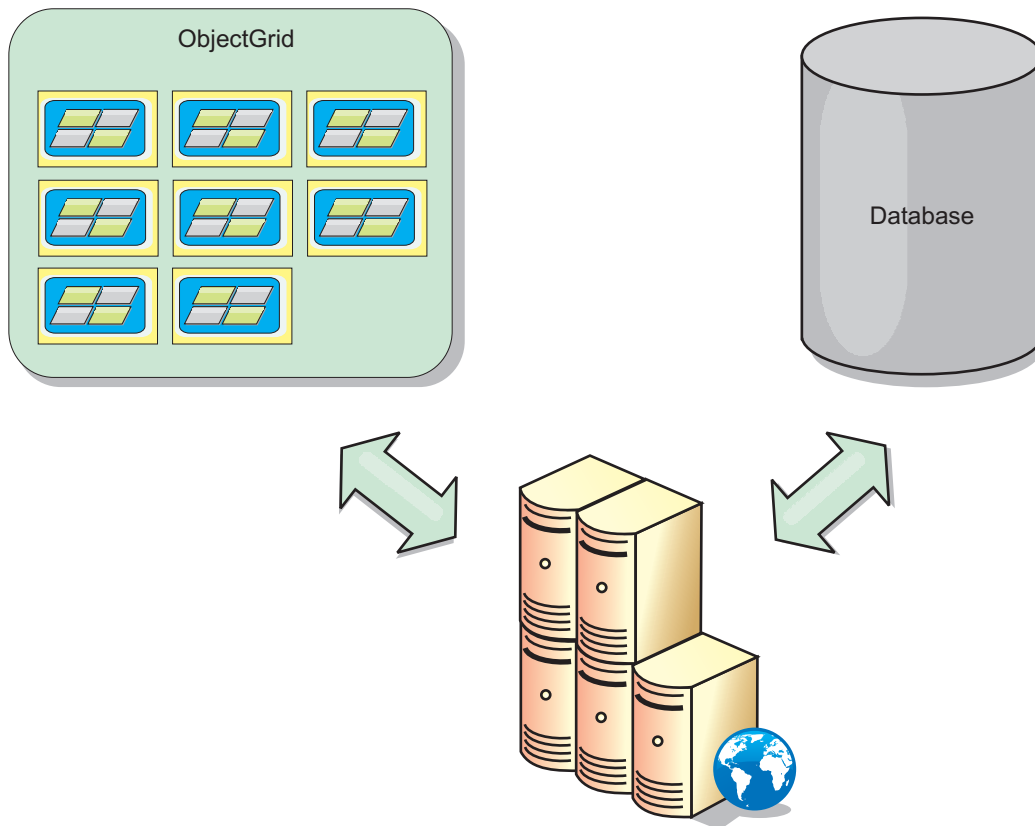


Figure 20. ObjectGrid as a side cache

Sparse and complete cache

WebSphere eXtreme Scale can be used as a sparse cache or a complete cache. A sparse cache only keeps a subset of the total data, while a complete cache keeps all of the data. and can be populated lazily, as the data is needed. Sparse caches are normally accessed using keys (instead of indexes or queries) because the data is only partially available.

Sparse cache

When a key is not present in a sparse cache, or the data is not available and a cache miss occurs, the next tier is invoked. The data is fetched, from a database for example, and is inserted into the data grid cache tier. If you are using a query or index, only the currently loaded values are accessed and the requests are not forwarded to the other tiers.

Complete cache

A complete cache contains all of the required data and can be accessed using non-key attributes with indexes or queries. A complete cache is preloaded with data from the database before the application tries to access the data. A complete cache can function as a database replacement after data is loaded. Because all of the data is available, queries and indexes can be used to find and aggregate data.

Side cache

When WebSphere eXtreme Scale is used as a side cache, the back end is used with the data grid.

Side cache

You can configure the product as a side cache for the data access layer of an application. In this scenario, WebSphere eXtreme Scale is used to temporarily store objects that would normally be retrieved from a back-end database. Applications check to see if the data grid contains the data. If the data is in the data grid, the data is returned to the caller. If the data does not exist, the data is retrieved from the back-end database. The data is then inserted into the data grid so that the next request can use the cached copy. The following diagram illustrates how WebSphere eXtreme Scale can be used as a side-cache with an arbitrary data access layer such as OpenJPA or Hibernate.

Side cache plug-ins for Hibernate and OpenJPA

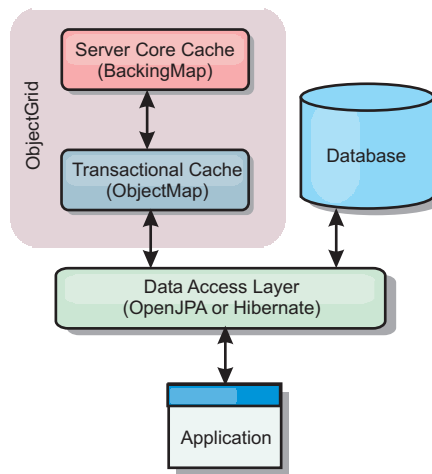


Figure 21. Side cache

Cache plug-ins for both OpenJPA and Hibernate are included in WebSphere eXtreme Scale, so you can use the product as an automatic side-cache. Using WebSphere eXtreme Scale as a cache provider increases performance when reading and querying data and reduces load to the database. There are advantages that WebSphere eXtreme Scale has over built-in cache implementations because the cache is automatically replicated between all processes. When one client caches a value, all other clients can use the cached value.

In-line cache

You can configure in-line caching for a database back end or as a side cache for a database. In-line caching uses eXtreme Scale as the primary means for interacting with the data. When eXtreme Scale is used as an in-line cache, the application interacts with the back end using a Loader plug-in.

In-line cache

When used as an in-line cache, WebSphere eXtreme Scale interacts with the back end using a Loader plug-in. This scenario can simplify data access because applications can access the eXtreme Scale APIs directly. Several different caching

scenarios are supported in eXtreme Scale to make sure the data in the cache and the data in the back end are synchronized. The following diagram illustrates how an in-line cache interacts with the application and back end.

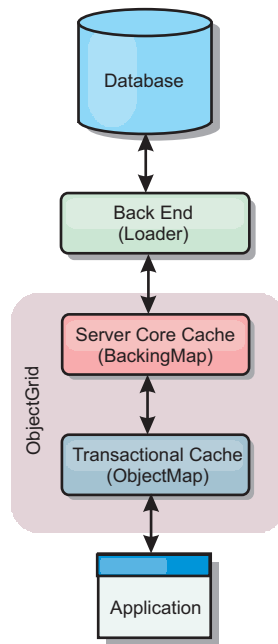


Figure 22. In-line cache

The in-line caching option simplifies data access because it allows applications to access the eXtreme Scale APIs directly. WebSphere eXtreme Scale supports several in-line caching scenarios, as follows.

- Read-through
- Write-through
- Write-behind

Read-through caching scenario

A read-through cache is a sparse cache that lazily loads data entries by key as they are requested. This is done without requiring the caller to know how the entries are populated. If the data cannot be found in the eXtreme Scale cache, eXtreme Scale will retrieve the missing data from the Loader plug-in, which loads the data from the back-end database and inserts the data into the cache. Subsequent requests for the same data key will be found in the cache until it is removed, invalidated or evicted.

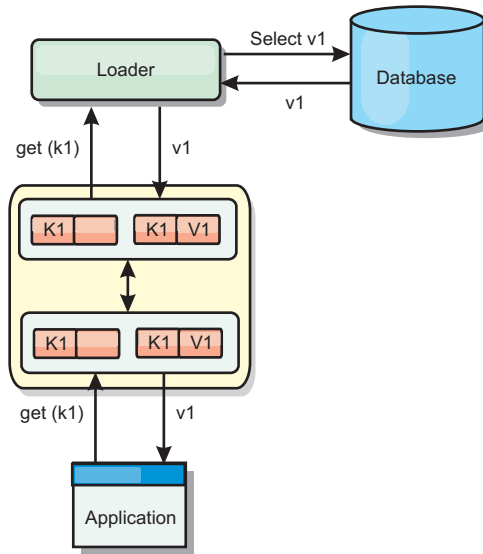


Figure 23. Read-through caching

Write-through caching scenario

In a write-through cache, every write to the cache synchronously writes to the database using the Loader. This method provides consistency with the back end, but decreases write performance since the database operation is synchronous. Since the cache and database are both updated, subsequent reads for the same data will be found in the cache, avoiding the database call. A write-through cache is often used in conjunction with a read-through cache.

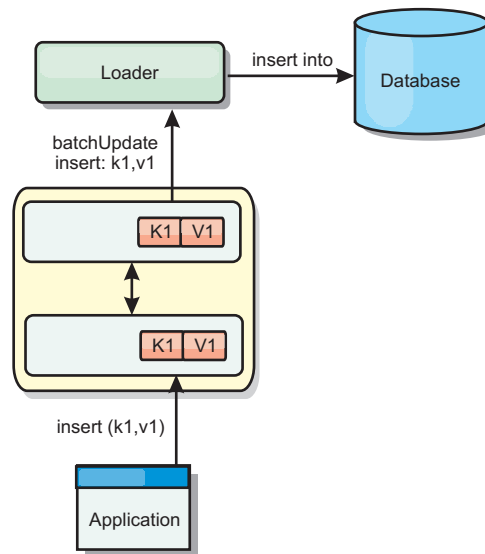


Figure 24. Write-through caching

Write-behind caching scenario

Database synchronization can be improved by writing changes asynchronously. This is known as a write-behind or write-back cache. Changes that would normally be written synchronously to the loader are instead buffered in eXtreme Scale and written to the database using a background thread. Write performance is

significantly improved because the database operation is removed from the client transaction and the database writes can be compressed.

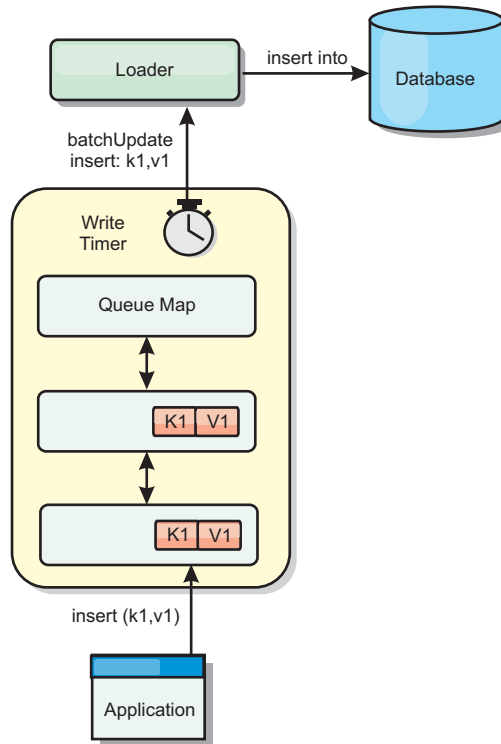


Figure 25. Write-behind caching

Write-behind caching

Java

You can use write-behind caching to reduce the overhead that occurs when updating a database you are using as a back end.

Write-behind caching overview

Write-behind caching asynchronously queues updates to the Loader plug-in. You can improve performance by disconnecting updates, inserts, and removes for a map, the overhead of updating the back-end database. The asynchronous update is performed after a time-based delay (for example, five minutes) or an entry-based delay (1000 entries).

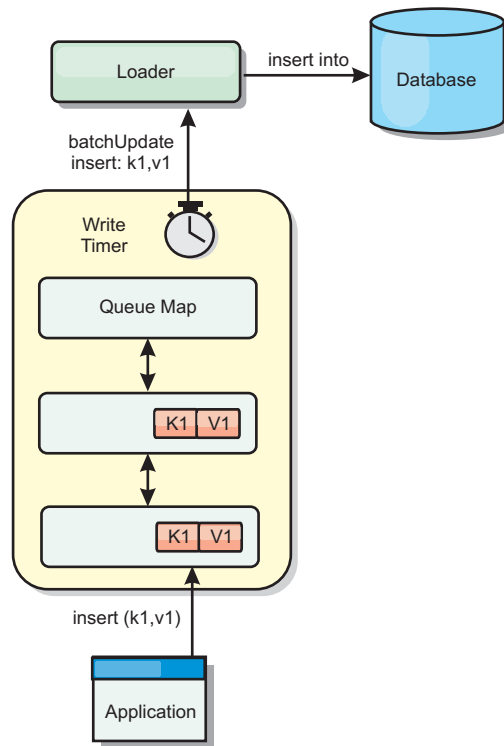


Figure 26. Write-behind caching

The write-behind configuration on a `BackingMap` creates a thread between the loader and the map. The loader then delegates data requests through the thread according to the configuration settings in the `BackingMap.setWriteBehind` method. When an eXtreme Scale transaction inserts, updates, or removes an entry from a map, a `LogElement` object is created for each of these records. These elements are sent to the write-behind loader and queued in a special `ObjectMap` called a queue map. Each backing map with the write-behind setting enabled has its own queue maps. A write-behind thread periodically removes the queued data from the queue maps and pushes them to the real back-end loader.

The write-behind loader only sends insert, update, and delete types of `LogElement` objects to the real loader. All other types of `LogElement` objects, for example, `EVICT` type, are ignored.

Write-behind support is an extension of the Loader plug-in, which you use to integrate eXtreme Scale with the database. For example, consult the [Configuring JPA loaders](#) information about configuring a JPA loader.

Benefits

Enabling write-behind support has the following benefits:

- **Back end failure isolation:** Write-behind caching provides an isolation layer from back end failures. When the back-end database fails, updates are queued in the queue map. The applications can continue driving transactions to eXtreme Scale. When the back end recovers, the data in the queue map is pushed to the back-end.
- **Reduced back end load:** The write-behind loader merges the updates on a key basis so only one merged update per key exists in the queue map. This merge decreases the number of updates to the back-end database.

- **Improved transaction performance:** Individual eXtreme Scale transaction times are reduced because the transaction does not need to wait for the data to be synchronized with the back-end.

Loaders

Java

With a Loader plug-in, a data grid map can behave as a memory cache for data that is typically kept in a persistent store on either the same system or another system. Typically, a database or file system is used as the persistent store. A remote Java virtual machine (JVM) can also be used as the source of data, allowing hub-based caches to be built using eXtreme Scale. A loader has the logic for reading and writing data to and from a persistent store.

Overview

Loaders are backing map plug-ins that are invoked when changes are made to the backing map or when the backing map is unable to satisfy a data request (a cache miss). The Loader is invoked when the cache is unable to satisfy a request for a key, providing read-through capability and lazy-population of the cache. A loader also allows updates to the database when cache values change. All changes within a transaction are grouped together to allow the number of database interactions to be minimized. A TransactionCallback plug-in is used in conjunction with the loader to trigger the demarcation of the backend transaction. Using this plug-in is important when multiple maps are included in a single transaction or when transaction data is flushed to the cache without committing.

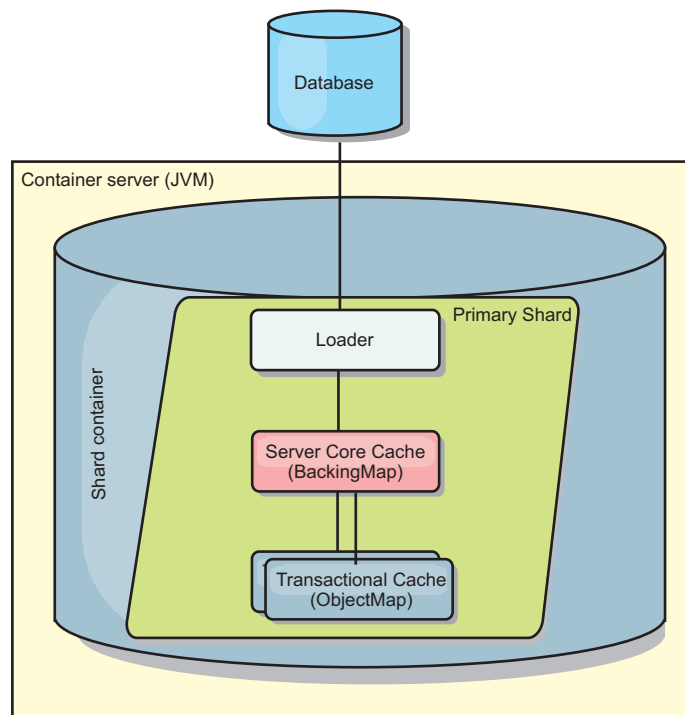


Figure 27. Loader

The loader can also use overqualified updates to avoid keeping database locks. By storing a version attribute in the cache value, the loader can see the before and

after image of the value as it is updated in the cache. This value can then be used when updating the database or back end to verify that the data has not been updated. A Loader can also be configured to preload the data grid when it is started. When partitioned, a Loader instance is associated with each partition. If the "Company" Map has ten partitions, there are ten Loader instances, one per primary partition. When the primary shard for the Map is activated, the preloadMap method for the loader is invoked synchronously or asynchronously which allows loading the map partition with data from the back-end to occur automatically. When invoked synchronously, all client transactions are blocked, preventing inconsistent access to the data grid. Alternatively, a client preloader can be used to load the entire data grid.

Two built-in loaders can greatly simplify integration with relational database back ends. The JPA loaders utilize the Object-Relational Mapping (ORM) capabilities of both the OpenJPA and Hibernate implementations of the Java Persistence API (JPA) specification. See "JPA Loaders" on page 69 for more information.

If you are using loaders in a multiple data center configuration, you must consider how revision data and cache consistency is maintained between the data grids. For more information, see "Loader considerations in a multi-master topology" on page 174.

Loader configuration

To add a Loader into the BackingMap configuration, you can use programmatic configuration or XML configuration. A loader has the following relationship with a backing map.

- A backing map can have only one loader.
- A client backing map (near cache) cannot have a loader.
- A loader definition can be applied to multiple backing maps, but each backing map has its own loader instance.

Data pre-loading and warm-up

In many scenarios that incorporate the use of a loader, you can prepare your data grid by pre-loading it with data.

When used as a complete cache, the data grid must hold all of the data and must be loaded before any clients can connect to it. When you are using a sparse cache, you can warm up the cache with data so that clients can have immediate access to data when they connect.

Two approaches exist for pre-loading data into the data grid: Using a Loader plug-in or using a client loader, as described in the following sections.

Loader plug-in

The loader plug-in is associated with each map and is responsible for synchronizing a single primary partition shard with the database. The preloadMap method of the loader plug-in is invoked automatically when a shard is activated. For example, if you have 100 partitions, 100 loader instances exist, each loading the data for its partition. When run synchronously, all clients are blocked until the preload has completed.

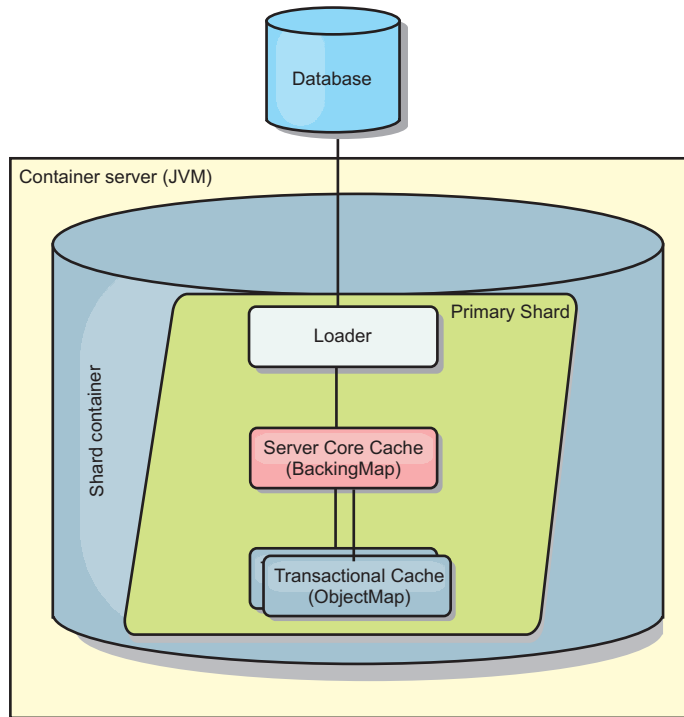


Figure 28. Loader plug-in

Client loader

A client loader is a pattern for using one or more clients to load the grid with data. Using multiple clients to load grid data can be effective when the partition scheme is not stored in the database. You can invoke client loaders manually or automatically when the data grid starts. Client loaders can optionally use the StateManager to set the state of the data grid to pre-load mode, so that clients are not able to access the grid while it is pre-loading the data. WebSphere eXtreme Scale includes a Java Persistence API (JPA)-based loader that you can use to automatically load the data grid with either the OpenJPA or Hibernate JPA providers. For more information about cache providers, see “JPA level 2 (L2) cache plug-in” on page 38.

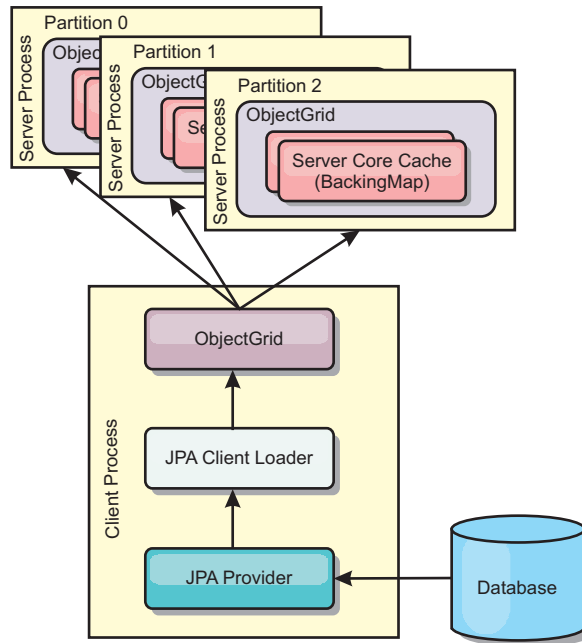


Figure 29. Client loader

Database synchronization techniques

When WebSphere eXtreme Scale is used as a cache, applications must be written to tolerate stale data if the database can be updated independently from an eXtreme Scale transaction. To serve as a synchronized in-memory database processing space, eXtreme Scale provides several ways of keeping the cache updated.

Database synchronization techniques

Periodic refresh

The cache can be automatically invalidated or updated periodically using the Java Persistence API (JPA) time-based database updater. The updater periodically queries the database using a JPA provider for any updates or inserts that have occurred since the previous update. Any changes identified are automatically invalidated or updated when used with a sparse cache. If used with a complete cache, the entries can be discovered and inserted into the cache. Entries are never removed from the cache.

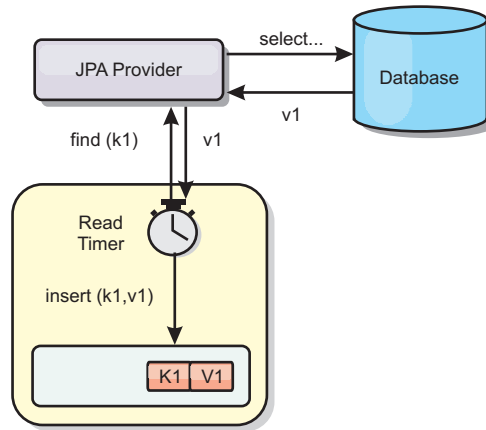


Figure 30. Periodic refresh

Eviction

Sparse caches can utilize eviction policies to automatically remove data from the cache without affecting the database. There are three built-in policies included in eXtreme Scale: time-to-live, least-recently-used, and least-frequently-used. All three policies can optionally evict data more aggressively as memory becomes constrained by enabling the memory-based eviction option.

Event-based invalidation

Sparse and complete caches can be invalidated or updated using an event generator such as Java Message Service (JMS). Invalidation using JMS can be manually tied to any process that updates the back-end using a database trigger. A JMS ObjectGridEventListener plug-in is provided in eXtreme Scale that can notify clients when the server cache has any changes. This can decrease the amount of time the client can see stale data.

Programmatic invalidation

The eXtreme Scale APIs allow manual interaction of the near and server cache using the `Session.beginNoWriteThrough()`, `ObjectMap.invalidate()` and `EntityManager.invalidate()` API methods. If a client or server process no longer needs a portion of the data, the invalidate methods can be used to remove data from the near or server cache. The `beginNoWriteThrough` method applies any `ObjectMap` or `EntityManager` operation to the local cache without calling the loader. If invoked from a client, the operation applies only to the near cache (the remote loader is not invoked). If invoked on the server, the operation applies only to the server core cache without invoking the loader.

Data invalidation

To remove stale cache data, you can use invalidation mechanisms.

Administrative invalidation

You can use the web console or the `xscmd` utility to invalidate data based on the key. You can filter the cache data with a regular expression and then invalidate the data based on the regular expression.

Event-based invalidation

Sparse and complete caches can be invalidated or updated using an event generator such as Java Message Service (JMS). Invalidation using JMS can be manually tied to any process that updates the back-end using a database trigger. A JMS ObjectGridEventListener plug-in is provided in eXtreme Scale that can notify clients when the server cache changes. This type of notification decreases the amount of time the client can see stale data.

Event-based invalidation normally consists of the following three components.

- **Event queue:** An event queue stores the data change events. It could be a JMS queue, a database, an in-memory FIFO queue, or any kind of manifest as long as it can manage the data change events.
- **Event publisher:** An event publisher publishes the data change events to the event queue. An event publisher is usually an application you create or an eXtreme Scale plug-in implementation. The event publisher knows when the data is changed or it changes the data itself. When a transaction commits, events are generated for the changed data and the event publisher publishes these events to the event queue.
- **Event consumer:** An event consumer consumes data change events. The event consumer is usually an application to ensure the target grid data is updated with the latest change from other grids. This event consumer interacts with the event queue to get the latest data change and applies the data changes in the target grid. The event consumers can use eXtreme Scale APIs to invalidate stale data or update the grid with the latest data.

For example, JMSObjectGridEventListener has an option for a client-server model, in which the event queue is a designated JMS destination. All server processes are event publishers. When a transaction commits, the server gets the data changes and publishes them to the designated JMS destination. All the client processes are event consumers. They receive the data changes from the designated JMS destination and apply the changes to the client's near cache.

See *Configuring Java Message Service (JMS)-based client synchronization* for more information.

Programmatic invalidation

The WebSphere eXtreme Scale APIs allow manual interaction of the near and server cache using the `Session.beginNoWriteThrough()`, `ObjectMap.invalidate()` and `EntityManager.invalidate()` API methods. If a client or server process no longer needs a portion of the data, the invalidate methods can be used to remove data from the near or server cache. The `beginNoWriteThrough` method applies any `ObjectMap` or `EntityManager` operation to the local cache without calling the loader. If invoked from a client, the operation applies only to the near cache (the remote loader is not invoked). If invoked on the server, the operation applies only to the server core cache without invoking the loader.

You can use programmatic invalidation with other techniques to determine when to invalidate the data. For example, this invalidation method uses event-based invalidation mechanisms to receive the data change events, and then uses APIs to invalidate the stale data.

8.6+

Near cache invalidation

If you are using a near cache, you can configure an asynchronous invalidation that is triggered each time an update, delete, invalidation operation is run against the data grid. Because the operation is asynchronous, you might still see stale data in the data grid.

To enable near cache invalidation, set the **nearCacheInvalidationEnabled** attribute on the backing map in the ObjectGrid descriptor XML file.

Indexing

Java

Use the `MapIndexPlugin` plug-in to build an index or several indexes on a `BackingMap` to support non-key data access.

Index types and configuration

The indexing feature is represented by the `MapIndexPlugin` plug-in or `Index` for short. The `Index` is a `BackingMap` plug-in. A `BackingMap` can have multiple `Index` plug-ins configured, as long as each one follows the `Index` configuration rules.

You can use the indexing feature to build one or more indexes on a `BackingMap`. An index is built from an attribute or a list of attributes of an object in the `BackingMap`. This feature provides a way for applications to find certain objects more quickly. With the indexing feature, applications can find objects with a specific value or within a range of values of indexed attributes.

Two types of indexing are possible: static and dynamic. With static indexing, you must configure the index plug-in on the `BackingMap` before initializing the `ObjectGrid` instance. You can do this configuration with XML or programmatic configuration of the `BackingMap`. Static indexing starts building an index during `ObjectGrid` initialization. The index is always synchronized with the `BackingMap` and ready for use. After the static indexing process starts, the maintenance of the index is part of the eXtreme Scale transaction management process. When transactions commit changes, these changes also update the static index, and index changes are rolled back if the transaction is rolled back.

With dynamic indexing, you can create an index on a `BackingMap` before or after the initialization of the containing `ObjectGrid` instance. Applications have life cycle control over the dynamic indexing process so that you can remove a dynamic index when it is no longer needed. When an application creates a dynamic index, the index might not be ready for immediate use because of the time it takes to complete the index building process. Because the amount of time depends upon the amount of data indexed, the `DynamicIndexCallback` interface is provided for applications that want to receive notifications when certain indexing events occur. These events include `ready`, `error`, and `destroy`. Applications can implement this callback interface and register with the dynamic indexing process.

8.6+ If a `BackingMap` has an index plug-in configured, you can obtain the application index proxy object from the corresponding `ObjectMap`. Calling the `getIndex` method on the `ObjectMap` and passing in the name of the index plug-in returns the index proxy object. You must cast the index proxy object to an appropriate application index interface, such as `MapIndex`, `MapRangeIndex`,

MapGlobalIndex, or a customized index interface. After obtaining the index proxy object, you can use methods defined in the application index interface to find cached objects.

The steps to use indexing are summarized in the following list:

- Add either static or dynamic index plug-ins into the BackingMap.
- Obtain an application index proxy object by issuing the getIndex method of the ObjectMap.
- Cast the index proxy object to an appropriate application index interface, such as MapIndex, MapRangeIndex, or a customized index interface.
- Use methods that are defined in application index interface to find cached objects.

8.6+ The HashIndex class is the built-in index plug-in implementation that can support the following built-in application index interfaces:

- MapIndex
- MapRangeIndex
- MapGlobalIndex

You also can create your own indexes. You can add HashIndex as either a static or dynamic index into the BackingMap, obtain either the MapIndex, MapRangeIndex, or MapGlobalIndex index proxy object, and use the index proxy object to find cached objects.

8.6+ Global index

Global index is an extension of the built-in HashIndex class that runs on shards in distributed, partitioned data grid environments. It tracks the location of indexed attributes and provides efficient ways to find partitions, keys, values, or entries using attributes in large, partitioned data grid environments.

If global index is enabled in the built-in HashIndex plug-in, then applications can cast an index proxy object to the MapGlobalIndex type, and use it to find data.

Default index

If you want to iterate through the keys in a local map, you can use the default index. This index does not require any configuration, but it must be used against the shard, using an agent or an ObjectGrid instance retrieved from the ShardEvents.shardActivated(ObjectGrid shard) method.

Data quality consideration

The results of index query methods only represent a snapshot of data at a point of time. No locks against data entries are obtained after the results return to the application. Application has to be aware that data updates may occur on a returned data set. For example, the application obtains the key of a cached object by running the findAll method of MapIndex. This returned key object is associated with a data entry in the cache. The application should be able to run the get method on ObjectMap to find an object by providing the key object. If another transaction removes the data object from the cache just before the get method is called, the returned result will be null.

Indexing performance considerations

One of the main objectives of the indexing feature is to improve overall BackingMap performance. If indexing is not used properly, the performance of the application might be compromised. Consider the following factors before using this feature.

- **The number of concurrent write transactions:** Index processing can occur every time a transaction writes data into a BackingMap. Performance degrades if many transactions are writing data into the map concurrently when an application attempts index query operations.
- **The size of the result set that is returned by a query operation:** As the size of the resultset increases, the query performance declines. Performance tends to degrade when the size of the result set is 15% or more of the BackingMap.
- **The number of indexes built over the same BackingMap:** Each index consumes system resources. As the number of the indexes built over the BackingMap increases, performance decreases.

The indexing function can improve BackingMap performance drastically. Ideal cases are when the BackingMap has mostly read operations, the query result set is of a small percentage of the BackingMap entries, and only few indexes are built over the BackingMap.

JPA Loaders

Java

The Java Persistence API (JPA) is a specification that allows mapping Java objects to relational databases. JPA contains a full object-relational mapping (ORM) specification using Java language metadata annotations, XML descriptors, or both to define the mapping between Java objects and a relational database. A number of open-source and commercial implementations are available.

You can use a Java Persistence API (JPA) loader plug-in implementation with eXtreme Scale to interact with any database supported by your chosen loader. To use JPA, you must have a supported JPA provider, such as OpenJPA or Hibernate, JAR files, and a META-INF/persistence.xml file in your class path.

The JPALoader `com.ibm.websphere.objectgrid.jpa.JPALoader` and the JPAEntityLoader `com.ibm.websphere.objectgrid.jpa.JPAEntityLoader` plug-ins are two built-in JPA loader plug-ins that are used to synchronize the ObjectGrid maps with a database. You must have a JPA implementation, such as Hibernate or OpenJPA, to use this feature. The database can be any back end that is supported by the chosen JPA provider.

You can use the JPALoader plug-in when you are storing data using the ObjectMap API. Use the JPAEntityLoader plug-in when you are storing data using the EntityManager API.

JPA loader architecture

The JPA Loader is used for eXtreme Scale maps that store plain old Java objects (POJO).

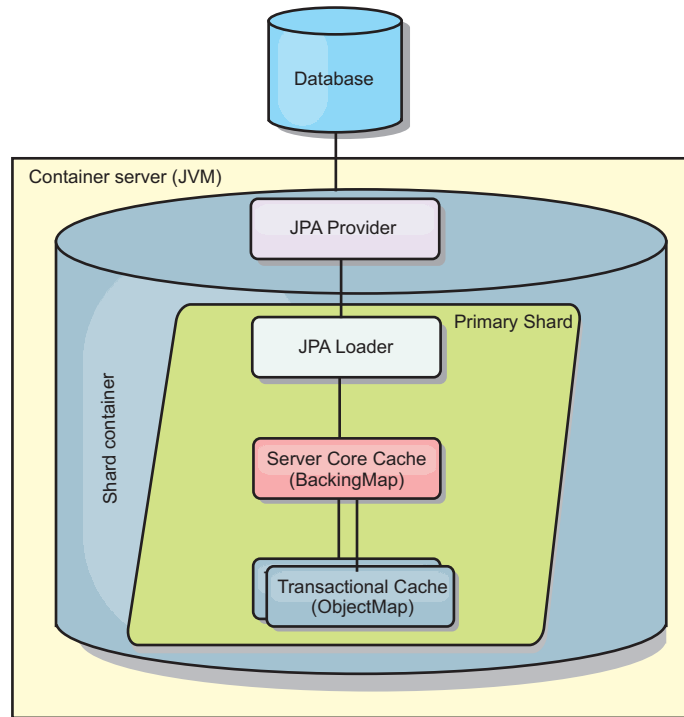


Figure 31. JPA Loader architecture

When an `ObjectMap.get(Object key)` method is called, the eXtreme Scale run time first checks whether the entry is contained in the `ObjectMap` layer. If not, the run time delegates the request to the JPA Loader. Upon request of loading the key, the `JPALoader` calls the `JPA EntityManager.find(Object key)` method to find the data from the JPA layer. If the data is contained in the JPA entity manager, it is returned; otherwise, the JPA provider interacts with the database to get the value.

When an update to `ObjectMap` occurs, for example, using the `ObjectMap.update(Object key, Object value)` method, the eXtreme Scale run time creates a `LogElement` for this update and sends it to the `JPALoader`. The `JPALoader` calls the `JPA EntityManager.merge(Object value)` method to update the value to the database.

For the `JPAEntityLoader`, the same four layers are involved. However, because the `JPAEntityLoader` plug-in is used for maps that store eXtreme Scale entities, relations among entities could complicate the usage scenario. An eXtreme Scale entity is distinguished from JPA entity. For more information, see the information about the `JPAEntityLoader` plug-in in the *Programming Guide*.

Methods

Loaders provide three main methods:

1. `get`: Returns a list of values that correspond to the list of keys that are passed in by retrieving the data using JPA. The method uses JPA to find the entities in the database. For the `JPALoader` plug-in, the returned list contains a list of JPA entities directly from the find operation. For the `JPAEntityLoader` plug-in, the returned list contains eXtreme Scale entity value tuples that are converted from the JPA entities.
2. `batchUpdate`: Writes the data from `ObjectGrid` maps to the database. Depending on different operation types (insert, update, or delete), the loader

uses the JPA persist, merge, and remove operations to update the data to the database. For the JPALoader, the objects in the map are directly used as JPA entities. For the JPAEntityLoader, the entity tuples in the map are converted into objects which are used as JPA entities.

3. preloadMap: Preloads the map using the ClientLoader.load client loader method. For partitioned maps, the preloadMap method is only called in one partition. The partition is specified the preloadPartition property of the JPALoader or JPAEntityLoader class. If the preloadPartition value is set to less than zero, or greater than (*total_number_of_partitions* - 1), preload is disabled.

Both JPALoader and JPAEntityLoader plug-ins work with the JPATxCallback class to coordinate the eXtreme Scale transactions and JPA transactions. JPATxCallback must be configured in the ObjectGrid instance to use these two loaders.

Configuration and programming

If you are using JPA loaders in a multi-master environment, see “Loader considerations in a multi-master topology” on page 174. For more information about configuring JPA loaders, see the information about JPA loaders in the *Administration Guide*. For more information about programming JPA loaders, see the *Programming Guide*.

Serialization overview

Java

.NET

Data is serialized to improve performance and reduce the memory footprint of the data grid. The serialization options that you choose depend on many factors, such as how your application interacts with the data grid, the programming language of your data grid applications, or how much data you are storing in the data grid.

When data is serialized, it is converted into a data stream for transmission over a network in the following situations:

- When clients communicate with servers, and those servers send information back to the client
- When servers replicate data from one server to another

Alternatively, you might decide to forgo the serialization process through WebSphere eXtreme Scale and store raw data as byte arrays. Byte arrays are much cheaper to store in memory. The Java virtual machine (JVM) has fewer objects to search for during garbage collection. The objects can be deserialized only when they are needed. Use byte arrays only if access to the objects with queries or indexes is not required. Because the data is stored as bytes, eXtreme Scale has no metadata for describing attributes to query.

Java

Serialization for Java applications

To serialize data, you can use eXtreme Data format (XDF), Java serialization, the ObjectTransformer plug-in, or the DataSerializer plug-ins. To optimize serialization with any of these options, you can use the COPY_TO_BYTES mode to improve performance up to 70 percent. With COPY_TO_BYTES mode, the data is serialized when transactions commit, which means that serialization happens only one time. The serialized data is sent unchanged from the client to the server or from the

server to replicated server. By using the COPY_TO_BYTES mode, you can reduce the memory footprint that a large graph of objects can use.

Use the following figures to help you determine which type of serialization method is most appropriate for your development needs.

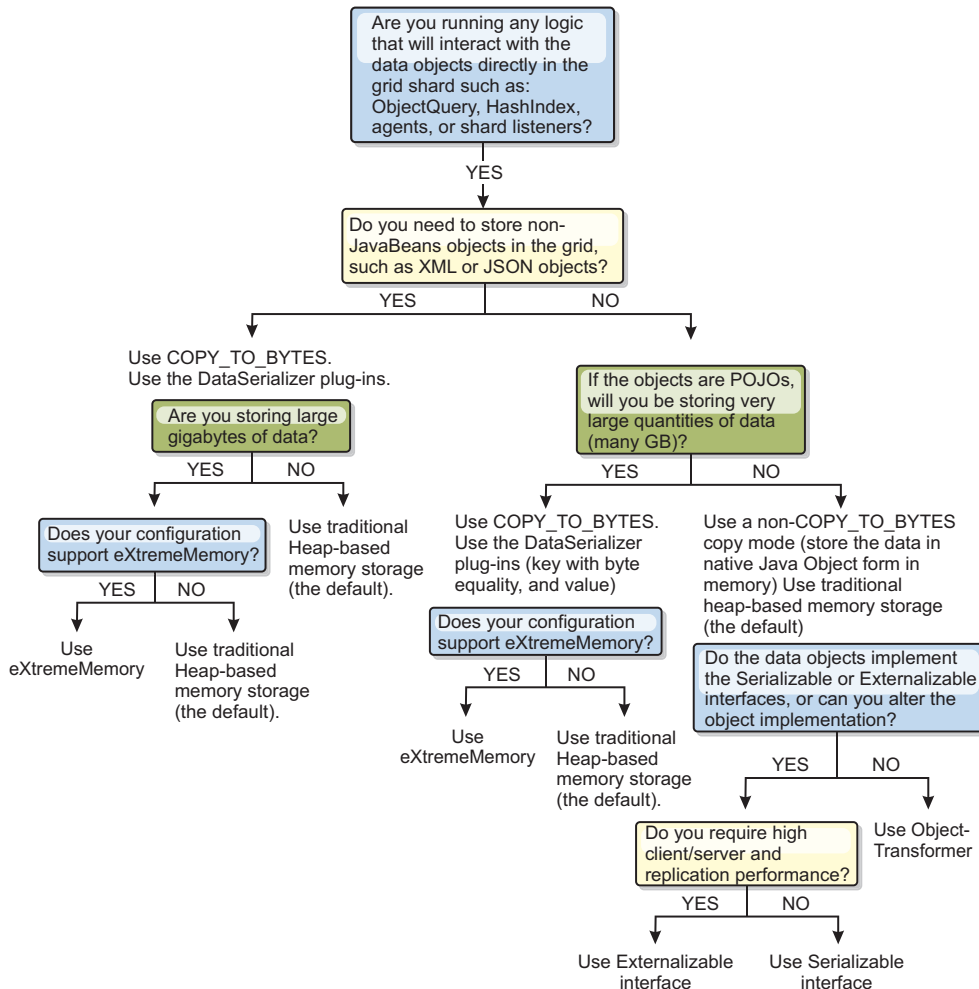


Figure 32. Serialization methods that are available when you are running logic that interacts with data objects directly in the data grid shard

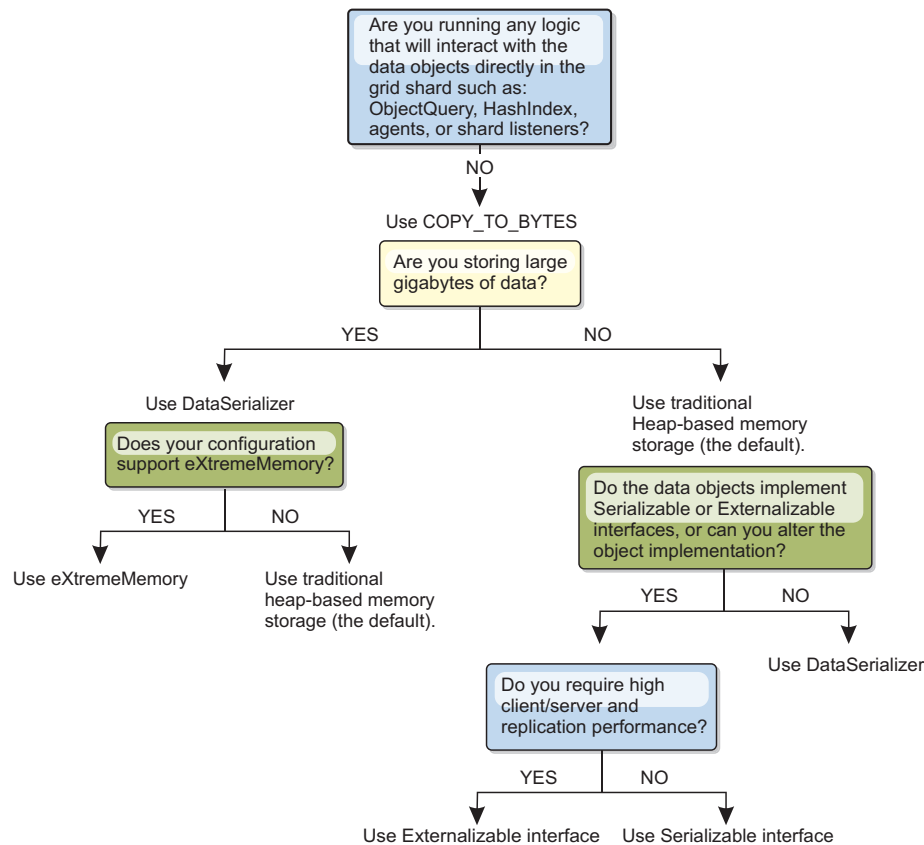


Figure 33. Serialization methods when you are not directly interacting with the data grid shard.

.NET 8.6+ Serialization for .NET applications

When your environment includes .NET applications, you must use XDF for your data serialization. With XDF, you can serialize and store keys and values in the data grid in a language-independent format. XDF is enabled when your environment has IBM eXtremeIO (XIO) enabled and COPY_TO_BYTES mode on the map.

Because eXtremeMemory is configured on container servers only, you can use it in an environment that has .NET applications. eXtremeMemory leads to more consistent relative response times in the environment.

To learn more about the supported forms of serialization in the eXtreme Scale product, see the following topics:

Serialization using Java

Java serialization refers to either default serialization, which uses the Serializable interface, or custom serialization, which uses both the Serializable and Externalizable interfaces.

Default serialization

To use default serialization, implement the `java.io.Serializable` interface, which includes the API that converts objects into bytes, which are later deserialized. Use the `java.io.ObjectOutputStream` class to persist the object. Then, call the

`ObjectOutputStream.writeObject()` method to initiate serialization and flatten the Java object.

Custom serialization

Some cases exist where objects must be modified to use custom serialization, such as implementing the `java.io.Externalizable` interface or by implementing the `writeObject` and `readObject` methods for classes implementing the `java.io.Serializable` interface. Custom serialization techniques should be employed when the objects are serialized using mechanisms other than the `ObjectGrid` API or `EntityManager` API methods.

For example, when objects or entities are stored as instance data in a `DataGrid` API agent or the agent returns objects or entities, those objects are not transformed using an `ObjectTransformer`. The agent, will however, automatically use the `ObjectTransformer` when using `EntityMixin` interface. See `DataGrid` agents and entity based `Maps` for further details.

ObjectTransformer plug-in

Java

With the `ObjectTransformer` plug-in, you can serialize, deserialize, and copy objects in the cache for increased performance.



The `ObjectTransformer` interface has been replaced by the `DataSerializer` plug-ins, which you can use to efficiently store arbitrary data in `WebSphere eXtreme Scale` so that existing product APIs can efficiently interact with your data.

If you see performance issues with processor usage, add an `ObjectTransformer` plug-in to each map. If you do not provide an `ObjectTransformer` plug-in, up to 60-70 percent of the total processor time is spent serializing and copying entries.

Purpose

With the `ObjectTransformer` plug-in, your applications can provide custom methods for the following operations:

- Serialize or deserialize the key for an entry
- Serialize or deserialize the value for an entry
- Copy a key or value for an entry

If no `ObjectTransformer` plug-in is provided, you must be able to serialize the keys and values because the `ObjectGrid` uses a serialize and deserialize sequence to copy the objects. This method is expensive, so use an `ObjectTransformer` plug-in when performance is critical. The copying occurs when an application looks up an object in a transaction for the first time. You can avoid this copying by setting the copy mode of the `Map` to `NO_COPY` or reduce the copying by setting the copy mode to `COPY_ON_READ`. Optimize the copy operation when needed by the application by providing a custom copy method on this plug-in. Such a plug-in can reduce the copy overhead from 65–70 percent to 2/3 percent of total processor time.

The default `copyKey` and `copyValue` method implementations first attempt to use the `clone` method, if the method is provided. If no `clone` method implementation is provided, the implementation defaults to serialization.

Object serialization is also used directly when the eXtreme Scale is running in distributed mode. The LogSequence uses the ObjectTransformer plug-in to help serialize keys and values before transmitting the changes to peers in the ObjectGrid. You must take care when providing a custom serialization method instead of using the built-in Java developer kit serialization. Object versioning is a complex issue and you might encounter problems with version compatibility if you do not ensure that your custom methods are designed for versioning.

The following list describes how the eXtreme Scale tries to serialize both keys and values:

- If a custom ObjectTransformer plug-in is written and plugged in, eXtreme Scale calls methods in the ObjectTransformer interface to serialize keys and values and get copies of object keys and values.
- If a custom ObjectTransformer plug-in is not used, eXtreme Scale serializes and deserializes values according to the default. If the default plug-in is used, each object is implemented as externalizable or is implemented as serializable.
 - If the object supports the Externalizable interface, the writeExternal method is called. Objects that are implemented as externalizable lead to better performance.
 - If the object does not support the Externalizable interface and does implement the Serializable interface, the object is saved using the ObjectOutputStream method.

Using the ObjectTransformer interface

An ObjectTransformer object must implement the ObjectTransformer interface and follow the common ObjectGrid plug-in conventions.

Two approaches, programmatic configuration and XML configuration, are used to add an ObjectTransformer object into the BackingMap configuration as follows.

Programmatically plug in an ObjectTransformer object

The following code snippet creates the custom ObjectTransformer object and adds it to a BackingMap:

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid", false);
BackingMap backingMap = myGrid.getMap("myMap");
MyObjectTransformer myObjectTransformer = new MyObjectTransformer();
backingMap.setObjectTransformer(myObjectTransformer);
```

XML configuration approach to plug in an ObjectTransformer

Assume that the class name of the ObjectTransformer implementation is the com.company.org.MyObjectTransformer class. This class implements the ObjectTransformer interface. An ObjectTransformer implementation can be configured using the following XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
    <objectGrid name="myGrid">
      <backingMap name="myMap" pluginCollectionRef="myMap" />
    </objectGrid>
  </objectGrids>

  <backingMapPluginCollections>
    <backingMapPluginCollection id="myMap">
```

```

        <bean id="ObjectTransformer" className="com.company.org.MyObjectTransformer" />
    </backingMapPluginCollection>
    </backingMapPluginCollections>
</objectGridConfig>

```

ObjectTransformer usage scenarios

You can use the ObjectTransformer plug-in in the following situations:

- Non-serializable object
- Serializable object but improve serialization performance
- Key or value copy

In the following example, ObjectGrid is used to store the Stock class:

```

/**
 * Stock object for ObjectGrid demo
 *
 *
 */
public class Stock implements Cloneable {
    String ticket;
    double price;
    String company;
    String description;
    int serialNumber;
    long lastTransactionTime;
    /**
     * @return Returns the description.
     */
    public String getDescription() {
        return description;
    }
    /**
     * @param description The description to set.
     */
    public void setDescription(String description) {
        this.description = description;
    }
    /**
     * @return Returns the lastTransactionTime.
     */
    public long getLastTransactionTime() {
        return lastTransactionTime;
    }
    /**
     * @param lastTransactionTime The lastTransactionTime to set.
     */
    public void setLastTransactionTime(long lastTransactionTime) {
        this.lastTransactionTime = lastTransactionTime;
    }
    /**
     * @return Returns the price.
     */
    public double getPrice() {
        return price;
    }
    /**
     * @param price The price to set.
     */
    public void setPrice(double price) {
        this.price = price;
    }
    /**
     * @return Returns the serialNumber.
     */
    public int getSerialNumber() {
        return serialNumber;
    }
    /**
     * @param serialNumber The serialNumber to set.
     */
    public void setSerialNumber(int serialNumber) {
        this.serialNumber = serialNumber;
    }
    /**
     * @return Returns the ticket.

```

```

    */
    public String getTicket() {
        return ticket;
    }
    /**
     * @param ticket The ticket to set.
     */
    public void setTicket(String ticket) {
        this.ticket = ticket;
    }
    /**
     * @return Returns the company.
     */
    public String getCompany() {
        return company;
    }
    /**
     * @param company The company to set.
     */
    public void setCompany(String company) {
        this.company = company;
    }
    //clone
    public Object clone() throws CloneNotSupportedException
    {
        return super.clone();
    }
}

```

You can write a custom object transformer class for the Stock class:

```

/**
 * Custom implementation of ObjectGrid ObjectTransformer for stock object
 *
 */
public class MyStockObjectTransformer implements ObjectTransformer {
    /* (non-Javadoc)
     * @see com.ibm.websphere.objectgrid.plugins.ObjectTransformer#serializeKey
     * (java.lang.Object,
     * java.io.ObjectOutputStream)
     */
    public void serializeKey(Object key, ObjectOutputStream stream) throws IOException {
        String ticket= (String) key;
        stream.writeUTF(ticket);
    }

    /* (non-Javadoc)
     * @see com.ibm.websphere.objectgrid.plugins.
    ObjectTransformer#serializeValue(java.lang.Object,
    java.io.ObjectOutputStream)
     */
    public void serializeValue(Object value, ObjectOutputStream stream) throws IOException {
        Stock stock= (Stock) value;
        stream.writeUTF(stock.getTicket());
        stream.writeUTF(stock.getCompany());
        stream.writeUTF(stock.getDescription());
        stream.writeDouble(stock.getPrice());
        stream.writeLong(stock.getLastTransactionTime());
        stream.writeInt(stock.getSerialNumber());
    }

    /* (non-Javadoc)
     * @see com.ibm.websphere.objectgrid.plugins.
    ObjectTransformer#inflateKey(java.io.ObjectInputStream)
     */
    public Object inflateKey(ObjectInputStream stream) throws IOException, ClassNotFoundException {
        String ticket=stream.readUTF();
        return ticket;
    }

    /* (non-Javadoc)
     * @see com.ibm.websphere.objectgrid.plugins.
    ObjectTransformer#inflateValue(java.io.ObjectInputStream)
     */
    public Object inflateValue(ObjectInputStream stream) throws IOException, ClassNotFoundException {
        Stock stock=new Stock();
        stock.setTicket(stream.readUTF());
        stock.setCompany(stream.readUTF());
        stock.setDescription(stream.readUTF());
        stock.setPrice(stream.readDouble());
        stock.setLastTransactionTime(stream.readLong());
        stock.setSerialNumber(stream.readInt());
        return stock;
    }
}

```

```

/* (non-Javadoc)
 * @see com.ibm.websphere.objectgrid.plugins.
ObjectTransformer#copyValue(java.lang.Object)
 */
public Object copyValue(Object value) {
    Stock stock = (Stock) value;
    try {
        return stock.clone();
    }
    catch (CloneNotSupportedException e)
    {
        // display exception message
    }
}

/* (non-Javadoc)
 * @see com.ibm.websphere.objectgrid.plugins.
ObjectTransformer#copyKey(java.lang.Object)
 */
public Object copyKey(Object key) {
    String ticket=(String) key;
    String ticketCopy= new String (ticket);
    return ticketCopy;
}
}

```

Then, plug in this custom MyStockObjectTransformer class into the BackingMap:

```

ObjectGridManager ogf=ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogf.getObjectGrid("NYSE");
BackingMap bm = og.defineMap("NYSEStocks");
MyStockObjectTransformer ot = new MyStockObjectTransformer();
bm.setObjectTransformer(ot);

```

Serialization using the DataSerializer plug-ins

Use the DataSerializer plug-ins to efficiently store arbitrary data in WebSphere eXtreme Scale so that existing product APIs can efficiently interact with your data.

Serialization methods such as Java serialization and the ObjectTransformer plug-in allow data to be marshalled over the network. In addition, when you use these serialization options with the COPY_TO_BYTES copy mode, moving data between clients and servers becomes less expensive and performance is improved. However, these options do not solve the following issues that can exist:

- Keys are not stored in bytes; they are still Java objects.
- Server-side code must still inflate the object; for example, query and index still use reflection and must inflate the object. Additionally, agents, listeners, and plug-ins still need the object form.
- Classes still need to be in the server classpath.
- Data is still in Java serialization form (ObjectOutputStream).

The DataSerializer plug-ins introduce an efficient way of solving these problems. Specifically, the DataSerializer plug-in gives you a way to describe your serialization format, or byte array, to WebSphere eXtreme Scale so that the product can interrogate the byte array without requiring a specific object format. The public DataSerializer plug-in classes and interfaces are in the package, com.ibm.websphere.objectgrid.plugins.io. For more information, refer to the .

Important: Entity Java objects are not stored directly into the BackingMaps when you use the EntityManager API. The EntityManager API converts the entity object to Tuple objects. Entity maps are automatically associated with a highly optimized ObjectTransformer. Whenever the ObjectMap API or EntityManager API is used to interact with entity maps, the ObjectTransformer entity is invoked. Therefore, when you use entities, no work is required for serialization because the product automatically completes this process for you.

Scalability overview

WebSphere eXtreme Scale is scalable through the use of partitioned data, and can scale to thousands of containers if required because each container is independent from other containers.

WebSphere eXtreme Scale divides data sets into distinct partitions that can be moved between processes or even between physical servers at run time. You can, for example, start with a deployment of four servers and then expand to a deployment with 10 servers as the demands on the cache grow. Just as you can add more physical servers and processing units for vertical scalability, you can extend the elastic scaling capability horizontally with partitioning. Horizontal scaling is a major advantage to using WebSphere eXtreme Scale over an in-memory database. In-memory databases can only scale vertically.

With WebSphere eXtreme Scale, you can also use a set of APIs to gain transactional access this partitioned and distributed data. The choices you make for interacting with the cache are as significant as the functions to manage the cache for availability from a performance perspective.

Note: Scalability is not available when containers communicate with one another. The availability management, or core grouping, protocol is an $O(N^2)$ heartbeat and view maintenance algorithm, but is mitigated by keeping the number of core group members under 20. Only peer to peer replication between shards exists.

Distributed clients

The WebSphere eXtreme Scale client protocol supports very large numbers of clients. The partitioning strategy offers assistance by assuming that all clients are not always interested in all partitions because connections can be spread across multiple containers. Clients are connected directly to the partitions so latency is limited to one transferred connection.

Data grids, partitions, and shards

A data grid is divided into partitions. A partition holds an exclusive subset of the data. A partition contains one or more shards: a primary shard and replica shards. Replica shards are not necessary in a partition, but you can use replica shards to provide high availability. Whether your deployment is an independent in-memory data grid or an in-memory database processing space, data access in eXtreme Scale relies heavily on shards.

The data for a partition is stored in a set of shards at run time. This set of shards includes a primary shared and possibly one or more replica shards. A shard is the smallest unit that eXtreme Scale can add or remove from a Java virtual machine.

Two placement strategies exist: fixed partition placement (default) and per container placement. The following discussion focuses on the usage of the fixed partition placement strategy.

Total number of shards

If your environment includes 10 partitions that hold 1 million objects with no replicas, then 10 shards would exist that each store 100,000 objects. If you add a replica to this scenario, then an extra shard exists in each partition. In this case, 20 shards exist: 10 primary shards and 10 replica shards. Each one of these shards

store 100,000 objects. Each partition consists of a primary shard and one or more (N) replica shards. Determining the optimal shard count is critical. If you configure few shards, data is not distributed evenly among the shards, resulting in out of memory errors and processor overloading issues. You must have at least 10 shards for each JVM as you scale. When you are initially deploying the data grid, you would potentially use many partitions.

Number of shards per JVM scenarios

Scenario: small number of shards for each JVM

Data is added and removed from a JVM using shard units. Shards are never split into pieces. If 10 GB of data existed, and 20 shards exist to hold this data, then each shard holds 500 MB of data on average. If nine Java virtual machines host the data grid, then on average each JVM has two shards. Because 20 is not evenly divisible by 9, a few Java virtual machines have three shards, in the following distribution:

- Seven Java virtual machines with two shards
- Two Java virtual machines with three shards

Because each shard holds 500 MB of data, the distribution of data is unequal. The seven Java virtual machines with two shards each host 1 GB of data. The two Java virtual machines with three shards have 50% more data, or 1.5 GB, which is a much larger memory burden. Because the two Java virtual machines are hosting three shards, they also receive 50% more requests for their data. As a result, having few shards for each JVM causes imbalance. To increase the performance, you increase the number of shards for each JVM.

Scenario: increased number of shards per JVM

In this scenario, consider a much larger number of shards. In this scenario, there are 101 shards with nine Java virtual machines hosting 10 GB of data. In this case, each shard holds 99 MB of data. The Java virtual machines have the following distribution of shards:

- Seven Java virtual machines with 11 shards
- Two Java virtual machines with 12 shards

The two Java virtual machines with 12 shards now have just 99 MB more data than the other shards, which is a 9% difference. This scenario is much more evenly distributed than the 50% difference in the scenario with few shards. From a processor use perspective, only 9% more work exists for the two Java virtual machines with the 12 shards compared to the seven Java virtual machines that have 11 shards. By increasing the number of shards in each JVM, the data and processor use is distributed in a fair and even way.

When you are creating your system, use 10 shards for each JVM in its maximally sized scenario, or when the system is running its maximum number of Java virtual machines in your planning horizon.

Additional placement factors

The number of partitions, the placement strategy, and number and type of replicas are set in the deployment policy. The number of shards that are placed depend on

the deployment policy that you define. The `minSyncReplicas`, `developmentMode`, `maxSyncReplicas`, and `maxAsyncReplicas` attributes affect where partitions and replicas are placed.

The following factors affect when shards can be placed:

- The `xscmd -c suspendBalancing` and `xscmd -c resumeBalancing` commands.
- The server properties file, which has the `placementDeferralInterval` property that defines the number of milliseconds before shards are placed on the container servers.
- The `numInitialContainers` attribute in the deployment policy.

If the maximum number of replicas are not placed during the initial startup, additional replicas might be placed if you start additional servers later. When you are planning the number of shards per JVM, the maximum number of primary and replica shards is dependent on having enough JVMs started to support the configured maximum number of replicas. A replica is never placed in the same process as its primary. If a process is lost, both the primary and the replica are lost. When the `developmentMode` attribute is set to `false`, the primary and replicas are not placed on the same physical server.

Partitioning

Use partitioning to scale out an application. You can define the number of partitions in your deployment policy.

About partitioning

Partitioning is not like Redundant Array of Independent Disks (RAID) striping, which slices each instance across all stripes. Each partition hosts the complete data for individual entries. Partitioning is a very effective means for scaling, but is not applicable to all applications. Applications that require transactional guarantees across large sets of data do not scale and cannot be partitioned effectively. WebSphere eXtreme Scale does not currently support two-phase commit across partitions.

Important: Select the number of partitions carefully. The number of partitions that are defined in the deployment policy directly affects the number of container servers to which an application can scale. Each partition is made up of a primary shard and the configured number of replica shards. The $(\text{Number_Partitions} * (1 + \text{Number_Replicas}))$ formula is the number of containers that can be used to scale out a single application.

Using partitions

A data grid can have up to thousands of partitions. A data grid can scale up to the product of the number of partitions times the number of shards per partition. For example, if you have 16 partitions and each partition has one primary and one replica, or two shards, then you can potentially scale to 32 Java virtual machines. In this case, one shard is defined for each JVM. You must choose a reasonable number of partitions based on the expected number of Java virtual machines that you are likely to use. Each shard increases processor and memory usage for the system. The system is designed to scale out to handle this overhead in line with how many server Java virtual machines are available.

Applications should not use thousands of partitions if the application runs on a data grid of four container server Java virtual machines. The application should be

configured to have a reasonable number of shards for each container server JVM. For example, an unreasonable configuration is 2000 partitions with two shards that are running on four container Java virtual machines. This configuration results in 4000 shards that are placed on four container Java virtual machines or 1000 shards per container JVM.

A better configuration would be under 10 shards for each expected container JVM. This configuration still gives the possibility of allowing for elastic scaling that is ten times the initial configuration while keeping a reasonable number of shards per container JVM.

Consider this scaling example: you currently have six physical servers with two container Java virtual machines per physical server. You expect to grow to 20 physical servers over the next three years. With 20 physical servers, you have 40 container server Java virtual machines, and choose 60 to be pessimistic. You want four shards per container JVM. You have 60 potential containers, or a total of 240 shards. If you have a primary and replica per partition, then you want 120 partitions. This example gives you 240 divided by 12 container Java virtual machines, or 20 shards per container JVM for the initial deployment with the potential to scale out to 20 computers later.

ObjectMap and partitioning

With the default FIXED_PARTITION placement strategy, maps are split across partitions and keys hash to different partitions. The client does not need to know to which partition the keys belong. If a mapSet has multiple maps, the maps should be committed in separate transactions.

Entities and partitioning

Entity manager entities have an optimization that helps clients that are working with entities on a server. The entity schema on the server for the map set can specify a single root entity. The client must access all entities through the root entity. The entity manager can then find related entities from that root in the same partition without requiring the related maps to have a common key. The root entity establishes affinity with a single partition. This partition is used for all entity fetches within the transaction after affinity is established. This affinity can save memory because the related maps do not require a common key. The root entity must be specified with a modified entity annotation as shown in the following example:

```
@Entity(schemaRoot=true)
```

Use the entity to find the root of the object graph. The object graph defines the relationships between one or more entities. Each linked entity must resolve to the same partition. All child entities are assumed to be in the same partition as the root. The child entities in the object graph are only accessible from a client from the root entity. Root entities are always required in partitioned environments when using an eXtreme Scale client to communicate to the server. Only one root entity type can be defined per client. Root entities are not required when using Extreme Transaction Processing (XTP) style ObjectGrids, because all communication to the partition is accomplished through direct, local access and not through the client and server mechanism.

Placement and partitions

You have two placement strategies available for WebSphere eXtreme Scale: fixed partition and per-container. The choice of placement strategy affects how your deployment configuration places partitions over the remote data grid.

Fixed partition placement

You can set the placement strategy in the deployment policy XML file. The default placement strategy is fixed-partition placement, enabled with the `FIXED_PARTITION` setting. The number of primary shards that are placed across the available containers is equal to the number of partitions that you have configured with the `numberOfPartitions` attribute. If you have configured replicas, the minimum total number of shards placed is defined by the following formula: $((1 \text{ primary shard} + \text{minimum synchronous shards}) * \text{partitions defined})$. The maximum total number of shards placed is defined by the following formula: $((1 \text{ primary shard} + \text{maximum synchronous shards} + \text{maximum asynchronous shards}) * \text{partitions})$. Your WebSphere eXtreme Scale deployment spreads these shards over the available containers. The keys of each map are hashed into assigned partitions based on the total partitions you have defined. They keys hash to the same partition even if the partition moves because of failover or server changes.

For example, if the `numberPartitions` value is 6 and the `minSync` value is 1 for `MapSet1`, the total shards for that map set is 12 because each of the 6 partitions requires a synchronous replica. If three containers are started, WebSphere eXtreme Scale places four shards per container for `MapSet1`.

Per-container placement

The alternate placement strategy is per-container placement, which is enabled with the `PER_CONTAINER` setting for the `placementStrategy` attribute in the map set element in the deployment XML file. With this strategy, the number of primary shards placed on each new container is equal to the number of partitions, P , that you have configured. The WebSphere eXtreme Scale deployment environment places P replicas of each partition for each remaining container. The `numInitialContainers` setting is ignored when you are using per-container placement. The partitions get larger as the containers grow. The keys for maps are not fixed to a certain partition in this strategy. The client routes to a partition and uses a random primary. If a client wants to reconnect to the same session that it used to find a key again, it must use a session handle.

For more information, see the topic on using a `SessionHandle` for routing in the *Programming Guide*.

For failover or stopped servers, the WebSphere eXtreme Scale environment moves the primary shards in the per-container placement strategy if they still contain data. If the shards are empty, they are discarded. In the per-container strategy, old primary shards are not kept because new primary shards are placed for every container.

WebSphere eXtreme Scale allows per-container placement as an alternative to what could be termed the "typical" placement strategy, a fixed-partition approach with the key of a Map hashed to one of those partitions. In a per-container case (which you set with `PER_CONTAINER`), your deployment places the partitions on the set of online container servers and automatically scales them out or in as containers are added or removed from the server data grid. A data grid with the

fixed-partition approach works well for key-based grids, where the application uses a key object to locate data in the grid. The following discusses the alternative.

Example of a per-container data grid

PER_CONTAINER data grids are different. You specify that the data grid uses the PER_CONTAINER placement strategy with the placementStrategy attribute in your deployment XML file. Instead of configuring how many partitions total you want in the data grid, you specify how many partitions you want per container that you start.

For example, if you set five partitions per container, five new anonymous partition primaries are created when you start that container server, and the necessary replicas are created on the other deployed container servers.

The following is a potential sequence in a per-container environment as the data grid grows.

1. Start container C0 hosting 5 primaries (P0 - P4).
 - C0 hosts: P0, P1, P2, P3, P4.
2. Start container C1 hosting 5 more primaries (P5 - P9). Replicas are balanced on the containers.
 - C0 hosts: P0, P1, P2, P3, P4, R5, R6, R7, R8, R9.
 - C1 hosts: P5, P6, P7, P8, P9, R0, R1, R2, R3, R4.
3. Start container C2 hosting 5 more primaries (P10 - P14). Replicas are balanced further.
 - C0 hosts: P0, P1, P2, P3, P4, R7, R8, R9, R10, R11, R12.
 - C1 hosts: P5, P6, P7, P8, P9, R2, R3, R4, R13, R14.
 - C2 hosts: P10, P11, P12, P13, P14, R5, R6, R0, R1.

The pattern continues as more containers are started, creating five new primary partitions each time and rebalancing replicas on the available containers in the data grid.

Note: WebSphere eXtreme Scale does not move primary shards when using the PER_CONTAINER strategy, only replicas.

Remember that the partition numbers are arbitrary and have nothing to do with keys, so you cannot use key-based routing. If a container stops then the partition IDs created for that container are no longer used, so there is a gap in the partition IDs. In the example, there would no longer be partitions P5 - P9 if the container C1 failed, leaving only P0 - P4 and P10 - P14, so key-based hashing is impossible.

Using numbers like five or even more likely 10 for how many partitions per container works best if you consider the consequences of a container failure. To spread the load of hosting shards evenly across the data grid, you need more than just one partition for each container. If you had a single partition per container, then when a container fails, only one container (the one hosting the corresponding replica shard) must bear the full load of the lost primary. In this case, the load is immediately doubled for the container. However, if you have five partitions per container, then five containers pick up the load of the lost container, lowering impact on each by 80 percent. Using multiple partitions per container generally lowers the potential impact on each container substantially. More directly, consider a case in which a container spikes unexpectedly—the replication load of that container is spread over 5 containers rather than only one.

Using the per-container policy

Several scenarios make the per-container strategy an ideal configuration, such as with HTTP session replication or application session state. In such a case, an HTTP router assigns a session to a servlet container. The servlet container needs to create an HTTP session and chooses one of the 5 local partition primaries for the session. The "ID" of the partition chosen is then stored in a cookie. The servlet container now has local access to the session state which means zero latency access to the data for this request as long as you maintain session affinity. And eXtreme Scale replicates any changes to the partition.

In practice, remember the repercussions of a case in which you have multiple partitions per container (say 5 again). Of course, with each new container started, you have 5 more partition primaries and 5 more replicas. Over time, more partitions should be created and they should not move or be destroyed. But this is not how the containers would actually behave. When a container starts, it hosts 5 primary shards, which can be called "home" primaries, existing on the respective containers that created them. If the container fails, the replicas become primaries and eXtreme Scale creates 5 more replicas to maintain high availability (unless you disabled auto repair). The new primaries are in a different container than the one that created them, which can be called "foreign" primaries. The application should never place new state or sessions in a foreign primary. Eventually, the foreign primary has no entries and eXtreme Scale automatically deletes it and its associated replicas. The foreign primaries' purpose is to allow existing sessions to still be available (but not new sessions).

A client can still interact with a data grid that does not rely on keys. The client just begins a transaction and stores data in the data grid independent of any keys. It asks the Session for a SessionHandle object, a serializable handle allowing the client to interact with the same partition when necessary. For more information see the topic on using a SessionHandle for routing in the *Programming Guide*. WebSphere eXtreme Scale chooses a partition for the client from the list of home partition primaries. It does not return a foreign primary partition. The SessionHandle can be serialized in an HTTP cookie, for example, and later convert the cookie back into a SessionHandle. Then the WebSphere eXtreme Scale APIs can obtain a Session bound to the same partition again, using the SessionHandle.

Note: You cannot use agents to interact with a PER_CONTAINER data grid.

Advantages

The previous description is different from a normal FIXED_PARTITION or hash data grid because the per-container client stores data in a place in the grid, gets a handle to it and uses the handle to access it again. There is no application-supplied key as there is in the fixed-partition case.

Your deployment does not make a new partition for each Session. So in a per-container deployment, the keys used to store data in the partition must be unique within that partition. For example, you may have your client generate a unique SessionID and then use it as the key to find information in Maps in that partition. Multiple client sessions then interact with the same partition so the application needs to use unique keys to store session data in each given partition.

The previous examples used 5 partitions, but the numberOfPartitions parameter in the objectgrid XML file can be used to specify the partitions as required. Instead of

per data grid, the setting is per container. (The number of replicas is specified in the same way as with the fixed-partition policy.)

The per-container policy can also be used with multiple zones. If possible, eXtreme Scale returns a SessionHandle to a partition whose primary is located in the same zone as that client. The client can specify the zone as a parameter to the container or by using an API. The client zone ID can be set using `serverproperties` or `clientproperties`.

The PER_CONTAINER strategy for a data grid suits applications storing conversational type state rather than database-oriented data. The key to access the data would be a conversation ID and is not related to a specific database record. It provides higher performance (because the partition primaries can be collocated with the servlets for example) and easier configuration (without having to calculate partitions and containers).

Single-partition and cross-data-grid transactions

The major distinction between WebSphere eXtreme Scale and traditional data storage solutions like relational databases or in-memory databases is the use of partitioning, which allows the cache to scale linearly. The important types of transactions to consider are single-partition and every-partition (cross-data-grid) transactions.

In general, interactions with the cache can be categorized as single-partition transactions or cross-data-grid transactions.

.NET **8.6+** WebSphere eXtreme Scale Client for .NET supports single-partition transactions only.

Single-partition transactions

Single-partition transactions are the preferable method for interacting with caches that are hosted by WebSphere eXtreme Scale. When a transaction is limited to a single partition, then by default it is limited to a single Java virtual machine, and therefore a single server computer. A server can complete M number of these transactions per second, and if you have N computers, you can complete $M*N$ transactions per second. If your business increases and you need to perform twice as many of these transactions per second, you can double N by buying more computers. Then you can meet capacity demands without changing the application, upgrading hardware, or even taking the application offline.

In addition to letting the cache scale so significantly, single-partition transactions also maximize the availability of the cache. Each transaction only depends on one computer. Any of the other $(N-1)$ computers can fail without affecting the success or response time of the transaction. So if you are running 100 computers and one of them fails, only 1 percent of the transactions in flight at the moment that server failed are rolled back. After the server fails, WebSphere eXtreme Scale relocates the partitions that are hosted by the failed server to the other 99 computers. During this brief period, before the operation completes, the other 99 computers can still complete transactions. Only the transactions that would involve the partitions that are being relocated are blocked. After the failover process is complete, the cache can continue running, fully operational, at 99 percent of its original throughput capacity. After the failed server is replaced and returned to the data grid, the cache returns to 100 percent throughput capacity.

Cross-data-grid transactions

In terms of performance, availability and scalability, cross-data-grid transactions are the opposite of single-partition transactions. Cross-data-grid transactions access every partition and therefore every computer in the configuration. Each computer in the data grid is asked to look up some data and then return the result. The transaction cannot complete until every computer has responded, and therefore the throughput of the entire data grid is limited by the slowest computer. Adding computers does not make the slowest computer faster and therefore does not improve the throughput of the cache.

Cross-data-grid transactions have a similar effect on availability. Extending the previous example, if you are running 100 servers and one server fails, then 100 percent of the transactions that are in progress at the moment that server failed are rolled back. After the server fails, WebSphere eXtreme Scale starts to relocate the partitions that are hosted by that server to the other 99 computers. During this time, before the failover process completes, the data grid cannot process any of these transactions. After the failover process is complete, the cache can continue running, but at reduced capacity. If each computer in the data grid serviced 10 partitions, then 10 of the remaining 99 computers receive at least one extra partition as part of the failover process. Adding an extra partition increases the workload of that computer by at least 10 percent. Because the throughput of the data grid is limited to the throughput of the slowest computer in a cross-data-grid transaction, on average, the throughput is reduced by 10 percent.

Single-partition transactions are preferable to cross-data-grid transactions for scaling out with a distributed, highly available, object cache like WebSphere eXtreme Scale. Maximizing the performance of these kinds of systems requires the use of techniques that are different from traditional relational methodologies, but you can turn cross-data-grid transactions into scalable single-partition transactions.

Best practices for building scalable data models

The best practices for building scalable applications with products like WebSphere eXtreme Scale include two categories: foundational principles and implementation tips. Foundational principles are core ideas that need to be captured in the design of the data itself. An application that does not observe these principles is unlikely to scale well, even for its mainline transactions. Implementation tips are applied for problematic transactions in an otherwise well-designed application that observes the general principles for scalable data models.

Foundational principles

Some of the important means of optimizing scalability are basic concepts or principles to keep in mind.

Duplicate instead of normalizing

The key thing to remember about products like WebSphere eXtreme Scale is that they are designed to spread data across a large number of computers. If the goal is to make most or all transactions complete on a single partition, then the data model design needs to ensure that all the data the transaction might need is in the partition. Most of the time, the only way to achieve this is by duplicating data.

For example, consider an application like a message board. Two important transactions for a message board are showing all the posts from a user and all the posts on a topic. First, consider how these transactions would work with a normalized data model that contains a user record, a topic record, and a post record that contains the actual text. If posts are partitioned with user records, then displaying the topic becomes a cross-grid transaction, and vice versa. Topics and users cannot be partitioned together because they have a many-to-many relationship.

The best way to make this message board scale is to duplicate the posts, storing one copy with the topic record and one copy with the user record. Then, displaying the posts from a user is a single-partition transaction, displaying the posts on a topic is a single-partition transaction, and updating or deleting a post is a two-partition transaction. All three of these transactions scale linearly as the number of computers in the data grid increases.

Scalability rather than resources

The biggest obstacle to overcome when you are considering denormalized data models is the impact that these models have on resources. Keeping two, three, or more copies of some data can seem to use too many resources to be practical. When you are confronted with this scenario, remember the following facts: Hardware resources get cheaper every year. Second, and more importantly, WebSphere eXtreme Scale eliminates most hidden costs that are associated with deploying more resources.

Measure resources in terms of cost rather than computer terms such as megabytes and processors. Data stores that work with normalized relational data generally must be on the same computer. This required collocation means that a single larger enterprise computer must be purchased rather than several smaller computers. With enterprise hardware, it is not uncommon for one computer that is capable of completing one million transactions per second to cost much more than the combined cost of 10 computers capable of doing 100,000 transactions per second each.

A business cost in adding resources also exists. A growing business eventually runs out of capacity. When you run out of capacity, you either need to shut down while moving to a bigger, faster computer, or create a second production environment to which you can switch. Either way, additional costs will come in the form of lost business or maintaining almost twice the capacity needed during the transition period.

With WebSphere eXtreme Scale, the application does not need to be shut down to add capacity. If your business projects that you need 10 percent more capacity for the coming year, then increase the number of computers in the data grid by 10 percent. You can increase this percentage without application downtime and without purchasing excess capacity.

Avoid data transformations

When you are using WebSphere eXtreme Scale, data should be stored in a format that is directly consumable by the business logic. Breaking the data down into a more primitive form is costly. The transformation needs to be done when the data is written and when the data is read. With relational databases, this transformation is done out of necessity because the data is ultimately persisted to disk frequently. With WebSphere eXtreme Scale, these transformations are not necessary. Usually, data is stored in memory and can therefore be stored in the exact form that the application needs.

Observing this simple rule helps denormalize your data in accordance with the first principle. The most common type of transformation for business data is the JOIN operations that are necessary to turn normalized data into a result set that fits the needs of the application. Storing the data in the correct format implicitly avoids performing these JOIN operations and produces a denormalized data model.

Java *Eliminate unbounded queries*

No matter how well you structure your data, unbounded queries do not scale well. For example, do not have a transaction that asks for a list of all items that are sorted by value. This transaction might work at first when the total number of items is 1000, but when the total number of items reaches 10 million, the transaction returns all 10 million items. If you run this transaction, the two most likely outcomes are the transaction timing out, or the client encounters an out-of-memory error.

The best option is to alter the business logic so that only the top 10 or 20 items can be returned. This logic alteration keeps the size of the transaction manageable no matter how many items are in the cache.

Define schema

The main advantage of normalizing data is that the database system can take care of data consistency behind the scenes. When data is denormalized for scalability, this automatic data consistency management no longer exists. You must implement a data model that can work in the application layer or as a plug-in to the distributed data grid to guarantee data consistency.

Consider the message board example. If a transaction removes a post from a topic, then the duplicate post on the user record must be removed. Without a data model, it is possible a developer might write the application code to remove the post from the topic and forget to remove the post from the user record. However, if the developer is using a data model instead of interacting with the cache directly, the `removePost` method on the data model pulls the user ID from the post, looks up the user record, and removes the duplicate post behind the scenes.

Alternately, you can implement a listener that runs on the actual partition that detects the change to the topic and automatically adjusts the user record. A listener might be beneficial because the adjustment to the user record might happen locally if the partition happens to have the user record. If the user record is on a different partition, the transaction takes place between servers instead of between the client and server. The network connection between servers is likely to be faster than the network connection between the client and the server.

Avoid contention

Avoid scenarios such as having a global counter. The data grid does not scale if a single record is being used a disproportionate number of times compared to the rest of the records. The performance of the data grid is limited by the performance of the computer that holds the record.

In these situations, try to break up the record so it is managed per partition. For example, consider a transaction that returns the total number of entries in the distributed cache. Instead of having every insert and remove operation access a single record that increments, have a listener on each partition track the insert and remove operations. With this listener tracking, insert and remove can become single-partition operations.

Reading the counter becomes a cross-data-grid operation. Usually, it was already as inefficient as a cross-data-grid operation because its performance was tied to the performance of the computer that is hosting the record.

Implementation tips

You can also consider the following tips to achieve the best scalability.

Java *Use reverse-lookup indexes*

Consider a properly denormalized data model where customer records are partitioned based on the customer ID number. This partitioning method is the logical choice because nearly every business operation that is performed with the customer record uses the customer ID number. However, an important transaction that does not use the customer ID number is the login transaction. It is more common to have user names or email addresses for login instead of customer ID numbers.

The simple approach to the login scenario is to use a cross-data-grid transaction to find the customer record. As explained previously, this approach does not scale.

The next option might be to partition on user name or email. This option is not practical because all the customer ID-based operations become cross-data-grid transactions. Also, the customers on your site might want to change their user name or email address. Products like WebSphere eXtreme Scale need the value that is used to partition the data to remain constant.

The correct solution is to use a reverse lookup index. With WebSphere eXtreme Scale, a cache can be created in the same distributed grid as the cache that holds all the user records. This cache is highly available, partitioned, and scalable. This cache can be used to map a user name or email address to a customer ID. This cache turns login into a two partition operation instead of a cross-grid operation. This scenario is not as good as a single-partition transaction, but the throughput still scales linearly as the number of computers increases.

Compute at write time

Commonly calculated values like averages or totals can be expensive to produce because these operations usually require reading a large number of entries. Because reads are more common than writes in most applications, it is efficient to compute these values at write time and then store the result in the cache. This practice makes read operations both faster and more scalable.

Optional fields

Consider a user record that holds a business, home, and telephone number. A user might have all, none or any combination of these numbers defined. If the data were normalized, then a user table and a telephone number table would exist. The telephone numbers for a user can be found with a JOIN operation between the two tables.

De-normalizing this record does not require data duplication, because most users do not share telephone numbers. Instead, empty slots in the user record must be allowed. Instead of having a telephone number table, add three attributes to each user record, one for each telephone number type. This addition of attributes eliminates the JOIN operation and makes a telephone number lookup for a user a single-partition operation.

Placement of many-to-many relationships

Consider an application that tracks products and the stores in which the products are sold. A single product is sold in many stores, and a single store sells many products. Assume that this application tracks 50 large retailers. Each product is sold in a maximum of 50 stores. Each store sells thousands of products.

Keep a list of stores inside the product entity (arrangement A), instead of keeping a list of products inside each store entity (arrangement B). Looking at some of the transactions this application must run illustrates why arrangement A is more scalable.

First look at updates. With arrangement A, removing a product from the inventory of a store locks the product entity. If the data grid holds 10000 products, only 1/10000 of the grid must be locked to complete the update. With arrangement B, the data grid only contains only 50 stores, so 1/50 of the data grid must be locked to complete the update. So even though both of these updates might be considered single-partition operations, arrangement A scales out more efficiently.

Now, considering reads with arrangement A, looking up the stores at which a product is sold is a single-partition transaction that scales and is fast because the transaction only transmits a small amount of data. With arrangement B, this transaction becomes a cross-data-grid transaction because each store entity must be accessed to see if the product is sold at that store, which reveals an enormous performance advantage for arrangement A.

Java *Scaling with normalized data*

One legitimate use of cross-data-grid transactions is to scale data processing. If a data grid has 5 computers and a cross-data-grid transaction is dispatched that sorts through about 100,000 records on each computer, then that transaction sorts through 500,000 records. If the slowest computer in the data grid can perform 10 of these transactions per second, then the data grid is capable of sorting through 5,000,000 records per second. If the data in the grid doubles, then each computer must sort through 200,000 records, and each transaction sorts through 1,000,000 records. This data increase decreases the throughput of the slowest computer to 5 transactions per second, reducing the throughput of the data grid to 5 transactions per second. Still, the data grid sorts through 5,000,000 records per second.

In this scenario, doubling the number of computers allows each computer to return to its previous load of sorting through 100,000 records, allowing the slowest computer to process 10 of these transactions per second. The throughput of the data grid stays the same at 10 requests per second, but now each transaction processes 1,000,000 records. As a result, the data grid doubled its capacity in terms of processing records to 10,000,000 per second.

Applications such as a search engine that needs to scale both in terms of data processing to accommodate the increasing size of the Internet and throughput to accommodate growth in the number of users, you must create multiple data grids, with a round robin of the requests between the data grids. If you must scale up the throughput, add computers and add another data grid to service requests. If data processing must be scaled up, add more computers and keep the number of data grids constant.

Scaling in units or pods

Although you can deploy a data grid over thousands of Java virtual machines, you might consider splitting the data grid into units or pods to increase the reliability and ease of testing of your configuration. A pod is a group of servers that is running the same set of applications.

Deploying a large single data grid

Testing has verified that eXtreme Scale can scale out to over 1000 JVMs. Such testing encourages building applications to deploy single data grids on large numbers of boxes. Although it is possible to do this, it is not recommended, for several reasons:

1. **Budget concerns:** Your environment cannot realistically test a 1000-server data grid. However, it can test a much smaller data grid considering budget reasons, so you do not need to buy twice the hardware, especially for such a large number of servers.
2. **Different application versions:** Requiring a large number of boxes for each testing thread is not practical. The risk is that you are not testing the same factors as you would in a production environment.
3. **Data loss:** Running a database on a single hard drive is unreliable. Any problem with the hard drive causes you to lose data. Running a growing application on a single data grid is similar. You will likely have bugs in your environment and in your applications. So placing all of the data on a single large system will often lead to a loss of large amounts of data.

Splitting the data grid

Splitting the application data grid into pods (units) is a more reliable option. A pod is a group of servers that are running a homogenous application stack. Pods can be of any size, but ideally they should consist of about 20 physical servers. Instead of having 500 physical servers in a single data grid, you can have 25 pods of 20 physical servers. A single version of an application stack should run on a given pod, but different pods can have their own versions of an application stack.

Generally, an application stack considers levels of the following components.

- Operating system
- Hardware
- JVM
- WebSphere eXtreme Scale version
- Application
- Other necessary components

A pod is a conveniently sized deployment unit for testing. Instead of having hundreds of servers for testing, it is more practical to have 20 servers. In this case, you are still testing the same configuration as you would have in production. Production uses grids with a maximum size of 20 servers, constituting a pod. You can stress-test a single pod and determine its capacity, number of users, amount of data, and transaction throughput. This makes planning easier and follows the standard of having predictable scaling at predictable cost.

Setting up a pod-based environment

In different cases, the pod does not necessarily have to have 20 servers. The purpose of the pod size is for practical testing. The size of a pod should be small enough that if a pod encounters problems in production, the fraction of transactions affected is tolerable.

Ideally, any bug impacts a single pod. A bug would only have an impact on four percent of the application transactions rather than 100 percent. In addition, upgrades are easier because they can be rolled out one pod at a time. As a result, if an upgrade to a pod creates problems, the user can switch that pod back to the prior level. Upgrades include any changes to the application, the application stack, or system updates. As much as possible, upgrades should only change a single element of the stack at a time to make problem diagnosis more precise.

To implement an environment with pods, you need a routing layer above the pods that is forwards and backwards compatible if pods get software upgrades. Also, you should create a directory that includes information about which pod has what data. You can use another eXtreme Scale data grid for this with a database behind it, preferably using the write-behind scenario.) This yields a two-tier solution. Tier 1 is the directory and is used to locate which pod handles a specific transaction. Tier 2 is composed of the pods themselves. When tier 1 identifies a pod, the setup routes each transaction to the correct server in the pod, which is usually the server holding the partition for the data used by the transaction. Optionally, you can also use a near cache on tier 1 to lower the impact associated with looking up the correct pod.

Using pods is slightly more complex than having a single data grid, but the operational, testing, and reliability improvements make it a crucial part of scalability testing.

Availability overview

High availability

With high availability, WebSphere eXtreme Scale provides reliable data redundancy and detection of failures.

WebSphere eXtreme Scale self-organizes data grids of Java virtual machines into a loosely federated tree. The catalog service at the root and core groups holding containers are at the leaves of the tree. See “Caching architecture: Maps, containers, clients, and catalogs” on page 16 for more information.

Each core group is automatically created by the catalog service into groups of about 20 servers. The core group members provide health monitoring for other members of the group. Also, each core group elects a member to be the leader for communicating group information to the catalog service. Limiting the core group size allows for good health monitoring and a highly scalable environment.

Note: In a WebSphere Application Server environment, in which core group size can be altered, eXtreme Scale does not support more than 50 members per core group.

Heart beating

1. Sockets are kept open between Java virtual machines, and if a socket closes unexpectedly, this unexpected closure is detected as a failure of the peer Java

virtual machine. This detection catches failure cases such as the Java virtual machine exiting very quickly. Such detection also allows recovery from these types of failures typically in less than a second.

2. Other types of failures include: an operating system panic, physical server failure, or network failure. These failures are discovered through heart beating.

Heartbeats are sent periodically between pairs of processes: When a fixed number of heartbeats are missed, a failure is assumed. This approach detects failures in $N \times M$ seconds. N is the number of missed heart beats and M is the heartbeat interval. Directly specifying M and N is not supported. A slider mechanism is used to allow a range of tested M and N combinations to be used.

Failures

There are several ways that a process can fail. The process could fail because some resource limit was reached, such as maximum heap size, or some process control logic terminated a process. The operating system could fail, causing all of the processes running on the system to be lost. Hardware can fail, though less frequently, like the network interface card (NIC), causing the operating system to be disconnected from the network. Many more points of failure can occur, causing the process to be unavailable. In this context, all of these failures can be categorized into one of two types: process failure and loss of connectivity.

Process failure

WebSphere eXtreme Scale reacts to process failures quickly. When a process fails, the operating system is responsible for cleaning up any left over resources that the process was using. This cleanup includes port allocation and connectivity. When a process fails, a signal is sent over the connections that were being used by that process to close each connection. With these signals, a process failure can be instantly detected by any other process that is connected to the failed process.

Loss of connectivity

Loss of connectivity occurs when the operating system becomes disconnected. As a result, the operating system cannot send signals to other processes. There are several reasons that loss of connectivity can occur, but they can be split into two categories: host failure and islanding.

Host failure

If the machine is unplugged from the power outlet, then it is gone instantly.

Islanding

This scenario presents the most complicated failure condition for software to handle correctly because the process is presumed to be unavailable, though it is not. Essentially, a server or other process appears to the system to have failed while it is actually running properly.

Container failures

Container failures are generally discovered by peer containers through the core group mechanism. When a container or set of containers fails, the catalog service migrates the shards that were hosted on that container or containers. The catalog service looks for a synchronous replica first before migrating to an asynchronous

replica. After the primary shards are migrated to new host containers, the catalog service looks for new host containers for the replicas that are now missing.

Note: Container islanding - The catalog service migrates shards off containers when the container is discovered to be unavailable. If those containers then become available, the catalog service considers the containers eligible for placement just like in the normal startup flow.

Container failure detection latency

Failures can be categorized into soft and hard failures. Soft failures are typically caused when a process fails. Such failures are detected by the operating system, which can recover used resources, such as network sockets, quickly. Typical failure detection for soft failures is less than one second. Hard failures might take up to 200 seconds to detect with the default heart beat tuning. Such failures include: physical machine crashes, network cable disconnects, or operating system failures. The run time relies on heart beating to detect hard failures which can be configured.

Catalog service failure

Because the catalog service grid is an eXtreme Scale grid, it also uses the core grouping mechanism in the same way as the container failure process. The primary difference is that the catalog service domain uses a peer election process for defining the primary shard instead of the catalog service algorithm that is used for the containers.

The placement service and the core grouping service are One of N services. A One of N service runs in one member of the high availability group. The location service and administration run in all of the members of the high availability group. The placement service and core grouping service are singletons because they are responsible for laying out the system. The location service and administration are read-only services and exist everywhere to provide scalability.

The catalog service uses replication to make itself fault tolerant. If a catalog service process fails, then the service restarts to restore the system to the wanted level of availability. If all of the processes that are hosting the catalog service fail, the data grid has a loss of critical data. This failure results in a required restart of all the container servers. Because the catalog service can run on many processes, this failure is an unlikely event. However, if you are running all of the processes on a single box, within a single blade chassis, or from a single network switch, a failure is more likely to occur. Try to remove common failure modes from boxes that are hosting the catalog service to reduce the possibility of failure.

Multiple container failures

A replica is never placed in the same process as its primary because if the process is lost, it would result in a loss of both the primary and the replica. In a development environment on a single machine, you might want to have two containers and replicate between them. You can define the development mode attribute in the deployment policy to configure a replica to be placed on the same machine as a primary. However, in production, using a single machine is not sufficient because loss of that host results in the loss of both container servers. To change between development mode on a single machine and a production mode with multiple machines, disable development mode in the deployment policy configuration file.

Table 2. Failure discovery and recovery summary

Loss type	Discovery (detection) mechanism	Recovery method
Process loss	I/O	Restart
Server loss	Heartbeat	Restart
Network outage	Heartbeat	Reestablish network and connection
Server-side hang	Heartbeat	Stop and restart server
Server busy	Heartbeat	Wait until server is available

Replication for availability

Replication provides fault tolerance and increases performance for a distributed eXtreme Scale topology. Replication is enabled by associating backing maps with a map set.

About map sets

A map set is a collection of maps that are categorized by a partition-key. This partition-key is derived from the key on the individual map by taking its hash modulo the number of partitions. If one group of maps within the map set has partition-key X, those maps are stored in a corresponding partition X in the data grid. If another group has partition-key Y, all of the maps are stored in partition Y, and so on. The data within the maps is replicated based on the policy defined on the map set. Replication occurs on distributed topologies.

Map sets are assigned the number of partitions and a replication policy. The map set replication configuration identifies the number of synchronous and asynchronous replica shards for the map set must in addition to the primary shard. For example, if one synchronous and one asynchronous replica exist, all of the BackingMaps that are assigned to the map set each have a replica shard distributed automatically within the set of available container servers for the data grid. The replication configuration can also enable clients to read data from synchronously replicated servers. This can spread the load for read requests over additional servers in the eXtreme Scale. Replication has a programming model impact only when preloading the backing maps.

Map preloading

Maps can be associated with Loaders. A loader is used to fetch objects when they cannot be found in the map (a cache miss) and also to write changes to a back-end when a transaction commits. Loaders can also be used for preloading data into a map. The preloadMap method of the Loader interface is called on each map when its corresponding partition in the map set becomes a primary. The preloadMap method is not called on replicas. It attempts to load all the intended referenced data from the back-end into the map using the provided session. The relevant map is identified by the BackingMap argument that is passed to the preloadMap method.

```
void preloadMap(Session session, BackingMap backingMap) throws LoaderException;
```

Preloading in partitioned map set

Maps can be partitioned into N partitions. Maps can therefore be striped across multiple servers, with each entry identified by a key that is stored only on one of those servers. Very large maps can be held in an eXtreme Scale because the

application is no longer limited by the heap size of a single JVM to hold all the entries of a Map. Applications that want to preload with the `preloadMap` method of the Loader interface must identify the subset of the data that it preloads. A fixed number of partitions always exists. You can determine this number by using the following code example:

```
int numPartitions = backingMap.getPartitionManager().getNumOfPartitions();
int myPartition = backingMap.getPartitionId();
```

This code example shows that an application can identify the subset of the data to preload from the database. Applications must always use these methods even when the map is not initially partitioned. These methods allow flexibility: If the map is later partitioned by the administrators, then the loader continues to work correctly.

The application must issue queries to retrieve the *myPartition* subset from the backend. If a database is used, then it might be easier to have a column with the partition identifier for a given record unless there is some natural query that allows the data in the table to partition easily.

Performance

The preload implementation copies data from the back-end into the map by storing multiple objects in the map in a single transaction. The optimal number of records to store per transaction depends on several factors, including complexity and size. For example, after the transaction includes blocks of more than 100 entries, the performance benefit decreases as you increase the number of entries. To determine the optimal number, begin with 100 entries and then increase the number until the performance benefit decreases to none. Larger transactions result in better replication performance. Remember, only the primary runs the preload code. The preloaded data is replicated from the primary to any replicas that are online.

Preloading map sets

If the application uses a map set with multiple maps then each map has its own loader. Each loader has a preload method. Each map is loaded serially by the eXtreme Scale. It might be more efficient to preload all the maps by designating a single map as the preloading map. This process is an application convention. For example, two maps, department and employee, might use the department Loader to preload both the department and the employee maps. This procedure ensures that, transactionally, if an application wants a department then the employees for that department are in the cache. When the department Loader preloads a department from the back-end, it also fetches the employees for that department. The department object and its associated employee objects are then added to the map using a single transaction.

Recoverable preloading

Some customers have very large data sets that need caching. Preloading this data can be very time consuming. Sometimes, the preloading must complete before the application can go online. You can benefit from making preloading recoverable. Suppose there are a million records to preload. The primary is preloading them and fails at the 800,000th record. Normally, the replica chosen to be the new primary clears any replicated state and starts from the beginning. eXtreme Scale can use a `ReplicaPreloadController` interface. The loader for the application would also need to implement the `ReplicaPreloadController` interface. This example adds a single method to the Loader: `Status checkPreloadStatus(Session session,`

BackingMap bmap); This method is called by the eXtreme Scale run time before the preload method of the Loader interface is normally called. The eXtreme Scale tests the result of this method (Status) to determine its behavior whenever a replica is promoted to a primary.

Table 3. Status value and response

Returned status value	eXtreme Scale response
Status.PRELOADED_ALREADY	eXtreme Scale does not call the preload method at all because this status value indicates that the map is fully preloaded.
Status.FULL_PRELOAD_NEEDED	eXtreme Scale clears the map and calls the preload method normally.
Status.PARTIAL_PRELOAD_NEEDED	eXtreme Scale leaves the map as-is and calls preload. This strategy allows the application loader to continue preloading from that point onwards.

Clearly, while a primary is preloading the map, it must leave some state in a map in the map set that is being replicated so that the replica determines what status to return. You can use an extra map named, for example, RecoveryMap. This RecoveryMap must be part of the same map set that is being preloaded to ensure that the map is replicated consistently with the data being preloaded. A suggested implementation follows.

As the preload commits each block of records, the process also updates a counter or value in the RecoveryMap as part of that transaction. The preloaded data and the RecoveryMap data are replicated atomically to the replicas. When the replica is promoted to primary, it can now check the RecoveryMap to see what has happened.

The RecoveryMap can hold a single entry with the state key. If no object exists for this key then you need a full preload (checkPreloadStatus returns FULL_PRELOAD_NEEDED). If an object exists for this state key and the value is COMPLETE, the preload completes, and the checkPreloadStatus method returns PRELOADED_ALREADY. Otherwise, the value object indicates where the preload restarts and the checkPreloadStatus method returns: PARTIAL_PRELOAD_NEEDED. The loader can store the recovery point in an instance variable for the loader so that when preload is called, the loader knows the starting point. The RecoveryMap can also hold an entry per map if each map is preloaded independently.

Handling recovery in synchronous replication mode with a Loader

The eXtreme Scale run time is designed not to lose committed data when the primary fails. The following section shows the algorithms used. These algorithms apply only when a replication group uses synchronous replication. A loader is optional.

The eXtreme Scale run time can be configured to replicate all changes from a primary to the replicas synchronously. When a synchronous replica is placed, it receives a copy of the existing data on the primary shard. During this time, the primary continues to receive transactions and copies them to the replica asynchronously. The replica is not considered to be online at this time.

After the replica catches up the primary, the replica enters peer mode and synchronous replication begins. Every transaction committed on the primary is sent to the synchronous replicas and the primary waits for a response from each replica. A synchronous commit sequence with a Loader on the primary looks like the following set of steps:

Table 4. Commit sequence on the primary

Step with loader	Step without loader
Get locks for entries	same
Flush changes to the loader	no-op
Save changes to the cache	same
Send changes to replicas and wait for acknowledgment	same
Commit to the loader through the TransactionCallback plug-in	Plug-in commit called, but does nothing
Release locks for entries	same

Notice that the changes are sent to the replica before they are committed to the loader. To determine when the changes are committed on the replica, revise this sequence: At initialize time, initialize the tx lists on the primary as below.

CommittedTx = {}, RolledBackTx = {}

During synchronous commit processing, use the following sequence:

Table 5. Synchronous commit processing

Step with loader	Step without loader
Get locks for entries	same
Flush changes to the loader	no-op
Save changes to the cache	same
Send changes with a committed transaction, roll back transaction to replica, and wait for acknowledgment	same
Clear list of committed transactions and rolled back transactions	same
Commit the loader through the TransactionCallBack plug-in	TransactionCallBack plug-in commit is still called, but typically does not do anything
If commit succeeds, add the transaction to the committed transactions, otherwise add to the rolled back transactions	no-op
Release locks for entries	same

For replica processing, use the following sequence:

1. Receive changes
2. Commit all received transactions in the committed transaction list
3. Roll back all received transactions in the rolled back transaction list
4. Start a transaction or session
5. Apply changes to the transaction or session
6. Save the transaction or session to the pending list
7. Send back reply

Notice that on the replica, no loader interactions occur while the replica is in replica mode. The primary must push all changes through the Loader. The replica does not push any changes. A side effect of this algorithm is that the replica always has the transactions, but they are not committed until the next primary

transaction sends the commit status of those transactions. The transactions are then committed or rolled back on the replica. Until then, the transactions are not committed. You can add a timer on the primary that sends the transaction outcome after a small period (a few seconds). This timer limits, but does not eliminate, any staleness to that time window. This staleness is only a problem when using replica read mode. Otherwise, the staleness does not have an impact on the application.

When the primary fails, it is likely that a few transactions were committed or rolled back on the primary, but the message never made it to the replica with these outcomes. When a replica is promoted to the new primary, one of the first actions is to handle this condition. Each pending transaction is reprocessed against the new primary's set of maps. If there is a Loader, then each transaction is given to the Loader. These transactions are applied in strict first in first out (FIFO) order. If a transaction fails, it is ignored. If three transactions are pending, A, B, and C, then A might commit, B might rollback, and C might also commit. No one transaction has any impact on the others. Assume that they are independent.

A loader might want to use slightly different logic when it is in failover recovery mode versus normal mode. The loader can easily know when it is in failover recovery mode by implementing the `ReplicaPreloadController` interface. The `checkPreloadStatus` method is only called when failover recovery completes. Therefore, if the `apply` method of the Loader interface is called before the `checkPreloadStatus` method, then it is a recovery transaction. After the `checkPreloadStatus` method is called, the failover recovery is complete.

Load balancing across replicas

The eXtreme Scale, unless configured otherwise, sends all read and write requests to the primary server for a given replication group. The primary must service all requests from clients. You might want to allow read requests to be sent to replicas of the primary. Sending read requests to the replicas allows the load of the read requests to be shared by multiple Java Virtual Machines (JVM). However, using replicas for read requests can result in inconsistent responses.

Load balancing across replicas is typically used only when clients are caching data that is changing all the time or when the clients are using pessimistic locking.

If the data is continually changing and then being invalidated in client near caches, the primary should see a relatively high get request rate from clients as a result. Likewise, in pessimistic locking mode, no local cache exists, so all requests are sent to the primary.

If the data is relatively static or if pessimistic mode is not used, then sending read requests to the replica does not have a large impact on performance. The frequency of get requests from clients with caches that are full of data is not high.

When a client first starts, its near cache is empty. Cache requests to the empty cache are forwarded to the primary. The client cache gets data over time, causing the request load to drop. If many clients start concurrently, then the load might be significant and replica read might be an appropriate performance choice.

Client-side replication

With eXtreme Scale, you can replicate a server map to one or more clients by using asynchronous replication. A client can request a local read-only copy of a server side map by using the `ClientReplicableMap.enableClientReplication` method.

```
void enableClientReplication(Mode mode, int[] partitions,
ReplicationMapListener listener) throws ObjectGridException;
```

The first parameter is the replication mode. This mode can be a continuous replication or a snapshot replication. The second parameter is an array of partition IDs that represent the partitions from which to replicate the data. If the value is null or an empty array, the data is replicated from all the partitions. The last parameter is a listener to receive client replication events. See `ClientReplicableMap` and `ReplicationMapListener` in the API documentation for details.

After the replication is enabled, then the server starts to replicate the map to the client. The client is eventually only a few transactions behind the server at any point in time.

High availability catalog service

A catalog service domain is the data grid of catalog servers you are using, which retain topology information for all of the container servers in your eXtreme Scale environment. The catalog service controls balancing and routing for all clients.

For more information about catalog servers, see “Catalog service” on page 16.

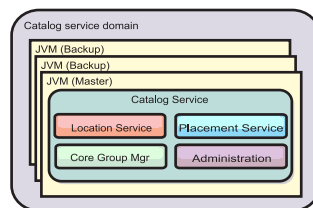


Figure 34. Catalog service domain

When multiple catalog servers start, one of the servers is elected as the master catalog server that accepts heartbeats and handles system data changes in response to any catalog service or container changes.

Configure at least three catalog servers in the catalog service domain. Catalog servers must be installed on separate nodes or separate installation images from your container servers to ensure that you can seamlessly upgrade your servers at a later date. If your configuration has zones, you can configure one catalog server per zone.

When a container server contacts one of the catalog servers, the routing table for the catalog service domain is also propagated to the catalog server and container server through the CORBA service context. Furthermore, if the contacted catalog server is not currently the master catalog server, the request is automatically rerouted to the current master catalog server and the routing table for the catalog server is updated.

Note: A catalog service domain and the container server data grid are very different. The catalog service domain is for high availability of your system data. The container server data grid is for your data high availability, scalability, and workload management. Therefore, two different routing tables exist: the routing table for the catalog service domain and the routing table for the container server data grid shards.

Catalog server quorums

When the quorum mechanism is enabled, all the catalog servers in the quorum must be available for placement operations to occur in the data grid.

- “Important terms”
- “Heartbeats and failure detection”
- “Quorum behavior” on page 103
 - “Container behavior during quorum loss” on page 105
- “Client behavior during quorum loss” on page 106

Important terms

- **Heartbeat:** A signal that is sent between servers to convey that they are running.
- **Quorum:** A group of catalog servers that communicate and conduct placement operations in the data grid. This group consists of all of the catalog servers in the data grid, unless you manually override the quorum mechanism with administrative actions.
- **Brownout:** A temporary loss of connectivity between one or more servers.
- **Blackout:** A permanent loss of connectivity between one or more servers.
- **Data center:** A geographically located group of servers that are generally connected with a local area network (LAN).
- **Zone:** A zone is a configuration option that is used to group servers together that share some physical characteristic. Examples of zones for a group of servers include: a data center, an area network, a building, or a floor of a building.

Heartbeats and failure detection

Container servers and core groups

The catalog service places container servers into core groups of a limited size. A core group tries to detect the failure of its members. A single member of a core group is elected to be the core group leader. The core group leader periodically tells the catalog service that the core group is alive and reports any membership changes to the catalog service. A membership change can be a JVM failing or a newly added JVM that joins the core group.

If a JVM socket is closed, that JVM is regarded as being no longer available. Each core group member also heart beats over these sockets at a rate determined by configuration. If a JVM does not respond to these heartbeats within a configured maximum time period, then the JVM is considered to be no longer available, which triggers a failure detection.

If the catalog service marks a container JVM as failed and the container server is later reported as being available, the container JVM is told to shut down the WebSphere eXtreme Scale container servers. A JVM in this state is not visible in `xscmd` utility command queries. Messages in the logs of the container JVM indicate that the container JVM has failed. You must manually restart these JVMs.

If the core group leader cannot contact any member, it continues to retry contacting the member.

The complete failure of all members of a core group is also a possibility. If the entire core group has failed, it is the responsibility of the catalog service to detect this loss.

Catalog service domain heart-beating

The catalog service domain looks like a private core group with a static membership and a quorum mechanism. It detects failures the same way as a normal core group. However, the behavior is modified to include quorum logic. The catalog service also uses a less aggressive heart-beating configuration.

Failure detection

WebSphere eXtreme Scale detects when processes terminate through abnormal socket closure events. The catalog service is notified immediately when a process terminates.

For more information about configuring heart-beating, see *Tuning the heartbeat interval setting for failover detection* and the information about configuring failover detection in the *Administration Guide*.

Quorum behavior

Normally, the members of the catalog service have full connectivity. The catalog service domain is a static set of JVMs. WebSphere eXtreme Scale expects all members of the catalog service to be online. When all the members are online, the catalog service has quorum. The catalog service responds to container events only while the catalog service has quorum.

Reasons for quorum loss

WebSphere eXtreme Scale expects to lose quorum for the following scenarios:

- A catalog service JVM member fails
- Network brown out occurs
- Data center loss occurs

WebSphere eXtreme Scale does not lose quorum in the following scenario:

- Stopping a catalog server instance with the **stopOgServer** command or any other administrative actions. The system knows that the server instance has stopped, which is different from a JVM failure or brownout.

If the catalog service loses a quorum, it waits for quorum to be reestablished. While the catalog service does not have a quorum, it ignores events from container servers. Container servers continue to try any requests that are rejected by the catalog server during this time. Heart-beating is suspended until a quorum is reestablished.

Quorum loss from JVM failure

A catalog server that fails causes quorum to be lost. If a JVM fails, quorum can be reestablished by either overriding quorum or by restarting the failed catalog server.

Quorum loss from network brownout

WebSphere eXtreme Scale is designed to expect the possibility of brownouts. A brownout is when a temporary loss of connectivity occurs between data centers. Brown outs are usually transient and clear within seconds or minutes. While WebSphere eXtreme Scale tries to maintain normal operation during the brownout

period, a brownout is regarded as a single failure event. The failure is expected to be fixed and then normal operation resumes with no actions necessary.

A long duration brown out can be classified as a blackout only through user intervention. Overriding quorum on one side of the brownout is required in order for the event to be classified as a blackout.

Catalog service JVM cycling

If a catalog server is stopped by using the **stopOgServer** command, then the quorum drops to one less server. The remaining servers still have quorum. Restarting the catalog server sets quorum back to the previous number.

Consequences of lost quorum

If a container JVM was to fail while quorum is lost, recovery does not occur until the brownout recovers. In a blackout scenario, the recovery does not occur until you run the override quorum command. Quorum loss and a container failure as are considered a double failure, which is a rare event. Because of the double failure, applications might lose write access to data that was stored on the failed JVM. When quorum is restored, the normal recovery occurs.

Similarly, if you attempt to start a container during a quorum loss event, the container does not start.

Full client connectivity is allowed during quorum loss. If no container failures or connectivity issues happen during the quorum loss event then clients can still fully interact with the container servers.

If a brownout occurs, then some clients might not have access to primary or replica copies of the data until the brownout clears.

New clients can be started because a catalog service JVM must exist in each data center. Therefore, at least one catalog server can be reached by a client even during a brownout event.

Quorum recovery

If quorum is lost for any reason, when quorum is reestablished, a recovery protocol is run. When the quorum loss event occurs, all liveness checking for core groups is suspended and failure reports are also ignored. After quorum is back, then the catalog service checks all the core groups to immediately determine their membership. Any shards previously hosted on container JVMs reported as failed are recovered. If primary shards were lost, then surviving replicas are promoted to being primary shards. If replica shards were lost then additional replicas shards are created on the survivors.

Overriding quorum

Override quorum only when a data center failure has occurred. Quorum loss due to a catalog service JVM failure or a network brownout recovers automatically after the catalog service JVM is restarted or the network brownout ends.

Administrators are the only ones with knowledge of a data center failure. WebSphere eXtreme Scale treats a brownout and a blackout similarly. You must inform the WebSphere eXtreme Scale environment of such failures with the **xscmd**

-c overrideQuorum command. This command tells the catalog service to assume that quorum is achieved with the current membership, and full recovery takes place. When issuing an override quorum command, you are guaranteeing that the JVMs in the failed data center have truly failed and do not have a chance of recovering.

The following list considers some scenarios for overriding quorum. In this scenario, you have three catalog servers: A, B, and C.

- **Brown out:** The C catalog server is isolated temporarily. The catalog service loses quorum and waits for the brownout to complete. After the brownout is over, the C catalog server rejoins the catalog service domain and quorum is reestablished. Your application sees no problems during this time.
- **Temporary failure:** During a temporary failure, the C catalog server fails and the catalog service loses quorum. You must override quorum. After quorum is reestablished, you can restart the C catalog server. The C catalog server joins the catalog service domain again when it restarts. Your application sees no problems during this time.
- **Data center failure:** You verify that the data center has failed and that it has been isolated on the network. Then you issue the `xsCmd -c overrideQuorum` command. The surviving two data centers run a full recovery by replacing shards that were hosted in the failed data center. The catalog service is now running with a full quorum of the A and B catalog servers. The application might see delays or exceptions during the interval between the start of the blackout and when quorum is overridden. After quorum is overridden, the data grid recovers and normal operation is resumed.
- **Data center recovery:** The surviving data centers are already running with quorum overridden. When the data center that contains the C catalog server is restarted, all JVMs in the data center must be restarted. Then the C catalog server joins the existing catalog service domain again and the quorum setting reverts to the normal situation with no user intervention.
- **Data center failure and brownout:** The data center that contains the C catalog server fails. Quorum is overridden and recovered on the remaining data centers. If a brownout between the A and B catalog servers occurs, the normal brownout recovery rules apply. After the brownout clears, quorum is reestablished and necessary recovery from the quorum loss occurs.

Container behavior during quorum loss

Containers host one or more shards. Shards are either primaries or replicas for a specific partition. The catalog service assigns shards to a container and the container server uses that assignment until new instructions arrive from the catalog service. For example, a primary shard continues to try communication with its replica shards during network brownouts, until the catalog service provides further instructions to the primary shard.

Synchronous replica behavior

The primary shard can accept new transactions while the connection is broken if the number of replicas online are at least at the `minsyc` property value for the map set. If any new transactions are processed on the primary shard while the link to the synchronous replica is broken, the replica is and resynchronized with the current state of the primary when the link is reestablished.

Do not configure synchronous replication between data centers or over a WAN-style link.

Asynchronous replica behavior

While the connection is broken, the primary shard can accept new transactions. The primary shard buffers the changes up to a limit. If the connection with the replica is reestablished before that limit is reached then the replica is updated with the buffered changes. If the limit is reached, then the primary destroys the buffered list and when the replica reattaches then it is cleared and resynchronized.

Client behavior during quorum loss

Clients are always able to connect to the catalog server to bootstrap to the data grid whether the catalog service domain has quorum or not. The client tries to connect to any catalog server instance to obtain a route table and then interact with the data grid. Network connectivity might prevent the client from interacting with some partitions due to network setup. The client might connect to local replicas for remote data if it has been configured to do so. Clients cannot update data if the primary partition for that data is not available.

Replicas and shards

With eXtreme Scale, an in-memory database or shard can be replicated from one Java virtual machine (JVM) to another. A shard represents a partition that is placed on a container. Multiple shards that represent different partitions can exist on a single container. Each partition has an instance that is a primary shard and a configurable number of replica shards. The replica shards are either synchronous or asynchronous. The types and placement of replica shards are determined by eXtreme Scale using a deployment policy, which specifies the minimum and maximum number of synchronous and asynchronous shards.

Shard types

Replication uses three types of shards:

- Primary
- Synchronous replica
- Asynchronous replica

The primary shard receives all insert, update and remove operations. The primary shard adds and removes replicas, replicates data to the replicas, and manages commits and rollbacks of transactions.

Synchronous replicas maintain the same state as the primary. When a primary replicates data to a synchronous replica, the transaction is not committed until it commits on the synchronous replica.

Asynchronous replicas might or might not be at the same state as the primary. When a primary replicates data to an asynchronous replica, the primary does not wait for the asynchronous replica to commit.

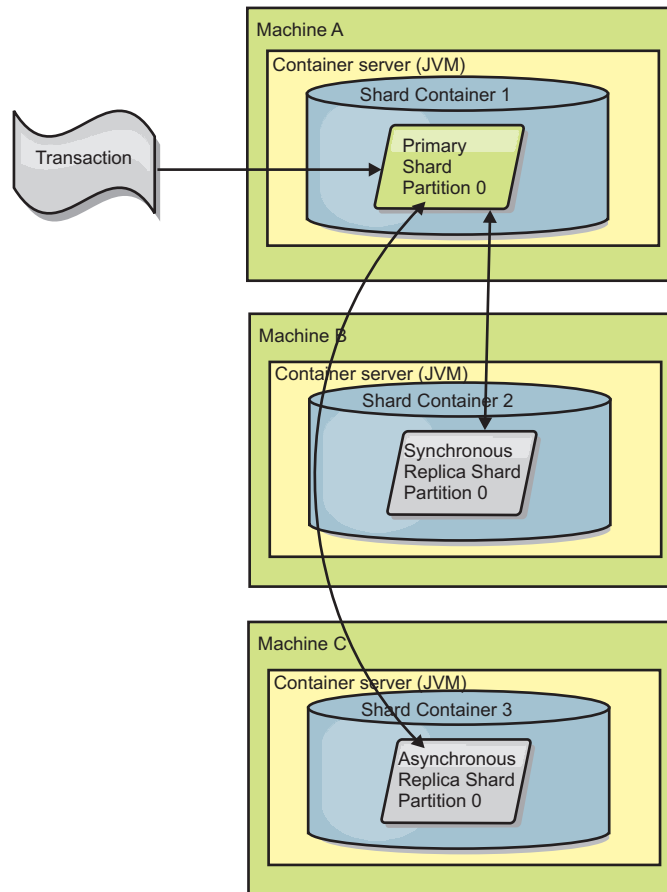


Figure 35. Communication path between a primary shard and replica shards

Minimum synchronous replica shards

When a primary prepares to commit data, it checks how many synchronous replica shards voted to commit the transaction. If the transaction processes normally on the replica, it votes to commit. If something went wrong on the synchronous replica, it votes not to commit. Before a primary commits, the number of synchronous replica shards that are voting to commit must meet the `minSyncReplica` setting from the deployment policy. When the number of synchronous replica shards that are voting to commit is too low, the primary does not commit the transaction and an error results. This action ensures that the required number of synchronous replicas are available with the correct data. Synchronous replicas that encountered errors reregister to fix their state. For more information about reregistering, see [Replica shard recovery](#).

The primary throws a `ReplicationVotedToRollbackTransactionException` error if too few synchronous replicas voted to commit.

Replication and Loaders

Normally, a primary shard writes changes synchronously through the Loader to a database. The Loader and database are always in sync. When the primary fails over to a replica shard, the database and Loader might not be in sync. For example:

- The primary can send the transaction to the replica and then fail before committing to the database.

- The primary can commit to the database and then fail before sending to the replica.

Either approach leads to either the replica being one transaction in front of or behind the database. This situation is not acceptable. eXtreme Scale uses a special protocol and a contract with the Loader implementation to solve this issue without two phase commit. The protocol follows:

Primary side

- Send the transaction along with the previous transaction outcomes.
- Write to the database and try to commit the transaction.
- If the database commits, then commit on eXtreme Scale. If the database does not commit, then roll back the transaction.
- Record the outcome.

Replica side

- Receive a transaction and buffer it.
- For all outcomes, send with the transaction, commit any buffered transactions and discard any rolled back transactions.

Replica side on failover

- For all buffered transactions, provide the transactions to the Loader and the Loader attempts to commit the transactions.
- The Loader needs to be written to make each transaction is idempotent.
- If the transaction is already in the database, then the Loader performs no operation.
- If the transaction is not in the database, then the Loader applies the transaction.
- After all transactions are processed, then the new primary can begin to serve requests.

This protocol ensures that the database is at the same level as the new primary state.

Shard placement

The catalog service is responsible for placing shards. Each ObjectGrid has a number of partitions, each of which has a primary shard and an optional set of replica shards. The catalog service allocates the shards by balancing them so that they are evenly distributed over the available container servers. Replica and primary shards for the same partition are never placed on the same container server or the same IP address, unless the configuration is in development mode.

If a new container server starts, then eXtreme Scale retrieves shards from relatively overloaded container servers to the new empty container server. This movement of shards enables horizontal scaling.

Scaling out

Scaling out means that when extra container servers are added to a data grid, eXtreme Scale tries to move existing shards, primaries or replicas, from the old set of container servers to the new set. This movement expands the data grid to take advantage of the processor, network and memory of the newly added container servers. The movement also balances the data grid and tries to ensure that each JVM in the data grid hosts the same amount of data. As the data grid expands,

each server hosts a smaller subset of the total grid. eXtreme Scale assumes that data is distributed evenly among the partitions. This expansion enables scaling out.

Scaling in

Scaling in means that if a JVM fails, then eXtreme Scale tries to repair the damage. If the failed JVM had a replica, then eXtreme Scale replaces the lost replica by creating a new replica on a surviving JVM. If the failed JVM had a primary, then eXtreme Scale finds the best replica on the survivors and promotes the replica to be the new primary. eXtreme Scale then replaces the promoted replica with a new replica that is created on the remaining servers. To maintain scalability, eXtreme Scale preserves the replica count for partitions as servers fail.

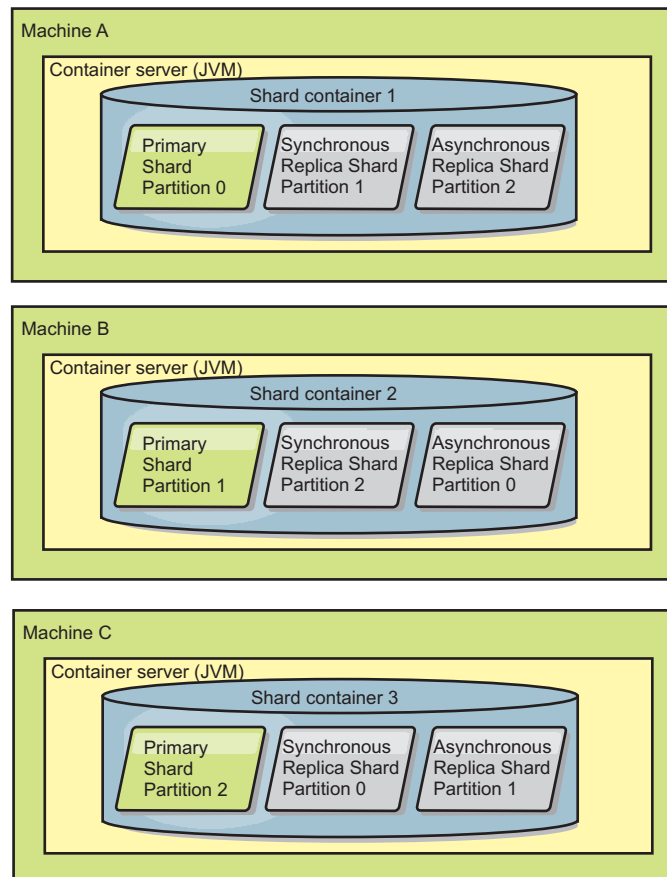


Figure 36. Placement of an ObjectGrid map set with a deployment policy of 3 partitions with a `minSyncReplicas` value of 1, a `maxSyncReplicas` value of 1, and a `maxAsyncReplicas` value of 1

Reading from replicas

You can configure map sets such that a client is permitted to read from a replica rather than being restricted to primary shards only.

It can often be advantageous to allow replicas to serve as more than simply potential primaries in the case of failures. For example, map sets can be configured to allow read operations to be routed to replicas by setting the `replicaReadEnabled` option on the MapSet to true. The default setting is false.

For more information on the MapSet element, see the topic on the deployment policy descriptor XML file in the *Administration Guide*.

Enabling reading of replicas can improve performance by spreading read requests to more Java™ virtual machines. If the option is not enabled, all read requests such as the `ObjectMap.get` or the `Query.getResultIterator` methods are routed to the primary. When `replicaReadEnabled` is set to true, some get requests might return stale data, so an application using this option must be able to tolerate this possibility. However, a cache miss will not occur. If the data is not on the replica, the get request is redirected to the primary and tried again.

The `replicaReadEnabled` option can be used with both synchronous and asynchronous replication.

Load balancing across replicas

Load balancing across replicas is typically used only when clients are caching data that is changing all the time or when the clients are using pessimistic locking.

The eXtreme Scale, unless configured otherwise, sends all read and write requests to the primary server for a given replication group. The primary must service all requests from clients. You might want to allow read requests to be sent to replicas of the primary. Sending read requests to the replicas allows the load of the read requests to be shared by multiple Java Virtual Machines (JVM). However, using replicas for read requests can result in inconsistent responses.

Load balancing across replicas is typically used only when clients are caching data that is changing all the time or when the clients are using pessimistic locking.

If the data is continually changing and then being invalidated in client near caches, the primary should see a relatively high get request rate from clients as a result. Likewise, in pessimistic locking mode, no local cache exists, so all requests are sent to the primary.

If the data is relatively static or if pessimistic mode is not used, then sending read requests to the replica does not have a big impact on performance. The frequency of get requests from clients with caches that are full of data is not high.

When a client first starts, its near cache is empty. Cache requests to the empty cache are forwarded to the primary. The client cache gets data over time, causing the request load to drop. If a large number of clients start concurrently, then the load might be significant and replica read might be an appropriate performance choice.

Shard lifecycles

Shards go through different states and events to support replication. The lifecycle of a shard includes coming online, run time, shut down, fail over and error handling. Shards can be promoted from a replica shard to a primary shard to handle server state changes.

Lifecycle events

When primary and replica shards are placed and started, they go through a series of events to bring themselves online and into listening mode.

Primary shard

The catalog service places a primary shard for a partition. The catalog service also does the work of balancing primary shard locations and initiating failover for primary shards.

When a shard becomes a primary shard, it receives a list of replicas from the catalog service. The new primary shard creates a replica group and registers all the replicas.

When the primary is ready, an open for business message displays in the SystemOut.log file for the container on which it is running. The open message, or the CWOBJ1511I message, lists the map name, map set name, and partition number of the primary shard that started.

```
CWOBJ1511I: mapName:mapSetName:partitionNumber (primary) is open for business.
```

See “Shard placement” on page 108 for more information on how the catalog service places shards.

Replica shard

Replica shards are mainly controlled by the primary shard unless the replica shard detects a problem. During a normal lifecycle, the primary shard places, registers, and de-registers a replica shard.

When the primary shard initializes a replica shard, a message displays the log that describes where the replica runs to indicate that the replica shard is available. The open message, or the CWOBJ1511I message, lists the map name, map set name, and partition number of the replica shard. This message follows:

```
CWOBJ1511I: mapName:mapSetName:partitionNumber (synchronous replica) is open for business.
```

or

```
CWOBJ1511I: mapName:mapSetName:partitionNumber (asynchronous replica) is open for business.
```

Asynchronous replica shard: An asynchronous replica shard polls the primary for data. The replica automatically will adjust the poll timing if it does not receive data from the primary, which indicates that it is caught up with the primary. It also will adjust if it receives an error that might indicate that the primary has failed, or if there is a network problem.

When the asynchronous replica starts replicating, it prints the following message to the SystemOut.log file for the replica. This message might print more than one time per CWOBJ1511 message. It will print again if the replica connects to a different primary or if template maps are added.

```
CWOBJ1543I: The asynchronous replica objectGridName:mapSetName:partitionNumber started or continued replicating from the primary. Replicating for maps: [mapName]
```

Synchronous replica shard: When the synchronous replica shard first starts, it is not yet in peer mode. When a replica shard is in peer mode, it receives data from the primary as data comes into the primary. Before entering peer mode, the replica shard needs a copy of all of the existing data on the primary shard.

The synchronous replica copies data from the primary shard similar to an asynchronous replica by polling for data. When it copies the existing data from the primary, it switches to peer mode and begins to receive data as the primary receives the data.

When a replica shard reaches peer mode, it prints a message to the SystemOut.log file for the replica. The time refers to the amount of time that it took the replica shard to get all of its initial data from the primary shard. The time might display as zero or very low if the primary shard does not have any existing data to

replicate. This message may print more than one time per CWOBJ1511 message. It will print again if the replica connects to a different primary or if template maps are added.

CWOB1526I: Replica objectGridName:mapsetName:partitionNumber:mapName entering peer mode after X seconds.

When the synchronous replica shard is in peer mode, the primary shard must replicate transactions to all peer mode synchronous replicas. The synchronous replica shard data remains at the same level as the primary shard data. If a minimum number of synchronous replicas or minSync is set in the deployment policy, that number of synchronous replicas must vote to commit before the transaction can successfully commit on the primary.

Recovery events

Replication is designed to recover from failure and error events. If a primary shard fails, another replica takes over. If errors are on the replica shards, the replica shard attempts to recover. The catalog service controls the placement and transactions of new primary shards or new replica shards.

Replica shard becomes a primary shard

A replica shard becomes a primary shard for two reasons. Either the primary shard stopped or failed, or a balance decision was made to move the previous primary shard to a new location.

The catalog service selects a new primary shard from the existing synchronous replica shards. If a primary move needs to take place and there are no replicas, a temporary replica will be placed to complete the transition. The new primary shard registers all of the existing replicas and accepts transactions as the new primary shard. If the existing replica shards have the correct level of data, the current data is preserved as the replica shards register with the new primary shard. Asynchronous replicas will poll against the new primary.

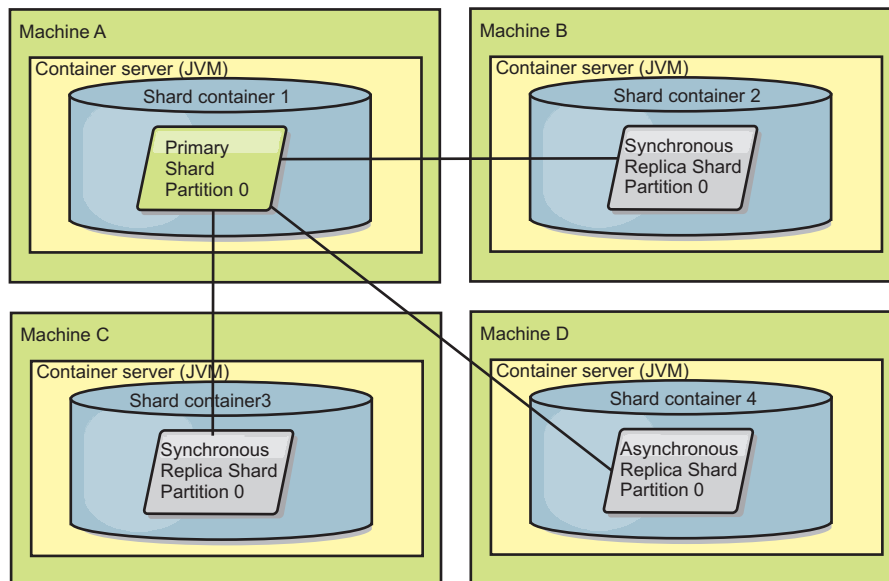


Figure 37. Example placement of an ObjectGrid map set for the partition0 partition. The deployment policy has a minSyncReplicas value of 1, a maxSyncReplicas value of 2, and a maxAsyncReplicas value of 1.

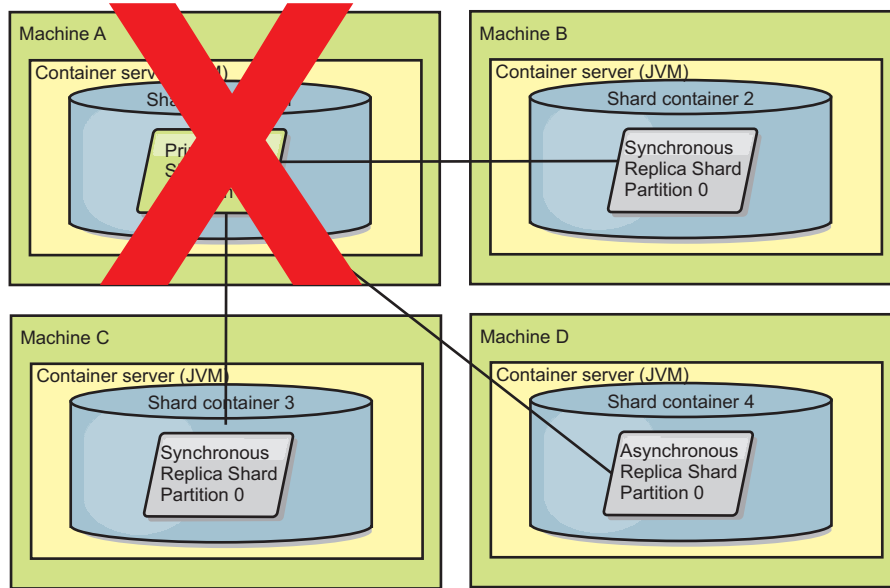


Figure 38. The container for the primary shard fails

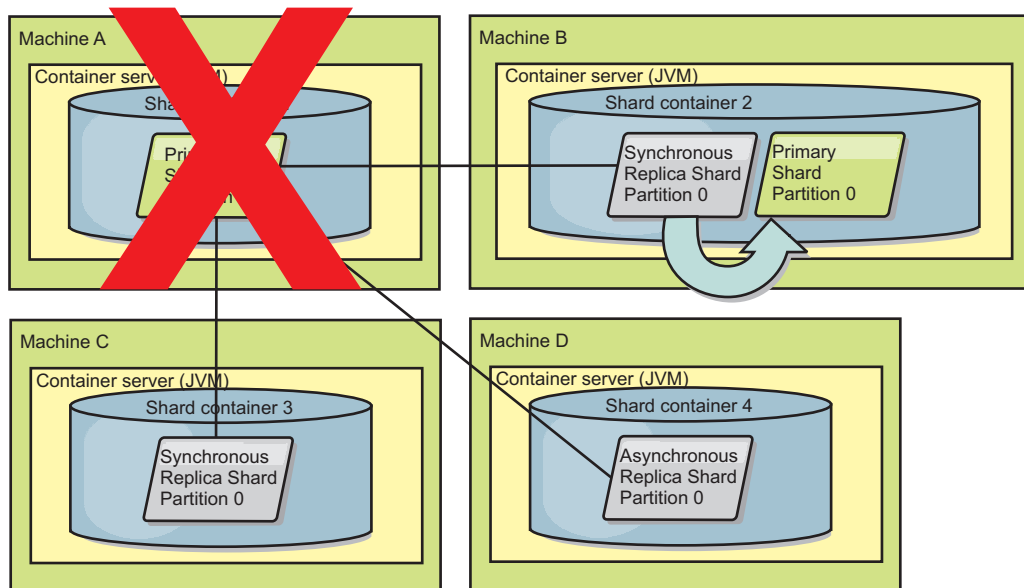


Figure 39. The synchronous replica shard on ObjectGrid container 2 becomes the primary shard

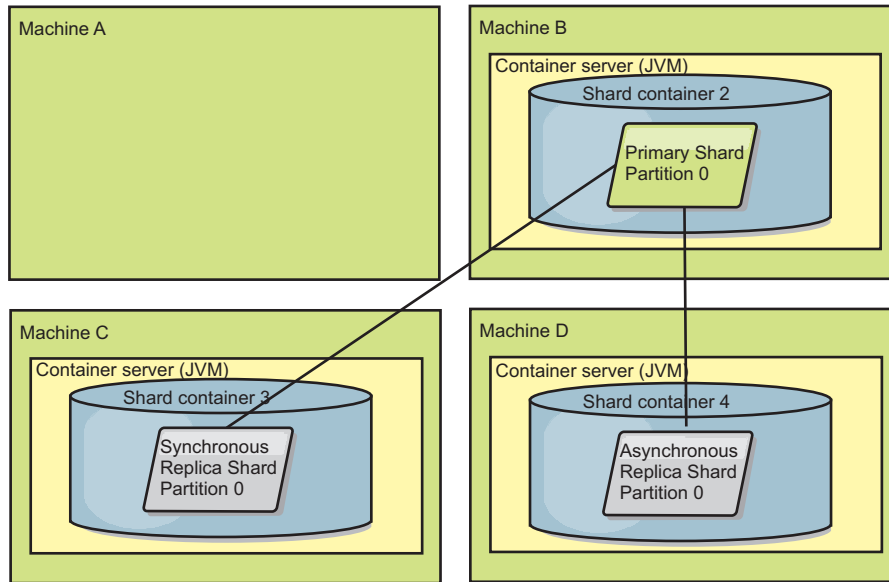


Figure 40. Machine B contains the primary shard. Depending on how automatic repair mode is set and the availability of the containers, a new synchronous replica shard might or might not be placed on a machine.

Replica shard recovery

A synchronous replica shard is controlled by the primary shard. However, if a replica shard detects a problem, it can trigger a reregister event to correct the state of the data. The replica clears the current data and gets a fresh copy from the primary.

When a replica shard initiates a reregister event, the replica prints a log message.

```
CW0BJ1524I: Replica listener
objectGridName:mapSetName:partition must re-register with the primary.
Reason: Exception listed
```

If a transaction causes an error on a replica shard during processing, then the replica shard is in an unknown state. The transaction successfully processed on the primary shard, but something went wrong on the replica. To correct this situation, the replica initiates a reregister event. With a new copy of data from the primary, the replica shard can continue. If the same problem reoccurs, the replica shard does not continuously reregister. See “Failure events” for more details.

Failure events

A replica can stop replicating data if it encounters error situations for which the replica cannot recover.

Too many register attempts

If a replica triggers a reregister multiple times without successfully committing data, the replica stops. Stopping prevents a replica from entering an endless reregister loop. By default, a replica shard tries to reregister three times in a row before stopping.

If a replica shard reregisters too many times, it prints the following message to the log.

```
CW0BJ1537E: objectGridName:mapSetName:partition exceeded the maximum number
of times to reregister (timesAllowed) without successful transactions..
```

If the replica is unable to recover by reregistering, a pervasive problem might exist with the transactions that are relative to the replica shard. A possible problem could be missing resources on the classpath if an error occurs while inflating the keys or values from the transaction.

Failure while entering peer mode

If a replica attempts to enter peer mode and experiences an error processing the bulk existing data from the primary (the checkpoint data), the replica shuts down. Shutting down prevents a replica from starting with incorrect initial data. Because it receives the same data from the primary if it reregisters, the replica does not retry.

If a replica shard fails to enter peer mode, it prints the following message to the log:

```
CW0BJ1527W Replica objectGridName:mapSetName:partition:mapName failed to enter peer mode after numSeconds seconds.
```

An additional message displays in the log that explains why the replica failed to enter peer mode.

Recovery after re-register or peer mode failure

If a replica fails to re-register or enter peer mode, the replica is in an inactive state until a new placement event occurs. A new placement event might be a new server starting or stopping. You can also start a placement event by using the `triggerPlacement` method on the `PlacementServiceMBean` Mbean.

Map sets for replication

Replication is enabled by associating `BackingMaps` with a map set.

A map set is a collection of maps that are categorized by partition-key. This partition-key is derived from the individual map's key by taking its hash modulo the number of partitions. If one group of maps within the map set has partition-key X, those maps will be stored in a corresponding partition X in the data grid. If another group has partition-key Y, all of the maps will be stored in partition Y, and so on. Also, the data within the maps is replicated based on the policy defined on the map set, which is only used for distributed eXtreme Scale topologies (unnecessary for local instances).

See “Partitioning” on page 81 for more details.

Map sets are assigned what number of partitions they will have and a replication policy. The map set replication configuration simply identifies the number of synchronous and asynchronous replica shards a map set should have in addition to the primary shard. For example, if there is to be 1 synchronous and 1 asynchronous replica, all of the `BackingMaps` assigned to the map set will each have a replica shard distributed automatically within the set of available containers for the eXtreme Scale. The replication configuration can also enable clients to read data from synchronously replicated servers. This can spread the load for read requests over additional servers in the eXtreme Scale. Replication only has a programming model impact when preloading the `BackingMaps`.

Transaction processing overview

WebSphere eXtreme Scale uses transactions as its mechanism for interaction with data.

Java

Transaction processing in Java applications

To interact with data, the thread in your application needs its own session. When the application wants to use the ObjectGrid on a thread, call one of the ObjectGrid.getSession methods to obtain a session. With the session, the application can work with data that is stored in the ObjectGrid maps.

When an application uses a Session object, the session must be in the context of a transaction. A transaction begins and commits or begins and rolls back with the begin, commit, and rollback methods on the Session object. Applications can also work in auto-commit mode, in which the Session automatically begins and commits a transaction whenever an operation runs on the map. Auto-commit mode cannot group multiple operations into a single transaction. Auto-commit mode is the slower option if you are creating a batch of multiple operations into a single transaction. However, for transactions that contain only one operation, auto-commit is the faster option.

When your application is finished with the Session, use the optional Session.close() method to close the session. Closing the Session releases it from the heap and allows subsequent calls to the getSession() method to be reused, improving performance.

.NET

8.6+

Transaction processing in .NET applications

To interact with data, each thread in your application needs its own transaction object. To use the IGrid interface on a thread in your application, call one of the following methods:

- IGrid.GetGridMapPessimisticAutoTx
- IGrid.GetGridMapPessimisticTx

When you call these methods, you obtain an IGridMap object that has a unique transaction object. With the IGridMap object, the application can work with data that is stored in the IGrid maps. When an application uses an IGridMapPessimisticTx object, the data grid operations must be in the context of a transaction. A transaction begins and commits or begins and rolls back the transaction with the begin, commit, and rollback methods on the IGridTransaction object. Applications can also work in auto-commit mode, in which the IGridMapPessimisticAutoTx automatically begins and commits a transaction whenever an operation runs on the map. Auto-commit mode cannot group multiple operations into a single transaction. Auto-commit mode is the slower option if you are creating a batch of multiple operations into a single transaction. However, for transactions that contain only one operation, auto-commit is the faster option.

When your application is finished with the IGridMap instance, dispose the IGridMap object. Disposing the object closes the associated transaction object. As a result, subsequent calls to the GetGridMapPessimisticAutoTx and

GetGridMapPessimisticTx methods can reuse an existing, free transaction object, which improves performance.

Transactions

Transactions have many advantages for data storage and manipulation. You can use transactions to protect the data grid from concurrent changes, to apply multiple changes as a concurrent unit, to replicate data, and to implement a lifecycle for locks on changes.

When a transaction starts, WebSphere eXtreme Scale allocates a special difference map to hold the current changes or copies of key and value pairs that the transaction uses. Typically, when a key and value pair is accessed, the value is copied before the application receives the value. In Java applications, the difference map tracks all changes for operations such as insert, update, get, and remove. In .NET applications, the difference map tracks changes in add, replace, get, and remove operations. Keys are not copied because they are assumed to be immutable. If a transaction is rolled back, then the difference map information is discarded, and locks on entries are released. When a transaction commits, the changes are applied to the maps and locks are released.

Java If an ObjectTransformer object is specified in a Java application, then this object is used for copying the value. If the transaction is using optimistic locking, then before images of the values are also tracked for comparison when the transaction commits.

Java If optimistic locking is being used in a Java application, then eXtreme Scale compares the before image versions of the values with the values that are in the map. These values must match for the transaction to commit. This comparison enables a multiple version locking scheme, but at a cost of two copies being made when the transaction accesses the entry. All values are copied again and the new copy is stored in the map. WebSphere eXtreme Scale performs this copy to protect itself against the application changing the application reference to the value after a commit.

You can avoid using several copies of the information. The application can save a copy by using pessimistic locking instead of optimistic locking as the cost of limiting concurrency. The copy of the value at commit time can also be avoided if the application agrees not to change a value after a commit.

Note: **.NET** **8.6+** .NET applications support pessimistic locking only.

Advantages of transactions

Use transactions for the following reasons:

By using transactions, you can:

- Roll back changes if an exception occurs or business logic needs to undo state changes.
- To apply multiple changes as an atomic unit at commit time.
- Hold and release locks on data to apply multiple changes as an atomic unit at commit time.
- Protect a thread from concurrent changes.
- Implement a lifecycle for locks on changes.

- Produce an atomic unit of replication.

Transaction size

Larger transactions are more efficient, especially for replication. However, larger transactions can adversely affect concurrency because the locks on entries are held for a longer time. If you use larger transactions, you can increase replication performance. This performance increase is important when you are pre-loading a Map. Experiment with different batch sizes to determine what works best for your scenario.

Larger transactions also help with loaders. If a loader is being used that can run SQL batching, then significant performance gains are possible depending on the transaction and significant load reductions on the database side. This performance gain depends on the Loader implementation.

Automatic commit mode

Java If no transaction is actively started, then when an application interacts with an ObjectMap object, an automatic begin and commit operation is done on behalf of the application. This automatic begin and commit operation works, but prevents rollback and locking from working effectively. Synchronous replication speed is impacted because of the very small transaction size. If you are using an entity manager application, then do not use automatic commit mode because objects that are looked up with the EntityManager.find method immediately become unmanaged on the method return and become unusable.

.NET **8.6+** In .NET applications, the GridMapPessimisticAutoTx map interface provides the equivalent automatic begin and commit operations. The limitations are the same: rollback and locking do not work correctly and synchronous replication speed is reduced.

Java

External transaction coordinators

Typically, transactions begin with the session.begin method and end with the session.commit method. However, when eXtreme Scale is embedded, the transactions might be started and ended by an external transaction coordinator. If you are using an external transaction coordinator, you do not need to call the session.begin method and end with the session.commit method. If you are using WebSphere Application Server, you can use the WebSphereTransactionCallback plug-in.

Java

Java EE transaction integration

eXtreme Scale includes a Java Connector Architecture (JCA) 1.5 compliant resource adapter that supports both client connections to a remote data grid and local transaction management. Java Platform, Enterprise Edition (Java EE) applications such as servlets, JavaServer Pages (JSP) files and Enterprise JavaBeans (EJB) components can demarcate eXtreme Scale transactions using the standard javax.resource.cci.LocalTransaction interface or the eXtreme Scale session interface.

When the running in WebSphere Application Server with last participant support enabled in the application, you can enlist the eXtreme Scale transaction in a global

transaction with other two-phase commit transactional resources.

Transaction processing in Java EE applications

WebSphere eXtreme Scale provides its own resource adapter that you can use to connect applications to the data grid and process local transactions.

Through support from the eXtreme Scale resource adapter, Java Platform, Enterprise Edition (Java EE) applications can look up eXtreme Scale client connections and demarcate local transactions using Java EE local transactions or using the eXtreme Scale APIs. When the resource adapter is configured, you can complete the following actions with your Java EE applications:

- Look up or inject eXtreme Scale resource adapter connection factories within a Java EE application component.
- Obtain standard connection handles to the eXtreme Scale client and share them between application components using Java EE conventions.
- Demarcate eXtreme Scale transactions using either the `javax.resource.cci.LocalTransaction` API or the `com.ibm.websphere.objectgrid.Session` interface.
- Use the entire eXtreme Scale client API, such as the `ObjectMap` API and `EntityManager` API.

The following additional capabilities are available with WebSphere Application Server:

- Enlist eXtreme Scale connections with a global transaction as a last participant with other two-phase commit resources. The eXtreme Scale resource adapter provides local transaction support, with a single-phase commit resource. With WebSphere Application Server, your applications can enlist one, single-phase commit resource into a global transaction through last participant support.
- Automatic resource adapter installation when the profile is augmented.
- Automatic security principal propagation.

Administrator responsibilities

The eXtreme Scale resource adapter is installed into the Java EE application server or embedded with the application. After you install the resource adapter, the administrator creates one or more resource adapter connection factories for each catalog service domain and optionally each data grid instance. The connection factory identifies the properties that are required to communicate with the data grid.

Applications reference the connection factory, which establishes the connection to the remote data grid. Each connection factory hosts a single eXtreme Scale client connection that is reused for all application components.

Important: Because the eXtreme Scale client connection might include a near cache, applications must not share a connection. A connection factory must exist for a single application instance to avoid problems sharing objects between applications.

The connection factory hosts an eXtreme Scale client connection, which is shared between all referencing application components. You can use a managed bean (MBean) to access information about the client connection or to reset the connection when it is no longer needed.

Application developer responsibilities

An application developer creates resource references for managed connection factories in the application deployment descriptor or with annotations. Each resource reference includes a local reference for the eXtreme Scale connection factory, as well as the resource-sharing scope.

Important: Enabling resource sharing is important because it allows the local transaction to be shared between application components.

Applications can inject the connection factory into the Java EE application component, or it can look up the connection factory using Java Naming Directory Interface (JNDI). The connection factory is used to obtain connection handles to the eXtreme Scale client connection. The eXtreme Scale client connection is managed independently from the resource adapter connection and is established on first use, and reused for all subsequent connections.

After finding the connection, the application retrieves an eXtreme Scale session reference. With the eXtreme Scale session reference, the application can use the entire eXtreme Scale client APIs and features.

You can demarcate transactions in one of the following ways:

- Use the `com.ibm.websphere.objectgrid.Session` transaction demarcation methods.
- Use the `javax.resource.cci.LocalTransaction` local transaction.
- Use a global transaction, when you use WebSphere Application Server with last participant support enabled. When you select this approach for demarcation, you must:
 - Use an application-managed global transaction with the `javax.transaction.UserTransaction`.
 - Use a container-managed transaction.

Application deployer responsibilities

The application deployer binds the local reference to the resource adapter connection factory that the application developer defines to the resource adapter connection factories that the administrator defines. The application deployer must assign the correct connection factory type and scope to the application and ensure that the connection factory is not shared between applications to avoid Java object sharing. The application deployer is also responsible for configuring and mapping other appropriate configuration information that is common to all connection factories.

CopyMode attribute

You can tune the number of copies by defining the `CopyMode` attribute of the `BackingMap` or `ObjectMap` objects in the `ObjectGrid` descriptor XML file.

For Java applications, you can tune the number of copies by defining the `CopyMode` attribute of the `BackingMap` or `ObjectMap` objects.

.NET **8.6+** For .NET applications, only `COPY_TO_BYTES` mode is supported.

The copy mode has the following values:

- COPY_ON_READ_AND_COMMIT
- COPY_ON_READ
- NO_COPY
- COPY_ON_WRITE
- COPY_TO_BYTES
- COPY_TO_BYTES_RAW

The COPY_ON_READ_AND_COMMIT value is the default. The COPY_ON_READ value copies the initial data when it is retrieved, but does not copy at commit time. This mode is safe if the application does not modify a value after committing a transaction. The NO_COPY value does not copy data, which is only safe for read-only data. If the data never changes, then you do not need to copy it for isolation reasons.

Be careful when you use the NO_COPY attribute value with maps that can be updated. WebSphere eXtreme Scale uses the copy on first touch to allow the transaction rollback. The application only changed the copy, and as a result, eXtreme Scale discards the copy. If the NO_COPY attribute value is used, and the application modifies the committed value, completing a rollback is not possible. Modifying the committed value leads to problems with indexes, replication, and so on because the indexes and replicas update when the transaction commits. If you modify committed data and then roll back the transaction, which does not actually roll back at all, then the indexes are not updated and replication does not take place. Other threads can see the uncommitted changes immediately, even if they have locks. Use the NO_COPY attribute value for read-only maps or for applications that complete the appropriate copy before modifying the value. If you use the NO_COPY attribute value and call IBM support with a data integrity problem, you are asked to reproduce the problem with the copy mode set to COPY_ON_READ_AND_COMMIT.

The COPY_TO_BYTES value stores values in the map in a serialized form. At read time, eXtreme Scale inflates the value from a serialized form and at commit time it stores the value to a serialized form. With this method, a copy occurs at both read and commit time.

Restriction: 8.6+

When you use optimistic locking with COPY_TO_BYTES, you might experience ClassNotFoundException exceptions during common operations, such as invalidating cache entries. These exceptions occur because the optimistic locking mechanism must call the equals() method of the cache object to detect any changes before the transaction is committed. To call the equals() method, the eXtreme Scale server must be able to deserialize the cached object, which means that eXtreme Scale must load the object class.

To resolve these exceptions, you can package the cached object classes so that the eXtreme Scale server can load the classes in stand-alone environments. Therefore, you must put the classes in the class path.

If your environment includes the OSGi framework, then package the classes into a fragment of the objectgrid.jar bundle. If you are running eXtreme Scale servers in the Liberty profile, package the classes as an OSGi bundle, and export the Java packages for those classes. Then, install the bundle by copying it into the grids directory.

In WebSphere Application Server, package the classes in the application or in a shared library that the application can access.

Alternatively, you can use custom serializers that can compare the byte arrays that are stored in eXtreme Scale to detect any changes.

The default copy mode for a map can be configured on the BackingMap object. You can also change the copy mode on maps before you start a transaction by using the ObjectMap.setCopyMode method.

An example of a backing map snippet from an objectgrid.xml file that shows how to set the copy mode for a backing map follows. This example assumes that you are using cc as the objectgrid/config namespace.

```
<cc:backingMap name="RuntimeLifespan" copyMode="NO_COPY"/>
```

Locking strategies

Locking strategies include pessimistic, optimistic, and none. To choose a locking strategy, you must consider issues such as the percentage of each type of operations you have, whether you use a loader, and so on.

Locks are bound by transactions. You can specify the following locking settings:

Java No locking

Running without the locking setting is the fastest. If you are using read-only data, then you might not need locking.

Restriction: 8.6+ BackingMaps configured to use a no locking strategy cannot participate in a multi-partition transaction.

Java .NET Pessimistic locking

Acquires locks on entries, then and holds the locks until commit time. This locking strategy provides good consistency at the expense of throughput.

Java Optimistic locking

Takes a before image of every record that the transaction touches and compares the image to the current entry values when the transaction commits. If the entry values change, then the transaction rolls back. No locks are held until commit time. This locking strategy provides better concurrency than the pessimistic strategy, at the risk of the transaction rolling back and the memory cost of making the extra copy of the entry.

Important: 8.6+ If you are using a client application with WebSphere eXtreme Scale Client for .NET, only pessimistic locking is supported.

Lock manager

When either a PESSIMISTIC or an OPTIMISTIC locking strategy is used, a lock manager is created for the BackingMap. The lock manager uses a hash map to track entries that are locked by one or more transactions. If many map entries exist in the hash map, more lock buckets can result in better performance. The risk of Java synchronization collisions is lower as the number of buckets grows. More lock buckets also lead to more concurrency. The previous examples show how an application can set the number of lock buckets to use for a given BackingMap instance.

Pessimistic locking

The PESSIMISTIC lock strategy acquires locks for cache entries and should be used when data is changed frequently. Any time a cache entry is read, a lock is acquired and conditionally held until the transaction completes. The duration of some locks can be tuned using transaction isolation levels for the session.

Use the pessimistic locking strategy for read and write maps when other locking strategies are not possible. When an ObjectGrid map is configured to use the pessimistic locking strategy, a pessimistic transaction lock for a map entry is obtained when a transaction first gets the entry from the BackingMap. The pessimistic lock is held until the application completes the transaction. Typically, the pessimistic locking strategy is used in the following situations:

- When the BackingMap is configured with or without a loader and versioning information is not available.

Restriction: 8.6+ BackingMaps that are configured with a Loader plug-in can read but cannot write to the map in a multi-partition transaction.

- When the BackingMap is used directly by an application that needs help from the eXtreme Scale for concurrency control.
- When versioning information is available, but update transactions frequently collide on the backing entries, resulting in optimistic update failures.

The pessimistic locking strategy has the greatest impact on performance and scalability. Therefore, use this strategy only for read and write maps when other locking strategies are not viable. For example, these situations might include when optimistic update failures occur frequently, or when recovery from optimistic failure is difficult for an application to handle.

When you use pessimistic locking, you can use lock methods to lock data, or keys, without returning any data values. For a list of the methods and what kind of locks they acquire, see “Lock types” on page 126.

Optimistic locking

The default lock strategy is OPTIMISTIC. Use optimistic locking when data is changed infrequently. Locks are only held for a short duration while data is being read from the cache and copied to the transaction. When the transaction cache is synchronized with the main cache, any cache objects that have been updated are checked against the original version. If the check fails, then the transaction is rolled back and an OptimisticCollisionException exception results.

The optimistic locking strategy assumes that no two transactions might attempt to update the same map entry while the transactions are running concurrently. The lock is not held for the lifecycle of the transaction because it is unlikely that more than one transaction might update the map entry concurrently. The optimistic locking strategy is typically used in the following situations:

- When a BackingMap is configured and versioning information is available. The BackingMap can be configured with or without a loader.

Restriction: 8.6+ BackingMaps that are configured with a Loader plug-in can read but cannot write to the map in a multi-partition transaction.

- When a BackingMap has mostly transactions that are read operations. Insert, update, or remove operations on map entries do not occur often on the BackingMap.
- When a BackingMap is inserted, updated, or removed more frequently than it is read, but transactions rarely collide on the same map entry.

Like the pessimistic locking strategy, the methods on the ObjectMap interface determine how eXtreme Scale automatically attempts to acquire a lock mode for the map entry that is being accessed. However, the following differences between the pessimistic and optimistic strategies exist:

- Like the pessimistic locking strategy, an S lock mode is acquired by the get and getAll methods when the method is called. However, with optimistic locking, the S lock mode is not held until the transaction is completed. Instead, the S lock mode is released before the method returns to the application. The purpose of acquiring the lock mode is so that eXtreme Scale can ensure that only committed data from other transactions is visible to the current transaction. After eXtreme Scale has verified that the data is committed, the S lock mode is released. At commit time, an optimistic versioning check is performed to ensure that no other transaction has changed the map entry after the current transaction released its S lock mode. If an entry is not fetched from the map before it is updated, invalidated, or deleted, the eXtreme Scale run time implicitly fetches the entry from the map. This implicit get operation is performed to get the current value at the time the entry was requested to be modified.
- Unlike pessimistic locking strategy, the getForUpdate and getAllForUpdate methods are handled exactly like the get and getAll methods when the optimistic locking strategy is used. That is, an S lock mode is acquired at the start of the method and the S lock mode is released before returning to the application.

All other ObjectMap methods are handled the same as the pessimistic locking strategy. When the commit method is called, an X lock mode is obtained for any map entry that is inserted, updated, removed, touched, or invalidated. The X lock mode is held until the transaction completes commit processing.

The optimistic locking strategy assumes that no concurrently running transactions attempt to update the same map entry. Because of this assumption, the lock mode does not need to be held for the life of the transaction because it is unlikely that more than one transaction might update the map entry concurrently. However, because a lock mode was not held, another concurrent transaction might potentially update the map entry after the current transaction has released its S lock mode.

To handle this possibility, eXtreme Scale gets an X lock at commit time and performs an optimistic versioning check to verify that no other transaction has changed the map entry after the current transaction read the map entry from the BackingMap. If another transaction changes the map entry, the version check fails and an OptimisticCollisionException exception occurs. This exception forces the current transaction to be rolled back and the application must try the entire transaction again. The optimistic locking strategy is useful when a map is mostly read and it is unlikely that updates for the same map entry might occur.

Restriction: 8.6+

When you use optimistic locking with COPY_TO_BYTES, you might experience ClassNotFoundException exceptions during common operations, such as

invalidating cache entries. These exceptions occur because the optimistic locking mechanism must call the equals() method of the cache object to detect any changes before the transaction is committed. To call the equals() method, the container server must be able to deserialize the cached object, which means that eXtreme Scale must load the object class.

To resolve these exceptions, you can package the cached object classes so that the container server can load the classes in stand-alone environments. Therefore, you must put the classes in the class path.

If your environment includes the OSGi framework, then package the classes into a fragment of the objectgrid.jar bundle. If you are running eXtreme Scale servers in the Liberty profile, package the classes as an OSGi bundle, and export the Java packages for those classes. Then, install the bundle by copying it into the grids directory.

In WebSphere Application Server, package the classes in the application or in a shared library that the application can access.

Alternatively, you can use custom serializers that can compare the byte arrays that are stored in eXtreme Scale to detect any changes.

Java

No locking

If locking is not required because the data is never updated or is only updated during quiet periods, you can disable locking by using the NONE lock strategy. 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.

When a BackingMap is configured to use no locking strategy, no transaction locks for a map entry are obtained.

Restriction: 8.6+ BackingMaps configured to use a no locking strategy cannot participate in a multi-partition transaction.

Using no locking strategy is useful when an application is a persistence manager such as an Enterprise JavaBeans (EJB) container or when an application uses Hibernate to obtain persistent data. In this scenario, the BackingMap is configured without a loader and the persistence manager uses the BackingMap as a data cache. In this scenario, the persistence manager provides concurrency control between transactions that are accessing the same Map entries.

WebSphere eXtreme Scale does not need to obtain any transaction locks for concurrency control. This situation assumes that the persistence manager does not release its transaction locks before updating the ObjectGrid map with committed changes. If the persistence manager releases its locks, then a pessimistic or optimistic lock strategy must be used. For example, suppose that the persistence manager of an EJB container is updating an ObjectGrid map with data that was committed in the EJB container-managed transaction. If the update of the ObjectGrid map occurs before the persistence manager transaction locks are released, then you can use the no lock strategy. If the ObjectGrid map update occurs after the persistence manager transaction locks are released, then you must use either the optimistic or pessimistic lock strategy.

Another scenario where no locking strategy can be used is when the application uses a BackingMap directly and a Loader is configured for the map. In this scenario, the loader uses the concurrency control support that is provided by a relational database management system (RDBMS) by using either Java database connectivity (JDBC) or Hibernate to access data in a relational database. The loader implementation can use either an optimistic or pessimistic approach. A loader that uses an optimistic locking or versioning approach helps to achieve the greatest amount of concurrency and performance. For more information about implementing an optimistic locking approach, see the OptimisticCallback section in Configuring database loaders. If you are using a loader that uses pessimistic locking support of an underlying backend, you might want to use the forUpdate parameter that is passed on the get method of the Loader interface. Set this parameter to true if the getForUpdate method of the ObjectMap interface was used by the application to get the data. The loader can use this parameter to determine whether to request an upgradeable lock on the row that is being read. For example, DB2[®] obtains an upgradeable lock when an SQL select statement contains a FOR UPDATE clause. This approach offers the same deadlock prevention that is described in “Pessimistic locking” on page 123.

Lock types

When you are using pessimistic and optimistic locking, shared (S), upgradeable (U) and exclusive (X) locks are used to maintain consistency. Understanding locking and its behavior is important when you have pessimistic locking enabled. With optimistic locking, the locks are not held. Different types of locks are compatible with others in various ways. Locks must be handled in the correct order to avoid deadlock scenarios.

Shared, upgradeable, and exclusive locks

When an application calls any method of the map programming interface, WebSphere eXtreme Scale automatically attempts to acquire a lock for the map entry that is being accessed.

Java In Java applications, locks are also acquired when the applications uses the find methods on an index, or does a query.

8.6+ When you are using pessimistic locking, you can use the lock methods to lock keys without returning any data values. With the lock methods, you can lock the key in the data grid or lock the key and determine whether the value exists in the data grid.

LockMode is an enum with possible values where you can specify the keys that you want to lock:

- **Java** SHARED, UPGRADABLE, and EXCLUSIVE
- **.NET** **8.6+** Shared, Upgradable, Exclusive

WebSphere eXtreme Scale uses the following lock modes that are based on the method the application calls in the map programming interface.

S lock A shared lock mode for the key of a map entry. The duration that the S lock is held depends on the transaction isolation level used. An S lock mode allows concurrency between transactions that attempt to acquire an S or an upgradeable lock (U lock) mode for the same key, but blocks other transactions that attempt to get an exclusive lock (X lock) mode for the same key.

U lock

An upgradeable lock mode for the key of a map entry. The U lock is held until the transaction completes. A U lock mode allows concurrency between transactions that acquire an S lock mode for the same key, but blocks other transactions that attempt to acquire a U lock or X lock mode for the same key.

X lock Exclusive lock mode for the key of a map entry. The X lock is held until the transaction completes. An X lock mode ensures that only one transaction is inserting, updating, or removing a map entry of a given key value. An X lock blocks all other transactions that attempt to acquire an S, U, or X lock mode for the same key.

An S lock mode is weaker than a U lock mode because it allows more transactions to run concurrently when they are accessing the same map entry. The U lock mode is slightly stronger than the S lock mode because it blocks other transactions that are requesting either a U or X lock mode. The S lock mode only blocks other transactions that are requesting an X lock mode. This small difference is important in preventing some deadlocks from occurring. The X lock mode is the strongest lock mode because it blocks all other transactions that are attempting to get an S, U, or X lock mode for the same map entry. The X lock mode ensures that only one transaction can insert, update, or remove a map entry and to prevent updates from being lost when more than one transaction is attempting to update the same map entry.

See the following table to understand the relationship between these lock mode values and the behavior of existing methods:**8.6+**

Table 6. LockMode values and existing method equivalents

Lock mode (Java / .NET)	Java method equivalent	.NET method equivalent
SHARED / Shared	get and getAll methods on the ObjectMap interface, index methods, and queries	Get(), GetAndLock(Key,LockMode.Shared), GetAndLockAll(KeyList, LockMode.Shared),GetAll methods
UPGRADABLE /Upgradable	getForUpdate(), getAllForUpdate()	GetAndLock(Key, LockMode.Updgradable), GetAndLockAll(KeyList, LockMode.Upgradable)

Table 6. LockMode values and existing method equivalents (continued)

Lock mode (Java / .NET)	Java method equivalent	.NET method equivalent
EXCLUSIVE / Exclusive	<p>getNextKey(), commit(), put, putAll, remove, removeAll, insert, update, and touch methods, global invalidate and global invalidateAll methods. (No locks are acquired for the local invalidate and invalidateAll methods because none of the BackingMap entries are invalidated by local invalidate method calls.)</p> <p>Note:  8.6+ The upsert and upsertAll methods replace the ObjectMap put and putAll methods. Use the upsert method to tell the BackingMap and loader that an entry in the data grid must place the key and value into the grid. The BackingMap and loader either inserts or an updates to place the value into the data grid and loader. If you run the upsert API within your applications, then the loader gets an UPSERT LogElement type, which allows loaders to do database merge or upsert calls instead of using insert or update.</p>	<p>Commit(), Add, AddAll, Put, PutAll, Remove, RemoveAll, Replace, ReplaceAll, Touch, TouchAll, Invalidate, InvalidateAll</p> <p>Note: The Put and PutAll methods are equivalent to the Java upsert and upsertAll methods.</p>

The following table is a lock mode compatibility matrix that summarizes the described lock modes, which you can use to determine which lock modes are compatible with each other. To read this matrix, the row in the matrix indicates a lock mode that is already granted. The column indicates the lock mode that is requested by another transaction. If Yes is displayed in the column, then the lock mode that is requested by the other transaction is granted because it is compatible with the lock mode that is already granted. No indicates that the lock mode is not compatible and the other transaction must wait for the first transaction to release the lock that it owns.

Table 7. Lock mode compatibility matrix

Lock	Lock type S (shared)	Lock type U (upgradeable)	Lock type X (exclusive)	Strength
S (shared)	Yes	Yes	No	weakest
U (upgradeable)	Yes	No	No	normal
X (exclusive)	No	No	No	strongest

Deadlocks

Deadlocks can occur when two transactions try to update the same cache entry.

Classic deadlock example

Consider the following sequence of lock mode requests:

1. X lock is granted to transaction 1 for key1.
2. X lock is granted to transaction 2 for key2.
3. X lock requested by transaction 1 for key2. (Transaction 1 is blocked and is waiting for the lock that is owned by transaction 2.)
4. X lock requested by transaction 2 for key1. (Transaction 2 is blocked and is waiting for the lock that is owned by transaction 1.)

The preceding sequence is the classic deadlock example of two transactions that attempt to acquire more than a single lock, and each transaction acquires the locks in a different order. To prevent this deadlock, each transaction must obtain the multiple locks in the same order.

Java

Deadlock prevention with optimistic locking

If the OPTIMISTIC lock strategy is used and the flush method on the ObjectMap interface is never used by the application, then lock modes are requested by the transaction only during the commit cycle. During the commit cycle, eXtreme Scale uses deterministic behavior. The keys for map entries that must be locked are determined. Then, the lock modes are requested in key sequence. With this behavior, eXtreme Scale prevents most of the classic deadlocks.

However, eXtreme Scale does not and cannot prevent all possible deadlock scenarios. A few scenarios exist that the application must consider. Following are the scenarios that the application must be aware of and take preventive action against.

One scenario exists where eXtreme Scale is able to detect a deadlock without having to wait for a lock wait timeout to occur. If this scenario does occur, a `com.ibm.websphere.objectgrid.LockDeadlockException` results. Consider the following code example:

```
Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
sess.begin();
Person p = (IPerson)person.get("Lynn");
// Lynn had a birthday; so make her 1 year older.
p.setAge( p.getAge() + 1 );
person.put( "Lynn", p );
sess.commit();
```

8.6+ In the same scenario, you can use the `upsert` method in the code example:

```
Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
sess.begin();
Person p = (IPerson)person.get("Lynn");
// Lynn had a birthday; so make her 1 year older.
p.setAge( p.getAge() + 1 );
person.upsert( "Lynn", p );
sess.commit();
```

In this situation, two transactions attempt to update the age of the Lynn person object. In this situation, both transactions own an S lock mode on the Lynn entry of the PERSON map as a result of the `person.get("Lynn")` method invocation. As a result of the `person.put("Lynn", p)` method call, both transactions attempt to

upgrade the S lock mode to an X lock mode. Both transactions are blocked and waiting for the other transaction to release the S lock mode it owns. As a result, a deadlock occurs because a circular wait state exists between the two transactions. A circular wait state results when more than one transaction attempts to promote a lock from a weaker to a stronger mode for the same map entry. In this scenario, a `LockDeadlockException` exception results instead of a `LockTimeoutException` exception.

In Java applications, the application can prevent the `LockDeadlockException` exception for the preceding example by using the optimistic lock strategy instead of the pessimistic lock strategy. Using the optimistic lock strategy is the preferred solution when the map is mostly read and updates to the map are infrequent.

Java

.NET

Deadlock prevention with pessimistic locking

Attention: `.NET` **8.6+** .NET applications support pessimistic locking only. In the following section, the Java method names are discussed. However, the .NET method names also apply. These methods include: `Get`, `GetAndLock`, `GetAndLockAll`, `Put`, `Add`, `Replace`, and `Remove`.

To prevent deadlocks when you are using the pessimistic locking strategy:

- Use a transaction isolation level of `READ_COMMITTED`. The `READ_COMMITTED` transaction isolation level prevents the S lock that is acquired by the `get` method from being held until the transaction completes. If the key is never invalidated in the transactional cache, repeatable reads are still guaranteed.
- Use alternative `get` methods instead of the `get` methods.
 - `Java` Use the `getForUpdate` method.
 - `.NET` **8.6+** Use the `GetAndLock` or `GetAndLockAll` method.

The first transaction to call to the `getForUpdate` method acquires a U lock mode instead of an S lock. This lock mode causes the second transaction to be blocked when it calls the `getForUpdate` method. One transaction is granted a U lock mode. Because the second transaction is blocked, it does not own any lock mode on the map entry. The first transaction does not block when it attempts to upgrade the U lock mode to an X lock mode as a result of the `put` method call from the first transaction. This feature demonstrates why U lock mode is called the upgradeable lock mode. When the first transaction is completed, the second transaction is unblocked and is granted the U lock mode. An application can prevent the lock promotion deadlock scenario by with the `getForUpdate` method instead of the `get` method when pessimistic locking strategy is being used.

Important: This solution does not prevent read-only transactions from being able to read a map entry. Read-only transactions call the `get` method. Read-only transactions never call the `put`, `insert`, `update`, or `remove` methods. Concurrency is as high as when the regular `get` method is used. The only reduction in concurrency occurs when the `getForUpdate` method is called by more than one transaction for the same map entry.

You must be aware when a transaction calls the `getForUpdate` method on more than one map entry to ensure that the U locks are acquired in the same order by each transaction. For example, suppose that the first transaction calls the method twice, for the key 1 for key 2. Another concurrent transaction calls the method for the same keys, but in reverse order. This sequence causes the classic deadlock because multiple locks are obtained in different orders by different

transactions. The application still must ensure that every transaction accesses multiple map entries in key sequence to ensure that deadlock does not occur. Because the U lock is obtained at the time that the `getForUpdate` method is called rather than at commit time, the eXtreme Scale cannot order the lock requests like it does during the commit cycle. The application must control the lock ordering in this case.

Data access and transactions

WebSphere eXtreme Scale uses transactions. After an application has a connection to a data grid, you can access and interact with data in the data grid.

Java

Transactions in Java applications

With Java applications, you can establish a client connection to a distributed instance or create a local instance.

When an application interacts with a Session, it must be in the context of a transaction. A transaction is begun and committed or rolled back using the `Session.begin`, `Session.commit`, and `Session.rollback` methods on the Session object. Applications can also work in auto-commit mode, where the Session automatically begins and commits a transaction whenever the application interacts with Maps. However, the auto-commit mode is slower.

A thread in a Java application needs its own Session. When you want your application to use the ObjectGrid on a thread, call one of the `getSession` methods to obtain a Session. After the application is finished with the Session, call the `Session.close()` method. This method closes the session, returning it to the pool and releasing its resources. Closing a session is optional, but improves the performance of subsequent calls to the `getSession()` method. If the application is using a dependency injection framework such as Spring, you can inject a Session into an application bean when necessary.

After you obtain a Session, the application can access data stored in maps in the ObjectGrid. If the ObjectGrid uses entities, you can use the EntityManager API, which you can obtain with the `Session.getEntityManager` method. Because it is closer to Java specifications, the EntityManager interface is simpler than the map-based API. However, the EntityManager API carries a performance overhead because it tracks changes in objects. The map-based API is obtained by using the `Session.getMap` method.

.NET

8.6+

Transactions in .NET applications

In .NET applications, each thread must have a separate `IGridMapPessimisticTx` or `IGridMapPessimisticAutoTx` object. With the `IGridMapPessimisticTx` object, you use the `Transaction` property to explicitly begin, commit or rollback the transaction. With the `IGridMapPessimisticAutoTx` object, the transaction begin, commit and rollback operations occur automatically. After you obtain one of these objects, the application can access stored data in the data grid.

The logic of using transactions

Transactions may seem to be slow. You must use transactions for the following reasons:

1. To allow rollback of changes if an exception occurs or business logic needs to undo state changes.
2. To hold locks on data and release locks within the lifetime of a transaction, allowing a set of changes to be made atomically, that is, all changes or no changes to data.
3. To produce an atomic unit of replication.
4. Java To update multiple partitions.

You can customize how much transaction support is needed. Your application can turn off rollback support and locking but at a cost to the application. The application must handle the lack of these features. Examples of how the application can manage transaction support follow:

- Java An application can turn off locking by configuring the BackingMap locking strategy to be NONE. This strategy is fast, but concurrent transactions can now modify the same data with no protection from each other. The application is responsible for all locking and data consistency when NONE is used. This option is not valid for WebSphere eXtreme Scale Client for .NET applications, which support the PESSIMISTIC locking strategy only.
- An application can change the way objects are copied when accessed by the transaction. The application can specify how objects are copied with the `ObjectMap.setCopyMode` method. With this method, you can turn off CopyMode. Turning off CopyMode is normally used for read-only transactions if different values can be returned for the same object within a transaction. Different values can be returned for the same object within a transaction.

For example, if the transaction called the `ObjectMap.get` (Java) or `IGridMapPessimisticTx.Get` (.NET) method for the object at T1, it got the value at that point in time. If it calls the get method again within that transaction at a later time T2, another thread might have changed the value. Because the value was changed by another thread, the application sees a different value.

Java If the application modifies an object retrieved using a NONE CopyMode value, it is changing the committed copy of that object directly. Rolling back the transaction has no meaning in this mode. You are changing the only copy in the ObjectGrid. Although using the NONE CopyMode is fast, be aware of its consequences. An application that uses a NONE CopyMode must never roll back the transaction. If the application rolls back the transaction, the indexes are not updated with the changes *and* the changes are not replicated if replication is turned on.

The default values are easy to use and less prone to errors. If you start trading performance in exchange for less reliable data, the application needs to be aware of what it is doing to avoid unintended problems.

CAUTION:

Be careful when you are changing either the locking or the CopyMode values. If you change the values, unpredictable application behavior occurs.

8.6+ **Transactions and partitions**

Transactions in Java applications can update a single or multiple partitions, however updating a single partition is the default behavior. .NET applications can only update a single partition.

Use the TxCommitProtocol Session API to enable multi-partition transaction support for WebSphere eXtreme Scale in a stand-alone environment. You can use the following two options:

- TxCommitProtocol.ONEPHASE (default): Transactions from a client can read from multiple partitions, but can update one partition only. Attempts made to update multiple partitions fail.
- TxCommitProtocol.TWOPHASE: Transaction from a client can read and update multiple partitions. The transaction uses the two-phase commit protocol to ensure data written to the partitions are automatically committed or rolled back. If the transaction only writes to a single partition then a one-phase commitment protocol is used.

You need to enable and configure eXtremeIO before configuring multi-transactions with WebSphere eXtreme Scale. For more information, see “Configuring IBM eXtremeIO (XIO)” on page 184.

Java

Queries and partitions

If a transaction has already searched for an Entity, the transaction is associated with the partition for that Entity. Any queries that run on a transaction that is associated with an Entity are routed to the associated partition.

If a query is run on a transaction before it is associated with a partition, you must set the partition ID to use for the query. The partition ID is an integer value. The query is then routed to that partition. This only applies if the transaction is configured to use a one-phase commitment protocol.

Queries only search within a single partition. However, if the session is set using a two-phase commitment protocol, then set the partition ID for the query to -1. This fetches results from all partitions. You can use the DataGrid APIs to run the same query in parallel on all partitions or a subset of partitions. Use the DataGrid APIs to find an entry that might be in any partition.

Transaction isolation

You can use one of three transaction isolation levels to tune the locking semantics that maintain consistency in each cache map: repeatable read, read committed and read uncommitted.

Transaction isolation overview

Transaction isolation defines how the changes that are made by one operation become visible to other concurrent operations.

You can define the following transaction isolation levels to tune the locking semantics that eXtreme Scale uses to maintain consistency in each cache map: repeatable read, read committed and read uncommitted.

You can set the transaction isolation level in one of the following ways:

- Java .NET With the **txIsolation** attribute in the ObjectGrid descriptor XML file. For more information, see ObjectGrid descriptor XML file.
- Java On the Session interface with the setTransactionIsolation method. The transaction isolation can be changed any time during the life of the session, if a transaction is not currently in progress.

- **.NET** On the IGridTransaction interface with the TransactionIsolationLevel property.

The product enforces the various transaction isolation semantics by adjusting the way in which shared (S) locks are requested and held. Transaction isolation has no effect on maps that are configured to use the optimistic locking or no locking or when upgradeable (U) locks are acquired.

Repeatable read with pessimistic locking

The repeatable read transaction isolation level is the default. This isolation level prevents dirty reads and non-repeatable reads, but does not prevent phantom reads. A dirty read is a read operation that occurs on data that has been modified by a transaction but has not been committed. A non-repeatable read might occur when read locks are not acquired when performing a read operation. A phantom read can occur when two identical read operations are performed, but two different sets of results are returned because an update has occurred on the data between the read operations. In Java applications, phantom reads are possible when you are using queries or indexes because locks are not acquired for ranges of data, only for the cache entries that match the index or query criteria. The product achieves a repeatable read by holding onto any S locks until the transaction that owns the lock completes. Because an X lock is not granted until all S locks are released, all transactions holding the S lock are guaranteed to see the same value when re-read.

Read committed with pessimistic locking

The read committed transaction isolation level can be used with eXtreme Scale, which prevents dirty reads, but does not prevent non-repeatable reads or phantom reads, so eXtreme Scale continues to use S locks to read data from the cache map, but immediately releases the locks.

Read uncommitted with pessimistic locking

The read uncommitted transaction isolation level can be used with eXtreme Scale, which is a level that allows dirty reads, non-repeatable reads and phantom reads.

Single-partition and cross-data-grid transactions

The major distinction between WebSphere eXtreme Scale and traditional data storage solutions like relational databases or in-memory databases is the use of partitioning, which allows the cache to scale linearly. The important types of transactions to consider are single-partition and every-partition (cross-data-grid) transactions.

In general, interactions with the cache can be categorized as single-partition transactions or cross-data-grid transactions.

.NET **8.6+** WebSphere eXtreme Scale Client for .NET supports single-partition transactions only.

Single-partition transactions

Single-partition transactions are the preferable method for interacting with caches that are hosted by WebSphere eXtreme Scale. When a transaction is limited to a

single partition, then by default it is limited to a single Java virtual machine, and therefore a single server computer. A server can complete M number of these transactions per second, and if you have N computers, you can complete $M*N$ transactions per second. If your business increases and you need to perform twice as many of these transactions per second, you can double N by buying more computers. Then you can meet capacity demands without changing the application, upgrading hardware, or even taking the application offline.

In addition to letting the cache scale so significantly, single-partition transactions also maximize the availability of the cache. Each transaction only depends on one computer. Any of the other $(N-1)$ computers can fail without affecting the success or response time of the transaction. So if you are running 100 computers and one of them fails, only 1 percent of the transactions in flight at the moment that server failed are rolled back. After the server fails, WebSphere eXtreme Scale relocates the partitions that are hosted by the failed server to the other 99 computers. During this brief period, before the operation completes, the other 99 computers can still complete transactions. Only the transactions that would involve the partitions that are being relocated are blocked. After the failover process is complete, the cache can continue running, fully operational, at 99 percent of its original throughput capacity. After the failed server is replaced and returned to the data grid, the cache returns to 100 percent throughput capacity.

Java

Cross-data-grid transactions

In terms of performance, availability and scalability, cross-data-grid transactions are the opposite of single-partition transactions. Cross-data-grid transactions access every partition and therefore every computer in the configuration. Each computer in the data grid is asked to look up some data and then return the result. The transaction cannot complete until every computer has responded, and therefore the throughput of the entire data grid is limited by the slowest computer. Adding computers does not make the slowest computer faster and therefore does not improve the throughput of the cache.

Cross-data-grid transactions have a similar effect on availability. Extending the previous example, if you are running 100 servers and one server fails, then 100 percent of the transactions that are in progress at the moment that server failed are rolled back. After the server fails, WebSphere eXtreme Scale starts to relocate the partitions that are hosted by that server to the other 99 computers. During this time, before the failover process completes, the data grid cannot process any of these transactions. After the failover process is complete, the cache can continue running, but at reduced capacity. If each computer in the data grid serviced 10 partitions, then 10 of the remaining 99 computers receive at least one extra partition as part of the failover process. Adding an extra partition increases the workload of that computer by at least 10 percent. Because the throughput of the data grid is limited to the throughput of the slowest computer in a cross-data-grid transaction, on average, the throughput is reduced by 10 percent.

Single-partition transactions are preferable to cross-data-grid transactions for scaling out with a distributed, highly available, object cache like WebSphere eXtreme Scale. Maximizing the performance of these kinds of systems requires the use of techniques that are different from traditional relational methodologies, but you can turn cross-data-grid transactions into scalable single-partition transactions.

Best practices for building scalable data models

The best practices for building scalable applications with products like WebSphere eXtreme Scale include two categories: foundational principles and implementation tips. Foundational principles are core ideas that need to be captured in the design of the data itself. An application that does not observe these principles is unlikely to scale well, even for its mainline transactions. Implementation tips are applied for problematic transactions in an otherwise well-designed application that observes the general principles for scalable data models.

Foundational principles

Some of the important means of optimizing scalability are basic concepts or principles to keep in mind.

Duplicate instead of normalizing

The key thing to remember about products like WebSphere eXtreme Scale is that they are designed to spread data across a large number of computers. If the goal is to make most or all transactions complete on a single partition, then the data model design needs to ensure that all the data the transaction might need is in the partition. Most of the time, the only way to achieve this is by duplicating data.

For example, consider an application like a message board. Two important transactions for a message board are showing all the posts from a user and all the posts on a topic. First, consider how these transactions would work with a normalized data model that contains a user record, a topic record, and a post record that contains the actual text. If posts are partitioned with user records, then displaying the topic becomes a cross-grid transaction, and vice versa. Topics and users cannot be partitioned together because they have a many-to-many relationship.

The best way to make this message board scale is to duplicate the posts, storing one copy with the topic record and one copy with the user record. Then, displaying the posts from a user is a single-partition transaction, displaying the posts on a topic is a single-partition transaction, and updating or deleting a post is a two-partition transaction. All three of these transactions scale linearly as the number of computers in the data grid increases.

Scalability rather than resources

The biggest obstacle to overcome when you are considering denormalized data models is the impact that these models have on resources. Keeping two, three, or more copies of some data can seem to use too many resources to be practical. When you are confronted with this scenario, remember the following facts: Hardware resources get cheaper every year. Second, and more importantly, WebSphere eXtreme Scale eliminates most hidden costs that are associated with deploying more resources.

Measure resources in terms of cost rather than computer terms such as megabytes and processors. Data stores that work with normalized relational data generally must be on the same computer. This required collocation means that a single larger enterprise computer must be purchased rather than several smaller computers. With enterprise hardware, it is not uncommon for one computer that is capable of

completing one million transactions per second to cost much more than the combined cost of 10 computers capable of doing 100,000 transactions per second each.

A business cost in adding resources also exists. A growing business eventually runs out of capacity. When you run out of capacity, you either need to shut down while moving to a bigger, faster computer, or create a second production environment to which you can switch. Either way, additional costs will come in the form of lost business or maintaining almost twice the capacity needed during the transition period.

With WebSphere eXtreme Scale, the application does not need to be shut down to add capacity. If your business projects that you need 10 percent more capacity for the coming year, then increase the number of computers in the data grid by 10 percent. You can increase this percentage without application downtime and without purchasing excess capacity.

Avoid data transformations

When you are using WebSphere eXtreme Scale, data should be stored in a format that is directly consumable by the business logic. Breaking the data down into a more primitive form is costly. The transformation needs to be done when the data is written and when the data is read. With relational databases, this transformation is done out of necessity because the data is ultimately persisted to disk frequently. With WebSphere eXtreme Scale, these transformations are not necessary. Usually, data is stored in memory and can therefore be stored in the exact form that the application needs.

Observing this simple rule helps denormalize your data in accordance with the first principle. The most common type of transformation for business data is the JOIN operations that are necessary to turn normalized data into a result set that fits the needs of the application. Storing the data in the correct format implicitly avoids performing these JOIN operations and produces a denormalized data model.

Java *Eliminate unbounded queries*

No matter how well you structure your data, unbounded queries do not scale well. For example, do not have a transaction that asks for a list of all items that are sorted by value. This transaction might work at first when the total number of items is 1000, but when the total number of items reaches 10 million, the transaction returns all 10 million items. If you run this transaction, the two most likely outcomes are the transaction timing out, or the client encounters an out-of-memory error.

The best option is to alter the business logic so that only the top 10 or 20 items can be returned. This logic alteration keeps the size of the transaction manageable no matter how many items are in the cache.

Define schema

The main advantage of normalizing data is that the database system can take care of data consistency behind the scenes. When data is denormalized for scalability, this automatic data consistency management no longer exists. You must implement a data model that can work in the application layer or as a plug-in to the distributed data grid to guarantee data consistency.

Consider the message board example. If a transaction removes a post from a topic, then the duplicate post on the user record must be removed. Without a data model, it is possible a developer might write the

application code to remove the post from the topic and forget to remove the post from the user record. However, if the developer is using a data model instead of interacting with the cache directly, the `removePost` method on the data model pulls the user ID from the post, looks up the user record, and removes the duplicate post behind the scenes.

Alternately, you can implement a listener that runs on the actual partition that detects the change to the topic and automatically adjusts the user record. A listener might be beneficial because the adjustment to the user record might happen locally if the partition happens to have the user record. If the user record is on a different partition, the transaction takes place between servers instead of between the client and server. The network connection between servers is likely to be faster than the network connection between the client and the server.

Avoid contention

Avoid scenarios such as having a global counter. The data grid does not scale if a single record is being used a disproportionate number of times compared to the rest of the records. The performance of the data grid is limited by the performance of the computer that holds the record.

In these situations, try to break up the record so it is managed per partition. For example, consider a transaction that returns the total number of entries in the distributed cache. Instead of having every insert and remove operation access a single record that increments, have a listener on each partition track the insert and remove operations. With this listener tracking, insert and remove can become single-partition operations.

Reading the counter becomes a cross-data-grid operation. Usually, it was already as inefficient as a cross-data-grid operation because its performance was tied to the performance of the computer that is hosting the record.

Implementation tips

You can also consider the following tips to achieve the best scalability.

Java *Use reverse-lookup indexes*

Consider a properly denormalized data model where customer records are partitioned based on the customer ID number. This partitioning method is the logical choice because nearly every business operation that is performed with the customer record uses the customer ID number. However, an important transaction that does not use the customer ID number is the login transaction. It is more common to have user names or email addresses for login instead of customer ID numbers.

The simple approach to the login scenario is to use a cross-data-grid transaction to find the customer record. As explained previously, this approach does not scale.

The next option might be to partition on user name or email. This option is not practical because all the customer ID-based operations become cross-data-grid transactions. Also, the customers on your site might want to change their user name or email address. Products like WebSphere eXtreme Scale need the value that is used to partition the data to remain constant.

The correct solution is to use a reverse lookup index. With WebSphere eXtreme Scale, a cache can be created in the same distributed grid as the cache that holds all the user records. This cache is highly available,

partitioned, and scalable. This cache can be used to map a user name or email address to a customer ID. This cache turns login into a two partition operation instead of a cross-grid operation. This scenario is not as good as a single-partition transaction, but the throughput still scales linearly as the number of computers increases.

Compute at write time

Commonly calculated values like averages or totals can be expensive to produce because these operations usually require reading a large number of entries. Because reads are more common than writes in most applications, it is efficient to compute these values at write time and then store the result in the cache. This practice makes read operations both faster and more scalable.

Optional fields

Consider a user record that holds a business, home, and telephone number. A user might have all, none or any combination of these numbers defined. If the data were normalized, then a user table and a telephone number table would exist. The telephone numbers for a user can be found with a JOIN operation between the two tables.

De-normalizing this record does not require data duplication, because most users do not share telephone numbers. Instead, empty slots in the user record must be allowed. Instead of having a telephone number table, add three attributes to each user record, one for each telephone number type. This addition of attributes eliminates the JOIN operation and makes a telephone number lookup for a user a single-partition operation.

Placement of many-to-many relationships

Consider an application that tracks products and the stores in which the products are sold. A single product is sold in many stores, and a single store sells many products. Assume that this application tracks 50 large retailers. Each product is sold in a maximum of 50 stores. Each store sells thousands of products.

Keep a list of stores inside the product entity (arrangement A), instead of keeping a list of products inside each store entity (arrangement B). Looking at some of the transactions this application must run illustrates why arrangement A is more scalable.

First look at updates. With arrangement A, removing a product from the inventory of a store locks the product entity. If the data grid holds 10000 products, only 1/10000 of the grid must be locked to complete the update. With arrangement B, the data grid only contains only 50 stores, so 1/50 of the data grid must be locked to complete the update. So even though both of these updates might be considered single-partition operations, arrangement A scales out more efficiently.

Now, considering reads with arrangement A, looking up the stores at which a product is sold is a single-partition transaction that scales and is fast because the transaction only transmits a small amount of data. With arrangement B, this transaction becomes a cross-data-grid transaction because each store entity must be accessed to see if the product is sold at that store, which reveals an enormous performance advantage for arrangement A.

One legitimate use of cross-data-grid transactions is to scale data processing. If a data grid has 5 computers and a cross-data-grid transaction is dispatched that sorts through about 100,000 records on each computer, then that transaction sorts through 500,000 records. If the slowest computer in the data grid can perform 10 of these transactions per second, then the data grid is capable of sorting through 5,000,000 records per second. If the data in the grid doubles, then each computer must sort through 200,000 records, and each transaction sorts through 1,000,000 records. This data increase decreases the throughput of the slowest computer to 5 transactions per second, reducing the throughput of the data grid to 5 transactions per second. Still, the data grid sorts through 5,000,000 records per second.

In this scenario, doubling the number of computers allows each computer to return to its previous load of sorting through 100,000 records, allowing the slowest computer to process 10 of these transactions per second. The throughput of the data grid stays the same at 10 requests per second, but now each transaction processes 1,000,000 records. As a result, the data grid doubled its capacity in terms of processing records to 10,000,000 per second.

Applications such as a search engine that needs to scale both in terms of data processing to accommodate the increasing size of the Internet and throughput to accommodate growth in the number of users, you must create multiple data grids, with a round robin of the requests between the data grids. If you must scale up the throughput, add computers and add another data grid to service requests. If data processing must be scaled up, add more computers and keep the number of data grids constant.

JMS for distributed transaction changes

Java

Use Java Message Service (JMS) for distributed transaction changes between different tiers or in environments on mixed platforms.

JMS is an ideal protocol for distributed changes between different tiers or in environments on mixed platforms. For example, some applications that use eXtreme Scale might be deployed on IBM WebSphere Application Server Community Edition, Apache Geronimo, or Apache Tomcat, whereas other applications might run on WebSphere Application Server Version 6.x. JMS is ideal for distributed changes between eXtreme Scale peers in these different environments. The high availability manager message transport is very fast, but can only distribute changes to Java virtual machines that are in a single core group. JMS is slower, but allows larger and more diverse sets of application clients to share an ObjectGrid. JMS is ideal when sharing data in an ObjectGrid between a fat Swing client and an application deployed on WebSphere Extended Deployment.

The built-in Client Invalidation Mechanism and Peer-to-Peer Replication are examples of JMS-based transactional changes distribution. See the information about configuring peer-to-peer replication with JMS in the *Administration Guide* for more information.

Implementing JMS

JMS is implemented for distributing transaction changes by using a Java object that behaves as an ObjectGridEventListener. This object can propagate the state in the following four ways:

1. Invalidate: Any entry that is evicted, updated or deleted is removed on all peer Java virtual machines when they receive the message.
2. Invalidate conditional: The entry is evicted only if the local version is the same or older than the version on the publisher.
3. Push: Any entry that was evicted, updated, deleted or inserted is added or overwritten on all peer Java virtual machines when they receive the JMS message.
4. Push conditional: The entry is only updated or added on the receive side if the local entry is less recent than the version that is being published.

Listen for changes for publishing

The plug-in implements the `ObjectGridEventListener` interface to intercept the `transactionEnd` event. When eXtreme Scale invokes this method, the plug-in attempts to convert the `LogSequence` list for each map that is touched by the transaction to a JMS message and then publish it. The plug-in can be configured to publish changes for all maps or a subset of maps. `LogSequence` objects are processed for the maps that have publishing enabled. The `LogSequenceTransformer` `ObjectGrid` class serializes a filtered `LogSequence` for each map to a stream. After all `LogSequences` are serialized to the stream, then a JMS `ObjectMessage` is created and published to a well-known topic.

Listen for JMS messages and apply them to the local ObjectGrid

The same plug-in also starts a thread that spins in a loop, receiving all messages that are published to the well known topic. When a message arrives, it passes the message contents to the `LogSequenceTransformer` class where it is converted to a set of `LogSequence` objects. Then, a no-write-through transaction is started. Each `LogSequence` object is provided to the `Session.processLogSequence` method, which updates the local Maps with the changes. The `processLogSequence` method understands the distribution mode. The transaction is committed and the local cache now reflects the changes. For more information about using JMS to distribute transaction changes, see the information about distributing changes between peer Java Virtual Machines in the *Administration Guide*.

Two-phase commit and error recovery

Java

The two-phase commit protocol coordinates all the partitions that participate in a distributed transaction on whether to commit or roll back the transaction.

In a distributed data grid, partitions are distributed across multiple Java virtual machines (JVM). These JVMs can be on more than one system. A transaction that writes to multiple partitions might involve transactional decisions that affect more than one system. When the transaction is committed with a two-phase commit protocol, this commit process ensures that the entire transaction is persisted, or none of transaction is persisted. The two-phase commit process ensures this outcome despite partition, system, or communication failures. If a failure occurs in the second phase, the WebSphere eXtreme Scale client attempts to resolve the failure automatically, unless the error meets certain criteria for which you can manually intervene.

A transaction that is enabled to write to multiple partitions uses the two-phase commit protocol. A two-phase commit protocol ensures that the commit process is

consistent across all partitions and systems. WebSphere eXtreme Scale acts as the coordinator that controls the two-phase commit process. The partitions that are involved in the transaction are called the participants or resource managers (RM). During the second phase of the commit protocol, the coordinator delegates one of the partitions to act as the transaction manager (TM). The TM is responsible for tracking the decision of each transaction and recovering the transaction if a failure occurs.

First phase:

When an application commits a transaction, WebSphere eXtreme Scale client starts the first phase by sending a prepare to commit request to each partition identified as an RM. Each partition applies the transaction changes to the backing maps and holds all locks to ensure data integrity. The RM notifies WebSphere eXtreme Scale client. After all partitions identified as an RM respond with success, WebSphere eXtreme Scale client begins the second phase of the commit protocol.

Second phase:

If at least one partition fails during the first phase, then the coordinator rolls back all partitions during the second phase. If all RM partitions respond with success, then the WebSphere eXtreme Scale client delegates one of the partitions to act as the TM partition. As the coordinator, WebSphere eXtreme Scale begins the second phase of the commit protocol by sending a commit or a rollback request to all partitions that are involved in the transaction. Each partition that is identified as an RM then either applies or rolls back the changes to the backing map and releases all the locks. The RM then notifies WebSphere eXtreme Scale client. If at least one partition failed during the second phase, then the delegated TM partition automatically recovers the transaction. Automatic recovery ensures all the partitions that are involved in the transaction are consistent.

In doubt phase:

The indoubt phase is the period between when the RM partition successfully processes the first phase, and is waiting to begin the second phase. During the indoubt period, the RM partition does not know whether to commit or roll back the transaction. The RM partition holds onto locks. Holding the locks can result in an increase in lock contention for other transactions.

Error recovery during a two-phase commit

If a failure occurs during the first phase, WebSphere eXtreme Scale client rolls back the transaction. If one of the partitions fails to commit the transaction, then the TM ensures that the transaction is committed by periodically attempting to commit the transaction. An example of log messages that occur in this scenario follow:

```
00000099 TransactionLog I CW0BJ8705I: Automatic resolution of transaction
WXS-40000139-DF01-216D-E002-1CB456931719 at RM:TestGrid:TestSet2:20 is
still waiting for a decision. Another attempt to resolve the transaction
will occur in 30 seconds.
```

Allow WebSphere eXtreme Scale client to resolve the transaction. Attempt to intervene manually only if the transaction is not recovered within 1 minute or the application is experiencing a high volume of lock contention because it is an indoubt transaction. For more information about how to manually recover a transaction, see [Troubleshooting lock timeout exceptions for a multi-partition transaction](#).

Security overview

WebSphere eXtreme Scale can secure data access, including allowing for integration with external security providers.

Note: In an existing non-cached data store such as a database, you likely have built-in security features that you might not need to actively configure or enable. However, after you have cached your data with eXtreme Scale, you must consider the important resulting situation that your backend security features are no longer in effect. You can configure eXtreme Scale security on necessary levels so that your new cached architecture for your data is also secured.

A brief summary of eXtreme Scale security features follows. For more detailed information about configuring security see the *Administration Guide* and the *Programming Guide*.

Distributed security basics

Distributed eXtreme Scale security is based on three key concepts:

Trustable authentication

The ability to determine the identity of the requester. WebSphere eXtreme Scale supports both client-to-server and server-to-server authentication.

Authorization

The ability to give permissions to grant access rights to the requester. WebSphere eXtreme Scale supports different authorizations for various operations.

Secure transport

The safe transmission of data over a network. WebSphere eXtreme Scale supports the Transport Layer Security/Secure Sockets Layer (TLS/SSL) protocols.

Authentication

WebSphere eXtreme Scale supports a distributed client server framework. A client server security infrastructure is in place to secure access to eXtreme Scale servers. For example, when authentication is required by the eXtreme Scale server, an eXtreme Scale client must provide credentials to authenticate to the server. These credentials can be a user name and password pair, a client certificate, a Kerberos ticket, or data that is presented in a format that is agreed upon by client and server.

Authorization

WebSphere eXtreme Scale authorizations are based on subjects and permissions. You can use the Java Authentication and Authorization Services (JAAS) to authorize the access, or you can plug in a custom approach, such as Tivoli® Access Manager (TAM), to handle the authorizations. The following authorizations can be given to a client or group:

Map authorization

Perform insert, read, update, evict, or delete operations on Maps.

ObjectGrid authorization

Perform object or entity queries on ObjectGrid objects.

DataGrid agent authorization

Allow DataGrid agents to be deployed to an ObjectGrid.

Server side map authorization

Replicate a server map to client side or create a dynamic index to the server map.

Administration authorization

Perform administration tasks.

Transport security

To secure the client server communication, WebSphere eXtreme Scale supports TLS/SSL. These protocols provide transport layer security with authenticity, integrity, and confidentiality for a secure connection between an eXtreme Scale client and server.

Grid security

In a secure environment, a server must be able to check the authenticity of another server. WebSphere eXtreme Scale uses a shared secret key string mechanism for this purpose. This secret key mechanism is similar to a shared password. All the eXtreme Scale servers agree on a shared secret string. When a server joins the data grid, the server is challenged to present the secret string. If the secret string of the joining server matches the one in the master server, then the joining server can join the grid. Otherwise, the join request is rejected.

Sending a clear text secret is not secure. The eXtreme Scale security infrastructure provides a SecureTokenManager plug-in to allow the server to secure this secret before sending it. You can choose how you implement the secure operation. WebSphere eXtreme Scale provides an implementation, in which the secure operation is implemented to encrypt and sign the secret.

Java Management Extensions (JMX) security in a dynamic deployment topology

JMX MBean security is supported in all versions of eXtreme Scale. Clients of catalog server MBeans and container server MBeans can be authenticated, and access to MBean operations can be enforced.

Local eXtreme Scale security

Local eXtreme Scale security is different from the distributed eXtreme Scale model because the application directly instantiates and uses an ObjectGrid instance. Your application and eXtreme Scale instances are in the same Java virtual machine (JVM). Because no client-server concept exists in this model, authentication is not supported. Your applications must manage their own authentication, and then pass the authenticated Subject object to the eXtreme Scale. However, the authorization mechanism that is used for the local eXtreme Scale programming model is the same as what is used for the client-server model.

Configuration and programming

For more information about configuring and programming for security, see Security integration with external providers and Security API.

REST data services overview

Java

The WebSphere eXtreme Scale REST data service is a Java HTTP service that is compatible with Microsoft WCF Data Services (formerly ADO.NET Data Services) and implements the Open Data Protocol (OData). Microsoft WCF Data Services is compatible with this specification when using Visual Studio 2008 SP1 and the .NET Framework 3.5 SP1.

Compatibility requirements

The REST data service allows any HTTP client to access a data grid. The REST data service is compatible with the WCF Data Services support supplied with the Microsoft .NET Framework 3.5 SP1. RESTful applications can be developed with the rich tooling provided by Microsoft Visual Studio 2008 SP1. The figure provides an overview of how WCF Data Services interacts with clients and databases.

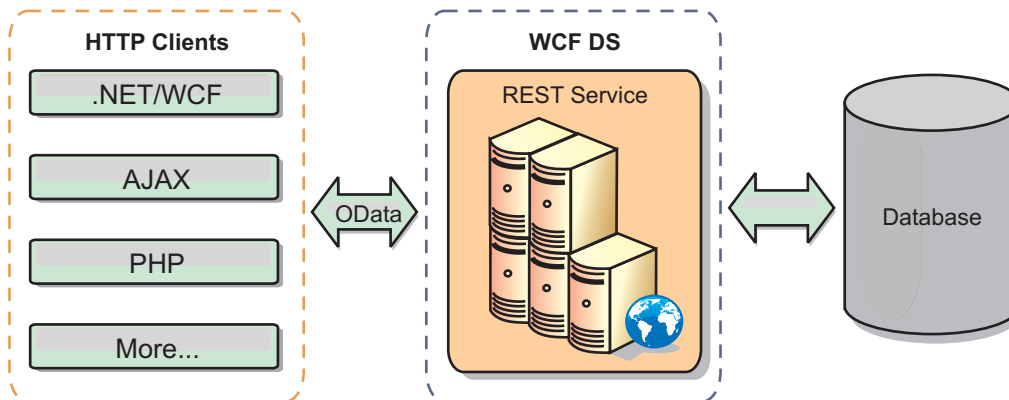


Figure 41. Microsoft WCF Data Services

WebSphere eXtreme Scale includes a function-rich API set for Java clients. As shown in the following figure, the REST data service is a gateway between HTTP clients and the WebSphere eXtreme Scale data grid, communicating with the grid through an WebSphere eXtreme Scale client. The REST data service is a Java servlet, which allows flexible deployments for common Java Platform, Enterprise Edition (JEE) platforms, such as WebSphere Application Server. The REST data service communicates with the WebSphere eXtreme Scale data grid using the WebSphere eXtreme Scale Java APIs. It allows WCF Data Services clients or any other client that can communicate with HTTP and XML.

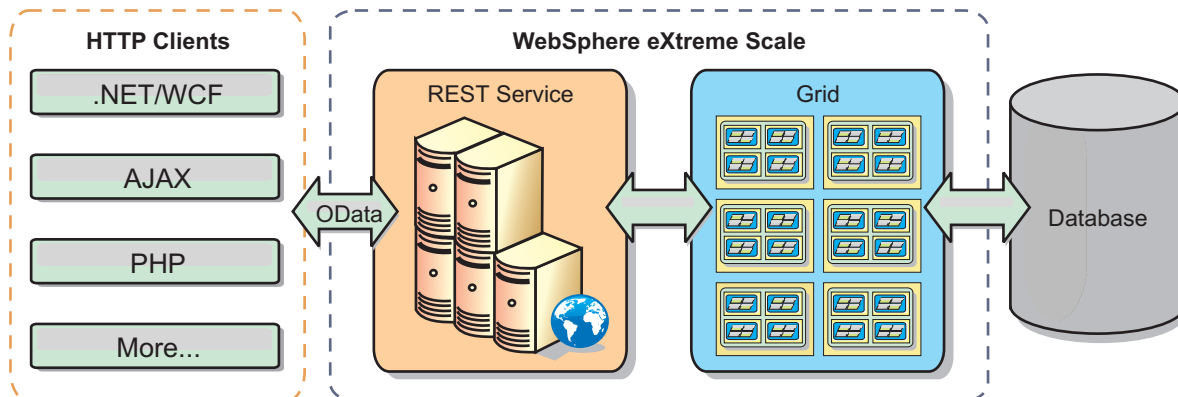


Figure 42. WebSphere eXtreme Scale REST data service

Refer to the Configuring REST data services, or use the following links to learn more about WCF Data Services.

- Microsoft WCF Data Services Developer Center
- ADO.NET Data Services overview on MSDN
- Whitepaper: Using ADO.NET Data Services
- Atom Publish Protocol: Data Services URI and Payload Extensions
- Conceptual Schema Definition File Format
- Entity Data Model for Data Services Packaging Format
- Open Data Protocol
- Open Data Protocol FAQ


Features

This version of the eXtreme Scale REST data service supports the following features:

- Automatic modeling of eXtreme Scale EntityManager API entities as WCF Data Services entities, which includes the following support:
 - Java data type to Entity Data Model type conversion
 - Entity association support
 - Schema root and key association support, which is required for partitioned data grids

See Entity model for more information.

- Atom Publish Protocol (AtomPub or APP) XML and JavaScript Object Notation (JSON) data payload format.
- Create, Read, Update and Delete (CRUD) operations using the respective HTTP request methods: POST, GET, PUT and DELETE. In addition, the Microsoft extension: MERGE is supported.

Note:  **8.6+** The upsert and upsertAll methods replace the ObjectMap put and putAll methods. Use the upsert method to tell the BackingMap and loader that an entry in the data grid needs to place the key and value into the grid. The BackingMap and loader does either an insert or an update to place the value into the grid and loader. If you run the upsert API within your applications, then the loader gets an UPSERT LogElement type, which allows loaders to do database merge or upsert calls instead of using insert or update.

- Simple queries, using filters
- Batch retrieval and change set requests
- Partitioned data grid support for high availability
- Interoperability with eXtreme Scale EntityManager API clients
- Support for standard JEE Web servers
- Optimistic concurrency
- User authorization and authentication between the REST data service and the eXtreme Scale data grid

Known problems and limitations

- Tunneled requests are not supported.

Chapter 2. Planning



Before you install WebSphere eXtreme Scale and deploy your data grid applications, you must decide on your caching topology, complete capacity planning, review the hardware and software requirements, networking and tuning settings, and so on. You can also use the operational checklist to ensure that your environment is ready to have an application deployed.

For a discussion of the best practices that you can use when you are designing your WebSphere eXtreme Scale applications, read the following article on developerWorks®: Principles and best practices for building high performing and highly resilient WebSphere eXtreme Scale applications.

Planning the topology

With WebSphere eXtreme Scale, your architecture can use local in-memory data caching or distributed client-server data caching. The architecture can have varied relationships with your databases. You can also configure the topology to span multiple data centers.

WebSphere eXtreme Scale requires minimal additional infrastructure to operate. The infrastructure consists of scripts to install, start, and stop a Java Platform, Enterprise Edition application on a server. Cached data is stored in the container servers, and clients remotely connect to the server.

In-memory environments

When you deploy in a local, in-memory environment, WebSphere eXtreme Scale runs within a single Java virtual machine and is not replicated. To configure a local environment you can use an ObjectGrid XML file or the ObjectGrid APIs.

Distributed environments

When you deploy in a distributed environment, WebSphere eXtreme Scale runs across a set of Java virtual machines, increasing the performance, availability and scalability. With this configuration, you can use data replication and partitioning. You can also add additional servers without restarting your existing eXtreme Scale servers. As with a local environment, an ObjectGrid XML file, or an equivalent programmatic configuration, is needed in a distributed environment. You must also provide a deployment policy XML file with configuration details

You can create either simple deployments or large, terabyte-sized deployments in which thousands of servers are needed.

Local in-memory cache

In the simplest case, WebSphere eXtreme Scale can be used as a local (non-distributed) in-memory data grid cache. The local case can especially benefit high-concurrency applications where multiple threads need to access and modify transient data. The data kept in a local data grid can be indexed and retrieved using queries. Queries help you to work with large in memory data sets. The support provided with the Java virtual machine (JVM), although it is ready to use, has a limited data structure.

The local in-memory cache topology for WebSphere eXtreme Scale is used to provide consistent, transactional access to temporary data within a single Java virtual machine.

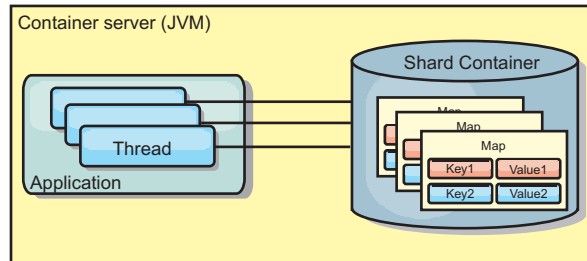


Figure 43. Local in-memory cache scenario

Advantages

- Simple setup: An ObjectGrid can be created programmatically or declaratively with the ObjectGrid deployment descriptor XML file or with other frameworks such as Spring.
- Fast: Each BackingMap can be independently tuned for optimal memory utilization and concurrency.
- Ideal for single-Java virtual machine topologies with small dataset or for caching frequently accessed data.
- Transactional. BackingMap updates can be grouped into a single unit of work and can be integrated as a last participant in 2-phase transactions such as Java Transaction Architecture (JTA) transactions.

Disadvantages

- Not fault tolerant.
- The data is not replicated. In-memory caches are best for read-only reference data.
- Not scalable. The amount of memory required by the database might overwhelm the Java virtual machine.
- Problems occur when adding Java virtual machines:
 - Data cannot easily be partitioned
 - Must manually replicate state between Java virtual machines or each cache instance could have different versions of the same data.
 - Invalidation is expensive.
 - Each cache must be warmed up independently. The warm-up is the period of loading a set of data so that the cache gets populated with valid data.

When to use

The local, in-memory cache deployment topology should only be used when the amount of data to be cached is small (can fit into a single Java virtual machine) and is relatively stable. Stale data must be tolerated with this approach. Using evictors to keep most frequently or recently used data in the cache can help keep the cache size low and increase relevance of the data.

Peer-replicated local cache

You must ensure the cache is synchronized if multiple processes with independent cache instances exist. To ensure that the cache instances are synchronized, enable a peer-replicated cache with Java Message Service (JMS).

WebSphere eXtreme Scale includes two plug-ins that automatically propagate transaction changes between peer ObjectGrid instances. The JMSObjectGridEventListener plug-in automatically propagates eXtreme Scale changes using JMS.

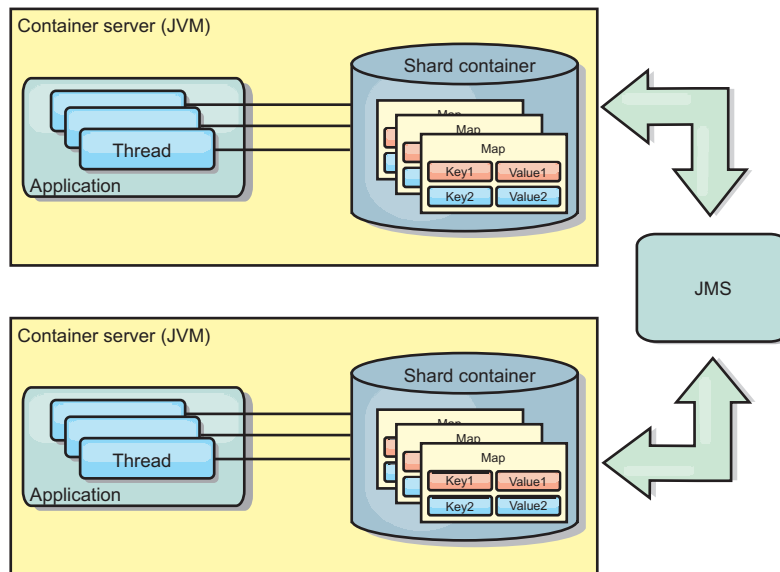


Figure 44. Peer-replicated cache with changes that are propagated with JMS

If you are running a WebSphere Application Server environment, the TranPropListener plug-in is also available. The TranPropListener plug-in uses the high availability (HA) manager to propagate the changes to each peer cache instance.

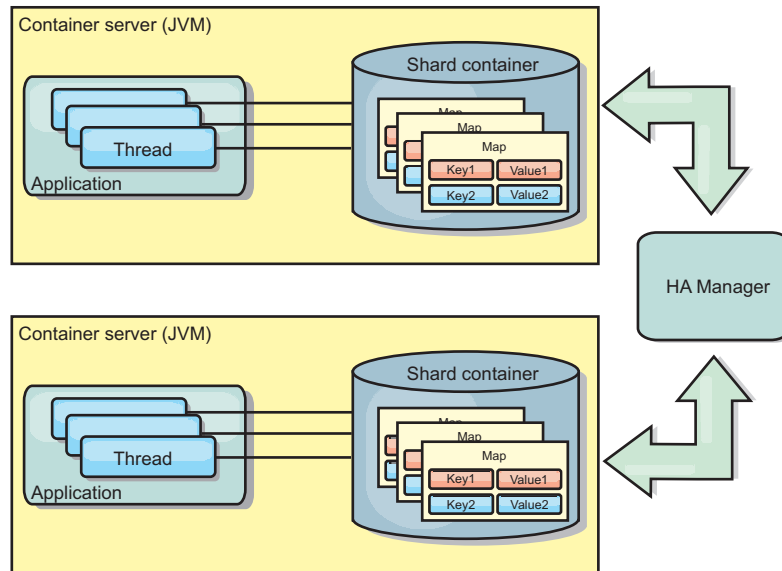


Figure 45. Peer-replicated cache with changes that are propagated with the high availability manager

Advantages

- The data is more valid because the data is updated more often.
- With the TranPropListener plug-in, like the local environment, the eXtreme Scale can be created programmatically or declaratively with the eXtreme Scale deployment descriptor XML file or with other frameworks such as Spring. Integration with the high availability manager is done automatically.
- Each BackingMap can be independently tuned for optimal memory utilization and concurrency.
- BackingMap updates can be grouped into a single unit of work and can be integrated as a last participant in 2-phase transactions such as Java Transaction Architecture (JTA) transactions.
- Ideal for few-JVM topologies with a reasonably small dataset or for caching frequently accessed data.
- Changes to the eXtreme Scale are replicated to all peer eXtreme Scale instances. The changes are consistent as long as a durable subscription is used.

Disadvantages

- Configuration and maintenance for the JMSObjectGridEventListener can be complex. eXtreme Scale can be created programmatically or declaratively with the eXtreme Scale deployment descriptor XML file or with other frameworks such as Spring.
- Not scalable: The amount of memory required by the database may overwhelm the JVM.
- Functions improperly when adding Java virtual machines:
 - Data cannot easily be partitioned
 - Invalidation is expensive.
 - Each cache must be warmed-up independently

When to use

Use deployment topology only when the amount of data to be cached is small, can fit into a single JVM, and is relatively stable.

Embedded cache

WebSphere eXtreme Scale grids can run within existing processes as embedded eXtreme Scale servers or you can manage them as external processes.

Embedded grids are useful when you are running in an application server, such as WebSphere Application Server. You can start eXtreme Scale servers that are not embedded by using command line scripts and run in a Java process.

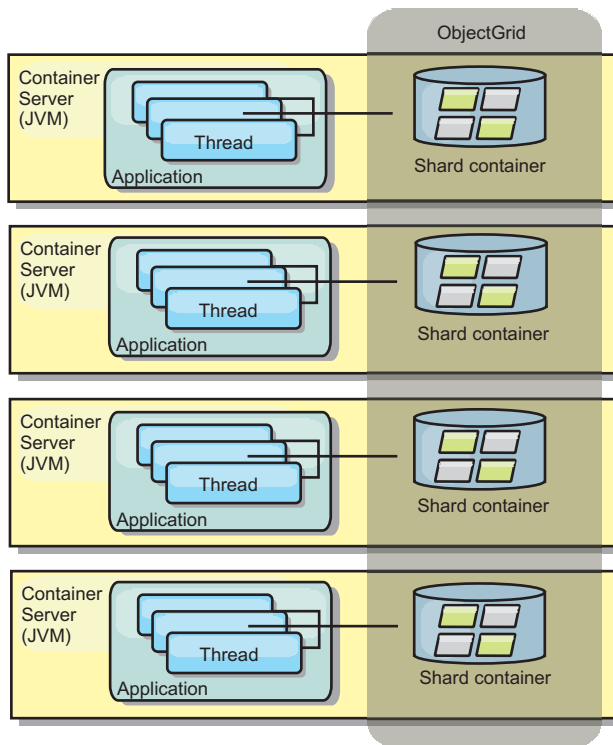


Figure 46. Embedded cache

Advantages

- Simplified administration since there are less processes to manage.
- Simplified application deployment since the grid is using the client application classloader.
- Supports partitioning and high availability.

Disadvantages

- Increased the memory footprint in client process since all of the data is collocated in the process.
- Increase CPU utilization for servicing client requests.
- More difficult to handle application upgrades since clients are using the same application Java archive files as the servers.
- Less flexible. Scaling of clients and grid servers cannot increase at the same rate. When servers are externally defined, you can have more flexibility in managing the number of processes.

When to use

Use embedded grids when there is plenty of memory free in the client process for grid data and potential failover data.

For more information, see the topic on enabling the client invalidation mechanism in the *Administration Guide*.

Distributed cache

WebSphere eXtreme Scale is most often used as a shared cache, to provide transactional access to data to multiple components where a traditional database would otherwise be used. The shared cache eliminates the need to configure a database.

Coherency of the cache

The cache is coherent because all of the clients see the same data in the cache. Each piece of data is stored on exactly one server in the cache, preventing wasteful copies of records that could potentially contain different versions of the data. A coherent cache can also hold more data as more servers are added to the data grid, and scales linearly as the grid grows in size. Because clients access data from this data grid with remote procedural calls, it can also be known as a remote cache, or far cache. Through data partitioning, each process holds a unique subset of the total data set. Larger data grids can both hold more data and service more requests for that data. Coherency also eliminates the need to push invalidation data around the data grid because no stale data exists. The coherent cache only holds the latest copy of each piece of data.

If you are running a WebSphere Application Server environment, the TranPropListener plug-in is also available. The TranPropListener plug-in uses the high availability component (HA Manager) of WebSphere Application Server to propagate the changes to each peer ObjectGrid cache instance.

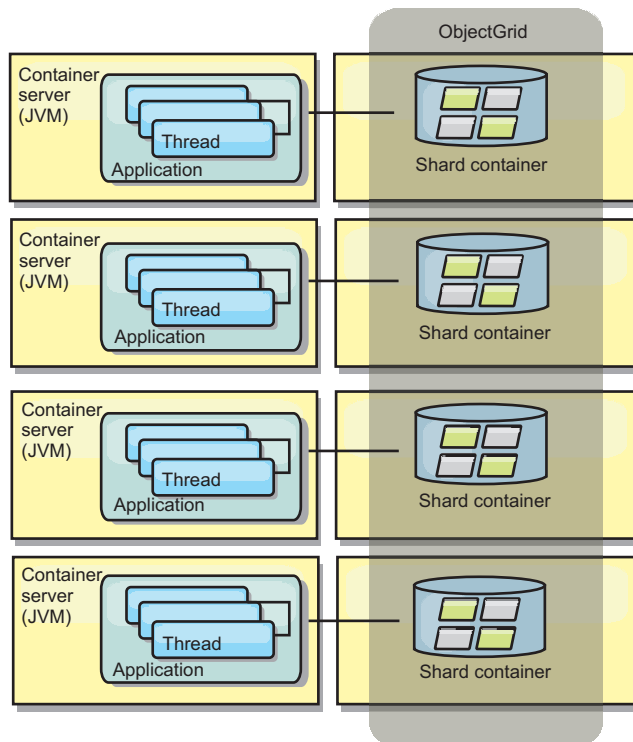


Figure 47. Distributed cache

Near cache

Clients can optionally have a local, in-line cache when eXtreme Scale is used in a distributed topology. This optional cache is called a near cache, an independent ObjectGrid on each client, serving as a cache for the remote, server-side cache. The near cache is enabled by default when locking is configured as optimistic or none and cannot be used when configured as pessimistic.

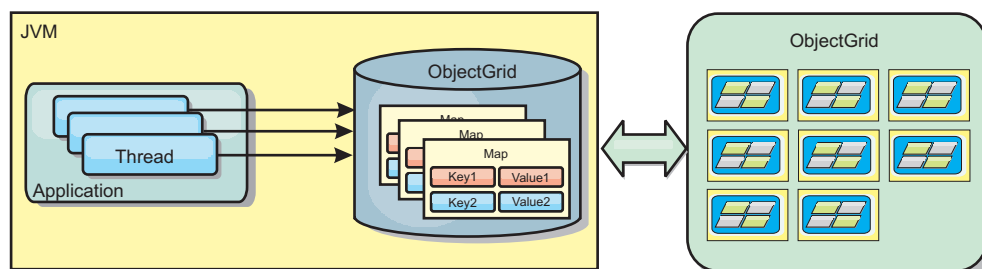


Figure 48. Near cache

A near cache is very fast because it provides in-memory access to a subset of the entire cached data set that is stored remotely in the eXtreme Scale servers. The near cache is not partitioned and contains data from any of the remote eXtreme Scale partitions. WebSphere eXtreme Scale can have up to three cache tiers as follows.

1. The transaction tier cache contains all changes for a single transaction. The transaction cache contains a working copy of the data until the transaction is committed. When a client transaction requests data from an ObjectMap, the transaction is checked first.

2. The near cache in the client tier contains a subset of the data from the server tier. When the transaction tier does not have the data, the data is fetched from the client tier, if available and inserted into the transaction cache
3. The data grid in the server tier contains the majority of the data and is shared among all clients. The server tier can be partitioned, which allows a large amount of data to be cached. When the client near cache does not have the data, it is fetched from the server tier and inserted into the client cache. The server tier can also have a Loader plug-in. When the data grid does not have the requested data, the Loader is invoked and the resulting data is inserted from the backend data store into the grid.

To disable the near cache, see [Configuring the near cache](#).

Advantage

- Fast response time because all access to the data is local. Looking for the data in the near cache first saves a trip to the grid of servers, thus making even the remote data locally accessible.

Disadvantages

- Increases duration of stale data because the near cache at each tier may be out of synch with the current data in the data grid.
- Relies on an evictor to invalidate data to avoid running out of memory.

When to use

Use when response time is important and stale data can be tolerated.

Database integration: Write-behind, in-line, and side caching

WebSphere eXtreme Scale is used to front a traditional database and eliminate read activity that is normally pushed to the database. A coherent cache can be used with an application directly or indirectly using an object relational mapper. The coherent cache can then offload the database or backend from reads. In a slightly more complex scenario, such as transactional access to a data set where only some of the data requires traditional persistence guarantees, filtering can be used to offload even write transactions.

You can configure WebSphere eXtreme Scale to function as a highly flexible in-memory database processing space. However, WebSphere eXtreme Scale is not an object relational mapper (ORM). It does not know where the data in the data grid came from. An application or an ORM can place data in an eXtreme Scale server. It is the responsibility of the source of the data to make sure that it stays consistent with the database where data originated. This means eXtreme Scale cannot invalidate data that is pulled from a database automatically. The application or mapper must provide this function and manage the data stored in eXtreme Scale.

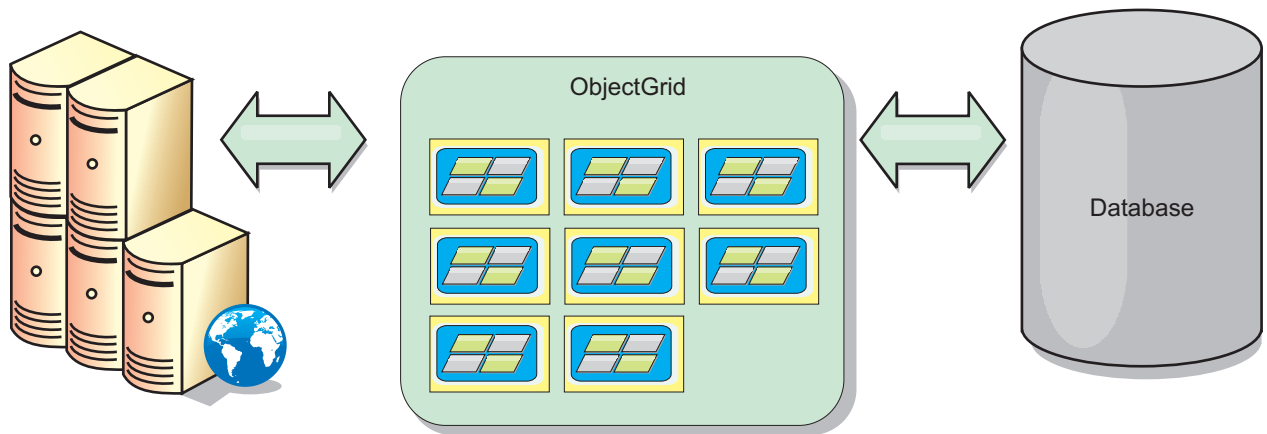


Figure 49. ObjectGrid as a database buffer

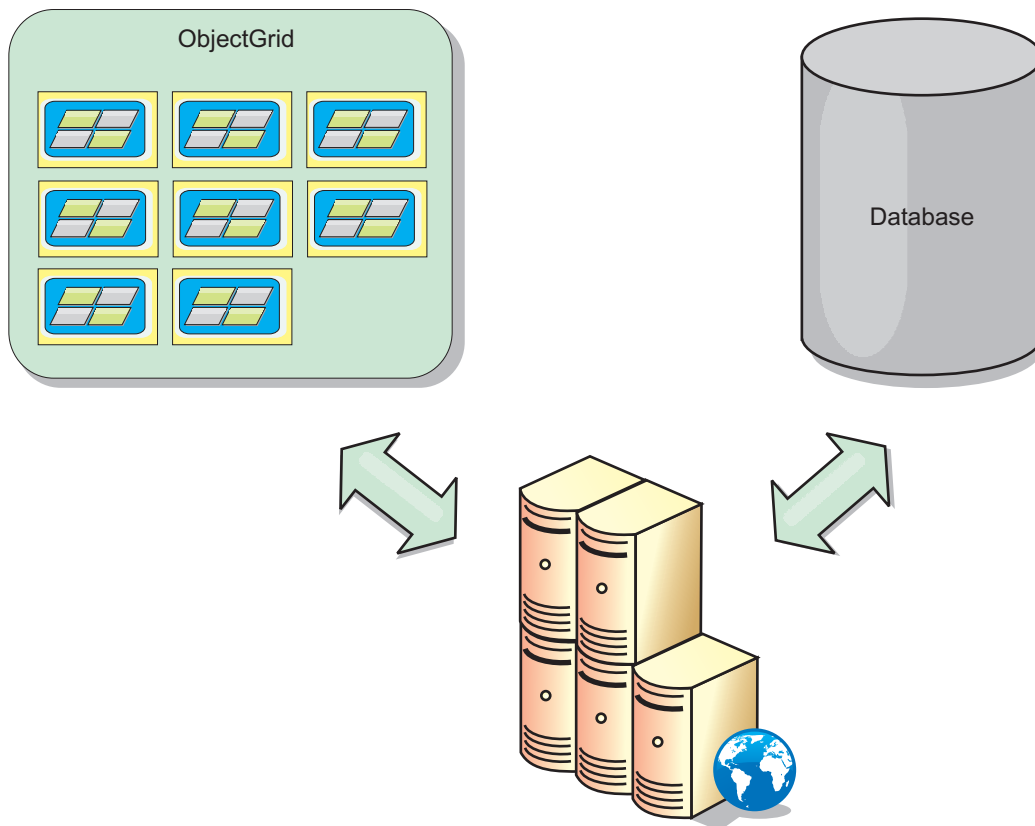


Figure 50. ObjectGrid as a side cache

Sparse and complete cache

WebSphere eXtreme Scale can be used as a sparse cache or a complete cache. A sparse cache only keeps a subset of the total data, while a complete cache keeps all of the data, and can be populated lazily, as the data is needed. Sparse caches are normally accessed using keys (instead of indexes or queries) because the data is only partially available.

Sparse cache

When a key is not present in a sparse cache, or the data is not available and a cache miss occurs, the next tier is invoked. The data is fetched, from a database for example, and is inserted into the data grid cache tier. If you are using a query or index, only the currently loaded values are accessed and the requests are not forwarded to the other tiers.

Complete cache

A complete cache contains all of the required data and can be accessed using non-key attributes with indexes or queries. A complete cache is preloaded with data from the database before the application tries to access the data. A complete cache can function as a database replacement after data is loaded. Because all of the data is available, queries and indexes can be used to find and aggregate data.

Side cache

When WebSphere eXtreme Scale is used as a side cache, the back end is used with the data grid.

Side cache

You can configure the product as a side cache for the data access layer of an application. In this scenario, WebSphere eXtreme Scale is used to temporarily store objects that would normally be retrieved from a back-end database. Applications check to see if the data grid contains the data. If the data is in the data grid, the data is returned to the caller. If the data does not exist, the data is retrieved from the back-end database. The data is then inserted into the data grid so that the next request can use the cached copy. The following diagram illustrates how WebSphere eXtreme Scale can be used as a side-cache with an arbitrary data access layer such as OpenJPA or Hibernate.

Side cache plug-ins for Hibernate and OpenJPA

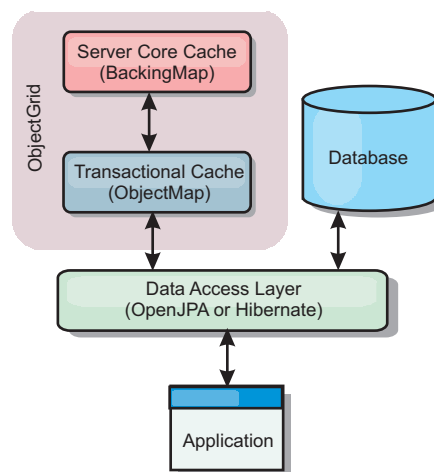


Figure 51. Side cache

Cache plug-ins for both OpenJPA and Hibernate are included in WebSphere eXtreme Scale, so you can use the product as an automatic side-cache. Using WebSphere eXtreme Scale as a cache provider increases performance when reading and querying data and reduces load to the database. There are advantages that

WebSphere eXtreme Scale has over built-in cache implementations because the cache is automatically replicated between all processes. When one client caches a value, all other clients can use the cached value.

In-line cache

You can configure in-line caching for a database back end or as a side cache for a database. In-line caching uses eXtreme Scale as the primary means for interacting with the data. When eXtreme Scale is used as an in-line cache, the application interacts with the back end using a Loader plug-in.

In-line cache

When used as an in-line cache, WebSphere eXtreme Scale interacts with the back end using a Loader plug-in. This scenario can simplify data access because applications can access the eXtreme Scale APIs directly. Several different caching scenarios are supported in eXtreme Scale to make sure the data in the cache and the data in the back end are synchronized. The following diagram illustrates how an in-line cache interacts with the application and back end.

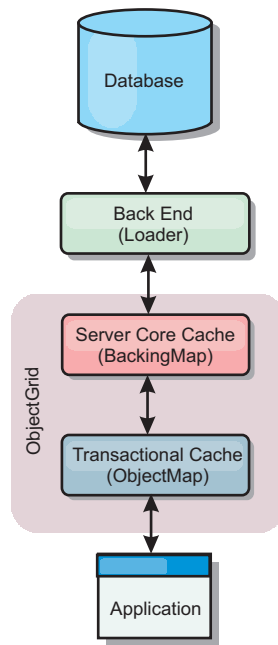


Figure 52. In-line cache

The in-line caching option simplifies data access because it allows applications to access the eXtreme Scale APIs directly. WebSphere eXtreme Scale supports several in-line caching scenarios, as follows.

- Read-through
- Write-through
- Write-behind

Read-through caching scenario

A read-through cache is a sparse cache that lazily loads data entries by key as they are requested. This is done without requiring the caller to know how the entries are populated. If the data cannot be found in the eXtreme Scale cache, eXtreme Scale will retrieve the missing data from the Loader plug-in, which loads the data

from the back-end database and inserts the data into the cache. Subsequent requests for the same data key will be found in the cache until it is removed, invalidated or evicted.

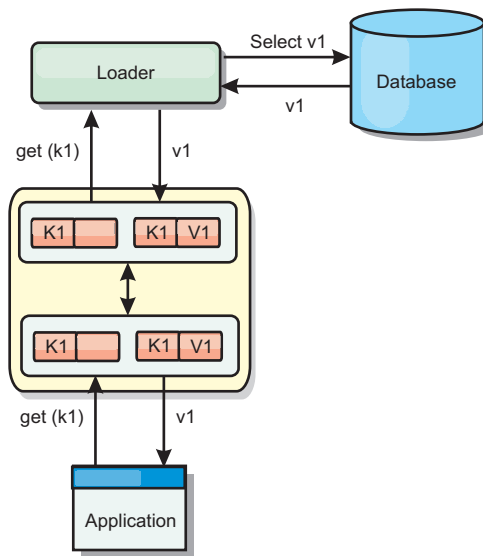


Figure 53. Read-through caching

Write-through caching scenario

In a write-through cache, every write to the cache synchronously writes to the database using the Loader. This method provides consistency with the back end, but decreases write performance since the database operation is synchronous. Since the cache and database are both updated, subsequent reads for the same data will be found in the cache, avoiding the database call. A write-through cache is often used in conjunction with a read-through cache.

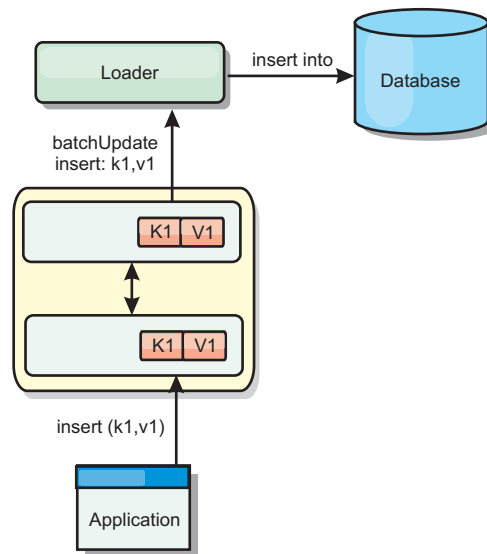


Figure 54. Write-through caching

Write-behind caching scenario

Database synchronization can be improved by writing changes asynchronously. This is known as a write-behind or write-back cache. Changes that would normally be written synchronously to the loader are instead buffered in eXtreme Scale and written to the database using a background thread. Write performance is significantly improved because the database operation is removed from the client transaction and the database writes can be compressed.

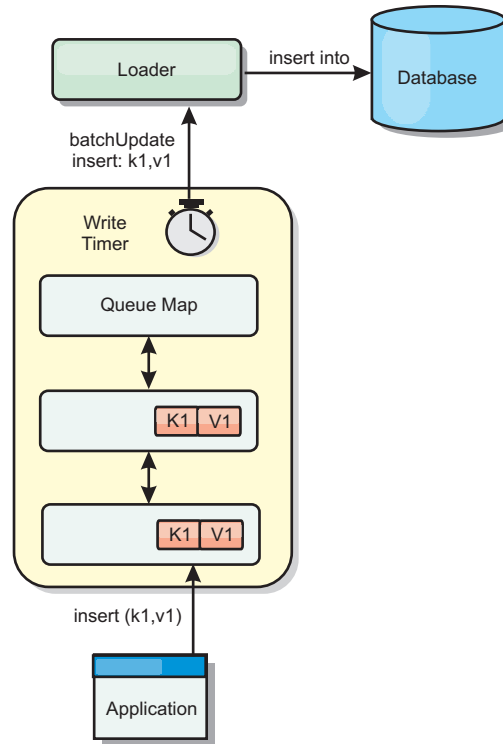


Figure 55. Write-behind caching

Write-behind caching

Java

You can use write-behind caching to reduce the overhead that occurs when updating a database you are using as a back end.

Write-behind caching overview

Write-behind caching asynchronously queues updates to the Loader plug-in. You can improve performance by disconnecting updates, inserts, and removes for a map, the overhead of updating the back-end database. The asynchronous update is performed after a time-based delay (for example, five minutes) or an entry-based delay (1000 entries).

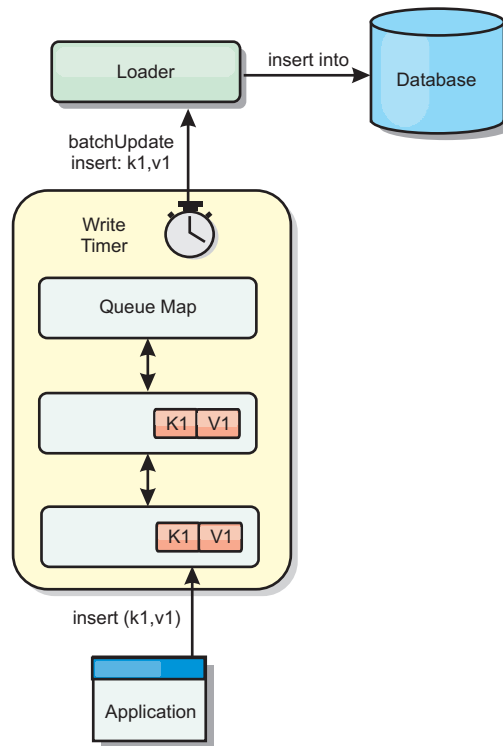


Figure 56. Write-behind caching

The write-behind configuration on a `BackingMap` creates a thread between the loader and the map. The loader then delegates data requests through the thread according to the configuration settings in the `BackingMap.setWriteBehind` method. When an eXtreme Scale transaction inserts, updates, or removes an entry from a map, a `LogElement` object is created for each of these records. These elements are sent to the write-behind loader and queued in a special `ObjectMap` called a queue map. Each backing map with the write-behind setting enabled has its own queue maps. A write-behind thread periodically removes the queued data from the queue maps and pushes them to the real back-end loader.

The write-behind loader only sends insert, update, and delete types of `LogElement` objects to the real loader. All other types of `LogElement` objects, for example, `EVICT` type, are ignored.

Write-behind support is an extension of the Loader plug-in, which you use to integrate eXtreme Scale with the database. For example, consult the [Configuring JPA loaders](#) information about configuring a JPA loader.

Benefits

Enabling write-behind support has the following benefits:

- **Back end failure isolation:** Write-behind caching provides an isolation layer from back end failures. When the back-end database fails, updates are queued in the queue map. The applications can continue driving transactions to eXtreme Scale. When the back end recovers, the data in the queue map is pushed to the back-end.
- **Reduced back end load:** The write-behind loader merges the updates on a key basis so only one merged update per key exists in the queue map. This merge decreases the number of updates to the back-end database.

- **Improved transaction performance:** Individual eXtreme Scale transaction times are reduced because the transaction does not need to wait for the data to be synchronized with the back-end.

Loaders

Java

With a Loader plug-in, a data grid map can behave as a memory cache for data that is typically kept in a persistent store on either the same system or another system. Typically, a database or file system is used as the persistent store. A remote Java virtual machine (JVM) can also be used as the source of data, allowing hub-based caches to be built using eXtreme Scale. A loader has the logic for reading and writing data to and from a persistent store.

Overview

Loaders are backing map plug-ins that are invoked when changes are made to the backing map or when the backing map is unable to satisfy a data request (a cache miss). The Loader is invoked when the cache is unable to satisfy a request for a key, providing read-through capability and lazy-population of the cache. A loader also allows updates to the database when cache values change. All changes within a transaction are grouped together to allow the number of database interactions to be minimized. A TransactionCallback plug-in is used in conjunction with the loader to trigger the demarcation of the backend transaction. Using this plug-in is important when multiple maps are included in a single transaction or when transaction data is flushed to the cache without committing.

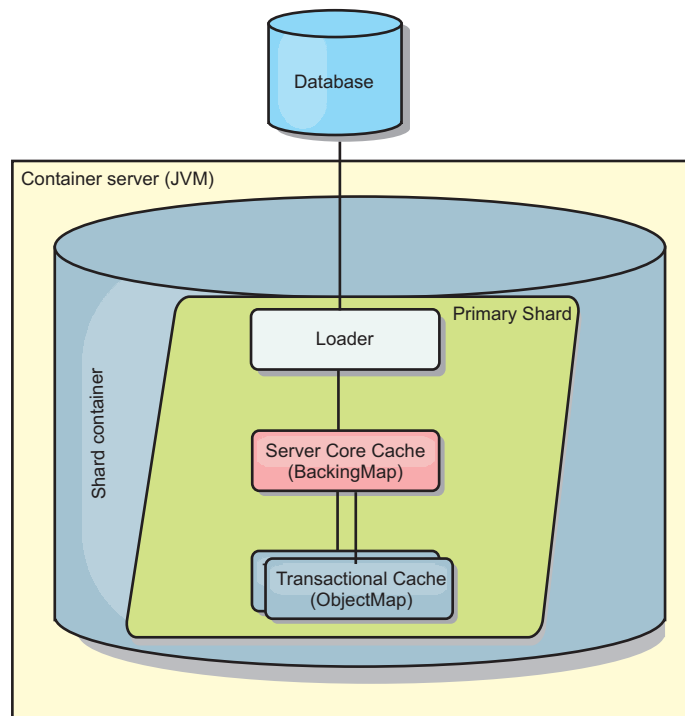


Figure 57. Loader

The loader can also use overqualified updates to avoid keeping database locks. By storing a version attribute in the cache value, the loader can see the before and

after image of the value as it is updated in the cache. This value can then be used when updating the database or back end to verify that the data has not been updated. A Loader can also be configured to preload the data grid when it is started. When partitioned, a Loader instance is associated with each partition. If the "Company" Map has ten partitions, there are ten Loader instances, one per primary partition. When the primary shard for the Map is activated, the preloadMap method for the loader is invoked synchronously or asynchronously which allows loading the map partition with data from the back-end to occur automatically. When invoked synchronously, all client transactions are blocked, preventing inconsistent access to the data grid. Alternatively, a client preloader can be used to load the entire data grid.

Two built-in loaders can greatly simplify integration with relational database back ends. The JPA loaders utilize the Object-Relational Mapping (ORM) capabilities of both the OpenJPA and Hibernate implementations of the Java Persistence API (JPA) specification. See "JPA Loaders" on page 69 for more information.

If you are using loaders in a multiple data center configuration, you must consider how revision data and cache consistency is maintained between the data grids. For more information, see "Loader considerations in a multi-master topology" on page 174.

Loader configuration

To add a Loader into the BackingMap configuration, you can use programmatic configuration or XML configuration. A loader has the following relationship with a backing map.

- A backing map can have only one loader.
- A client backing map (near cache) cannot have a loader.
- A loader definition can be applied to multiple backing maps, but each backing map has its own loader instance.

Data pre-loading and warm-up

In many scenarios that incorporate the use of a loader, you can prepare your data grid by pre-loading it with data.

When used as a complete cache, the data grid must hold all of the data and must be loaded before any clients can connect to it. When you are using a sparse cache, you can warm up the cache with data so that clients can have immediate access to data when they connect.

Two approaches exist for pre-loading data into the data grid: Using a Loader plug-in or using a client loader, as described in the following sections.

Loader plug-in

The loader plug-in is associated with each map and is responsible for synchronizing a single primary partition shard with the database. The preloadMap method of the loader plug-in is invoked automatically when a shard is activated. For example, if you have 100 partitions, 100 loader instances exist, each loading the data for its partition. When run synchronously, all clients are blocked until the preload has completed.

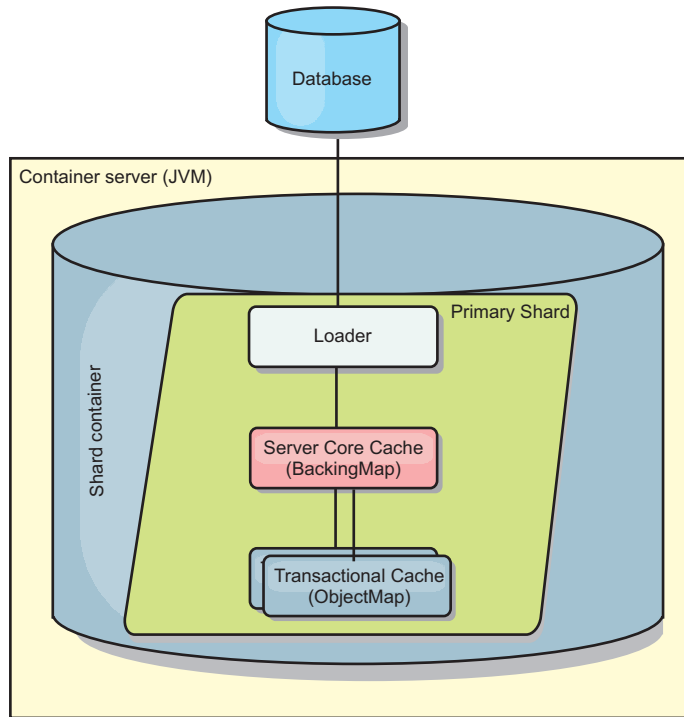


Figure 58. Loader plug-in

Client loader

A client loader is a pattern for using one or more clients to load the grid with data. Using multiple clients to load grid data can be effective when the partition scheme is not stored in the database. You can invoke client loaders manually or automatically when the data grid starts. Client loaders can optionally use the StateManager to set the state of the data grid to pre-load mode, so that clients are not able to access the grid while it is pre-loading the data. WebSphere eXtreme Scale includes a Java Persistence API (JPA)-based loader that you can use to automatically load the data grid with either the OpenJPA or Hibernate JPA providers. For more information about cache providers, see “JPA level 2 (L2) cache plug-in” on page 38.

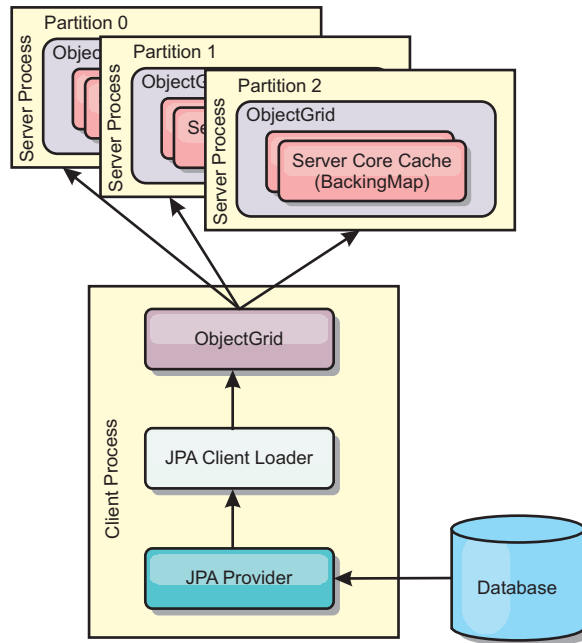


Figure 59. Client loader

Database synchronization techniques

When WebSphere eXtreme Scale is used as a cache, applications must be written to tolerate stale data if the database can be updated independently from an eXtreme Scale transaction. To serve as a synchronized in-memory database processing space, eXtreme Scale provides several ways of keeping the cache updated.

Database synchronization techniques

Periodic refresh

The cache can be automatically invalidated or updated periodically using the Java Persistence API (JPA) time-based database updater. The updater periodically queries the database using a JPA provider for any updates or inserts that have occurred since the previous update. Any changes identified are automatically invalidated or updated when used with a sparse cache. If used with a complete cache, the entries can be discovered and inserted into the cache. Entries are never removed from the cache.

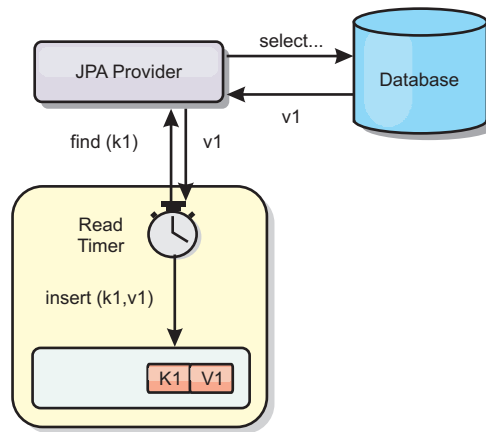


Figure 60. Periodic refresh

Eviction

Sparse caches can utilize eviction policies to automatically remove data from the cache without affecting the database. There are three built-in policies included in eXtreme Scale: time-to-live, least-recently-used, and least-frequently-used. All three policies can optionally evict data more aggressively as memory becomes constrained by enabling the memory-based eviction option.

Event-based invalidation

Sparse and complete caches can be invalidated or updated using an event generator such as Java Message Service (JMS). Invalidation using JMS can be manually tied to any process that updates the back-end using a database trigger. A JMS ObjectGridEventListener plug-in is provided in eXtreme Scale that can notify clients when the server cache has any changes. This can decrease the amount of time the client can see stale data.

Programmatic invalidation

The eXtreme Scale APIs allow manual interaction of the near and server cache using the `Session.beginNoWriteThrough()`, `ObjectMap.invalidate()` and `EntityManager.invalidate()` API methods. If a client or server process no longer needs a portion of the data, the invalidate methods can be used to remove data from the near or server cache. The `beginNoWriteThrough` method applies any `ObjectMap` or `EntityManager` operation to the local cache without calling the loader. If invoked from a client, the operation applies only to the near cache (the remote loader is not invoked). If invoked on the server, the operation applies only to the server core cache without invoking the loader.

Data invalidation

To remove stale cache data, you can use invalidation mechanisms.

Administrative invalidation

You can use the web console or the `xscmd` utility to invalidate data based on the key. You can filter the cache data with a regular expression and then invalidate the data based on the regular expression.

Event-based invalidation

Sparse and complete caches can be invalidated or updated using an event generator such as Java Message Service (JMS). Invalidation using JMS can be manually tied to any process that updates the back-end using a database trigger. A JMS ObjectGridEventListener plug-in is provided in eXtreme Scale that can notify clients when the server cache changes. This type of notification decreases the amount of time the client can see stale data.

Event-based invalidation normally consists of the following three components.

- **Event queue:** An event queue stores the data change events. It could be a JMS queue, a database, an in-memory FIFO queue, or any kind of manifest as long as it can manage the data change events.
- **Event publisher:** An event publisher publishes the data change events to the event queue. An event publisher is usually an application you create or an eXtreme Scale plug-in implementation. The event publisher knows when the data is changed or it changes the data itself. When a transaction commits, events are generated for the changed data and the event publisher publishes these events to the event queue.
- **Event consumer:** An event consumer consumes data change events. The event consumer is usually an application to ensure the target grid data is updated with the latest change from other grids. This event consumer interacts with the event queue to get the latest data change and applies the data changes in the target grid. The event consumers can use eXtreme Scale APIs to invalidate stale data or update the grid with the latest data.

For example, JMSObjectGridEventListener has an option for a client-server model, in which the event queue is a designated JMS destination. All server processes are event publishers. When a transaction commits, the server gets the data changes and publishes them to the designated JMS destination. All the client processes are event consumers. They receive the data changes from the designated JMS destination and apply the changes to the client's near cache.

See *Configuring Java Message Service (JMS)-based client synchronization* for more information.

Programmatic invalidation

The WebSphere eXtreme Scale APIs allow manual interaction of the near and server cache using the `Session.beginNoWriteThrough()`, `ObjectMap.invalidate()` and `EntityManager.invalidate()` API methods. If a client or server process no longer needs a portion of the data, the invalidate methods can be used to remove data from the near or server cache. The `beginNoWriteThrough` method applies any `ObjectMap` or `EntityManager` operation to the local cache without calling the loader. If invoked from a client, the operation applies only to the near cache (the remote loader is not invoked). If invoked on the server, the operation applies only to the server core cache without invoking the loader.

You can use programmatic invalidation with other techniques to determine when to invalidate the data. For example, this invalidation method uses event-based invalidation mechanisms to receive the data change events, and then uses APIs to invalidate the stale data.

8.6+

Near cache invalidation

If you are using a near cache, you can configure an asynchronous invalidation that is triggered each time an update, delete, invalidation operation is run against the data grid. Because the operation is asynchronous, you might still see stale data in the data grid.

To enable near cache invalidation, set the `nearCacheInvalidationEnabled` attribute on the backing map in the ObjectGrid descriptor XML file.

Indexing

Java

Use the `MapIndexPlugin` plug-in to build an index or several indexes on a `BackingMap` to support non-key data access.

Index types and configuration

The indexing feature is represented by the `MapIndexPlugin` plug-in or `Index` for short. The `Index` is a `BackingMap` plug-in. A `BackingMap` can have multiple `Index` plug-ins configured, as long as each one follows the `Index` configuration rules.

You can use the indexing feature to build one or more indexes on a `BackingMap`. An index is built from an attribute or a list of attributes of an object in the `BackingMap`. This feature provides a way for applications to find certain objects more quickly. With the indexing feature, applications can find objects with a specific value or within a range of values of indexed attributes.

Two types of indexing are possible: static and dynamic. With static indexing, you must configure the index plug-in on the `BackingMap` before initializing the `ObjectGrid` instance. You can do this configuration with XML or programmatic configuration of the `BackingMap`. Static indexing starts building an index during `ObjectGrid` initialization. The index is always synchronized with the `BackingMap` and ready for use. After the static indexing process starts, the maintenance of the index is part of the eXtreme Scale transaction management process. When transactions commit changes, these changes also update the static index, and index changes are rolled back if the transaction is rolled back.

With dynamic indexing, you can create an index on a `BackingMap` before or after the initialization of the containing `ObjectGrid` instance. Applications have life cycle control over the dynamic indexing process so that you can remove a dynamic index when it is no longer needed. When an application creates a dynamic index, the index might not be ready for immediate use because of the time it takes to complete the index building process. Because the amount of time depends upon the amount of data indexed, the `DynamicIndexCallback` interface is provided for applications that want to receive notifications when certain indexing events occur. These events include `ready`, `error`, and `destroy`. Applications can implement this callback interface and register with the dynamic indexing process.

8.6+ If a `BackingMap` has an index plug-in configured, you can obtain the application index proxy object from the corresponding `ObjectMap`. Calling the `getIndex` method on the `ObjectMap` and passing in the name of the index plug-in returns the index proxy object. You must cast the index proxy object to an appropriate application index interface, such as `MapIndex`, `MapRangeIndex`,

MapGlobalIndex, or a customized index interface. After obtaining the index proxy object, you can use methods defined in the application index interface to find cached objects.

The steps to use indexing are summarized in the following list:

- Add either static or dynamic index plug-ins into the BackingMap.
- Obtain an application index proxy object by issuing the getIndex method of the ObjectMap.
- Cast the index proxy object to an appropriate application index interface, such as MapIndex, MapRangeIndex, or a customized index interface.
- Use methods that are defined in application index interface to find cached objects.

8.6+ The HashIndex class is the built-in index plug-in implementation that can support the following built-in application index interfaces:

- MapIndex
- MapRangeIndex
- MapGlobalIndex

You also can create your own indexes. You can add HashIndex as either a static or dynamic index into the BackingMap, obtain either the MapIndex, MapRangeIndex, or MapGlobalIndex index proxy object, and use the index proxy object to find cached objects.

8.6+ **Global index**

Global index is an extension of the built-in HashIndex class that runs on shards in distributed, partitioned data grid environments. It tracks the location of indexed attributes and provides efficient ways to find partitions, keys, values, or entries using attributes in large, partitioned data grid environments.

If global index is enabled in the built-in HashIndex plug-in, then applications can cast an index proxy object to the MapGlobalIndex type, and use it to find data.

Default index

If you want to iterate through the keys in a local map, you can use the default index. This index does not require any configuration, but it must be used against the shard, using an agent or an ObjectGrid instance retrieved from the ShardEvents.shardActivated(ObjectGrid shard) method.

Data quality consideration

The results of index query methods only represent a snapshot of data at a point of time. No locks against data entries are obtained after the results return to the application. Application has to be aware that data updates may occur on a returned data set. For example, the application obtains the key of a cached object by running the findAll method of MapIndex. This returned key object is associated with a data entry in the cache. The application should be able to run the get method on ObjectMap to find an object by providing the key object. If another transaction removes the data object from the cache just before the get method is called, the returned result will be null.

Indexing performance considerations

One of the main objectives of the indexing feature is to improve overall BackingMap performance. If indexing is not used properly, the performance of the application might be compromised. Consider the following factors before using this feature.

- **The number of concurrent write transactions:** Index processing can occur every time a transaction writes data into a BackingMap. Performance degrades if many transactions are writing data into the map concurrently when an application attempts index query operations.
- **The size of the result set that is returned by a query operation:** As the size of the resultset increases, the query performance declines. Performance tends to degrade when the size of the result set is 15% or more of the BackingMap.
- **The number of indexes built over the same BackingMap:** Each index consumes system resources. As the number of the indexes built over the BackingMap increases, performance decreases.

The indexing function can improve BackingMap performance drastically. Ideal cases are when the BackingMap has mostly read operations, the query result set is of a small percentage of the BackingMap entries, and only few indexes are built over the BackingMap.

Planning multiple data center topologies

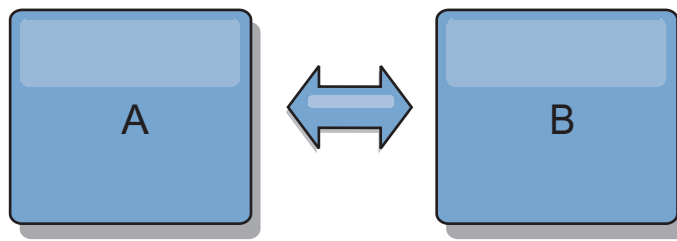
Using multi-master asynchronous replication, two or more data grids can become exact copies of each other. Each data grid is hosted in an independent catalog service domain, with its own catalog service, container servers, and a unique name. With multi-master asynchronous replication, you can use links to connect a collection of catalog service domains. The catalog service domains are then synchronized using replication over the links. You can construct almost any topology through the definition of links between the catalog service domains.

Topologies for multi-master replication

You have several different options when choosing the topology for your deployment that incorporates multi-master replication. Multi-master replication topologies can be implemented in the DataPower® XC10 Appliance by creating multiple collectives and linking them.

Links connecting catalog service domains

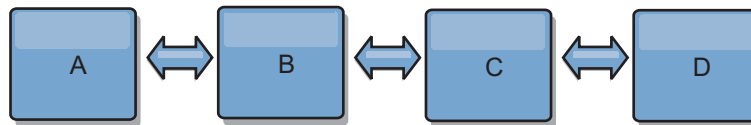
A replication data grid infrastructure is a connected graph of catalog service domains with bidirectional links among them. With a link, two catalog service domains can communicate data changes. For example, the simplest topology is a pair of catalog service domains with a single link between them. The catalog service domains are named alphabetically: A, B, C, and so on, from the left. A link can cross a wide area network (WAN), spanning large distances. Even if the link is interrupted, you can still change data in either catalog service domain. The topology reconciles changes when the link reconnects the catalog service domains. Links automatically try to reconnect if the network connection is interrupted.



After you set up the links, the product first tries to make every catalog service domain identical. Then, eXtreme Scale tries to maintain the identical conditions as changes occur in any catalog service domain. The goal is for each catalog service domain to be an exact mirror of every other catalog service domain connected by the links. The replication links between the catalog service domains help ensure that any changes made in one catalog service domain are copied to the other catalog service domains.

Line topologies

Although it is such a simple deployment, a line topology demonstrates some qualities of the links. First, it is not necessary for a catalog service domain to be connected directly to every other catalog service domain to receive changes. The catalog service domain B pulls changes from catalog service domain A. The catalog service domain C receives changes from catalog service domain A through catalog service domain B, which connects catalog service domains A and C. Similarly, catalog service domain D receives changes from the other catalog service domains through catalog service domain C. This ability spreads the load of distributing changes away from the source of the changes.



Notice that if catalog service domain C fails, the following actions would occur:

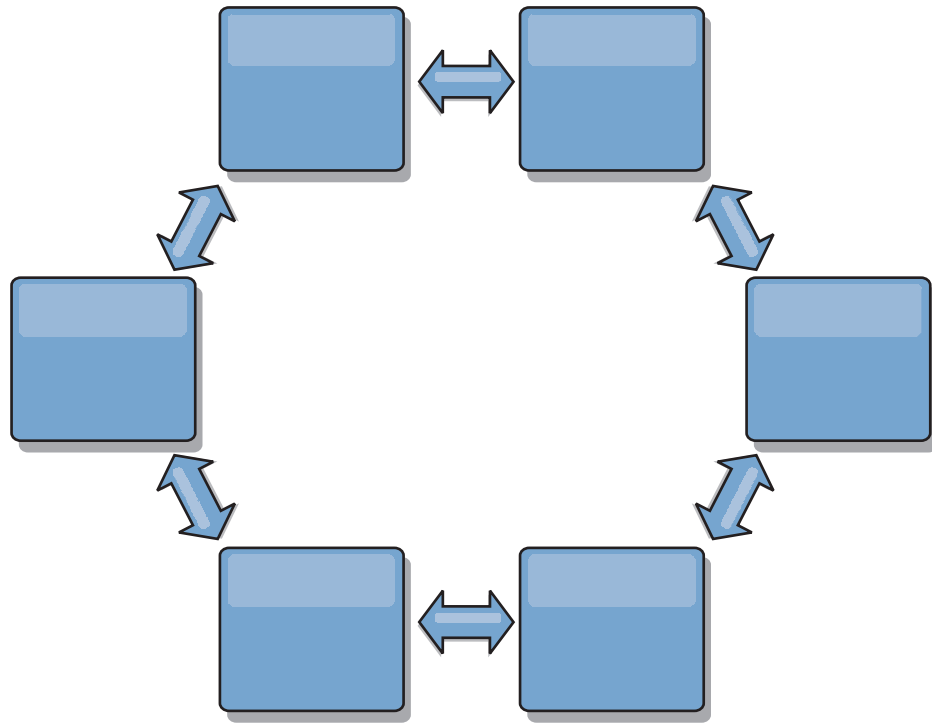
1. catalog service domain D would be orphaned until catalog service domain C was restarted
2. catalog service domain C would synchronize itself with catalog service domain B, which is a copy of catalog service domain A
3. catalog service domain D would use catalog service domain C to synchronize itself with changes on catalog service domain A and B. These changes initially occurred while catalog service domain D was orphaned (while catalog service domain C was down).

Ultimately, catalog service domains A, B, C, and D would all become identical to one other again.

Ring topologies

Ring topologies are an example of a more resilient topology. When a catalog service domain or a single link fails, the surviving catalog service domains can still obtain changes. The catalog service domains travel around the ring, away from the failure. Each catalog service domain has at most two links to other catalog service domains, no matter how large the ring topology. The latency to propagate changes can be large. Changes from a particular catalog service domain might need to

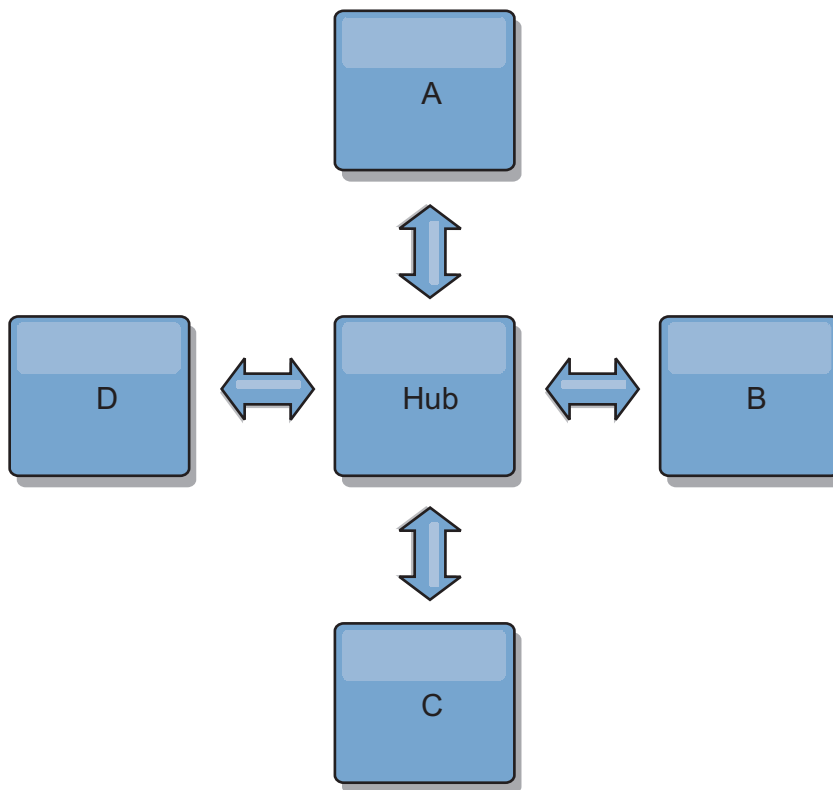
travel through several links before all the catalog service domains have the changes. A line topology has the same characteristic.



You can also deploy a more sophisticated ring topology, with a root catalog service domain at the center of the ring. The root catalog service domain functions as the central point of reconciliation. The other catalog service domains act as remote points of reconciliation for changes occurring in the root catalog service domain. The root catalog service domain can arbitrate changes among the catalog service domains. If a ring topology contains more than one ring around a root catalog service domain, the catalog service domain can only arbitrate changes among the innermost ring. However, the results of the arbitration spread throughout the catalog service domains in the other rings.

Hub-and-spoke topologies

With a hub-and-spoke topology, changes travel through a hub catalog service domain. Because the hub is the only intermediate catalog service domain that is specified, hub-and-spoke topologies have lower latency. The hub catalog service domain is connected to every spoke catalog service domain through a link. The hub distributes changes among the catalog service domains. The hub acts as a point of reconciliation for collisions. In an environment with a high update rate, the hub might require run on more hardware than the spokes to remain synchronized. WebSphere eXtreme Scale is designed to scale linearly, meaning you can make the hub larger, as needed, without difficulty. However, if the hub fails, then changes are not distributed until the hub restarts. Any changes on the spoke catalog service domains will be distributed after the hub is reconnected.



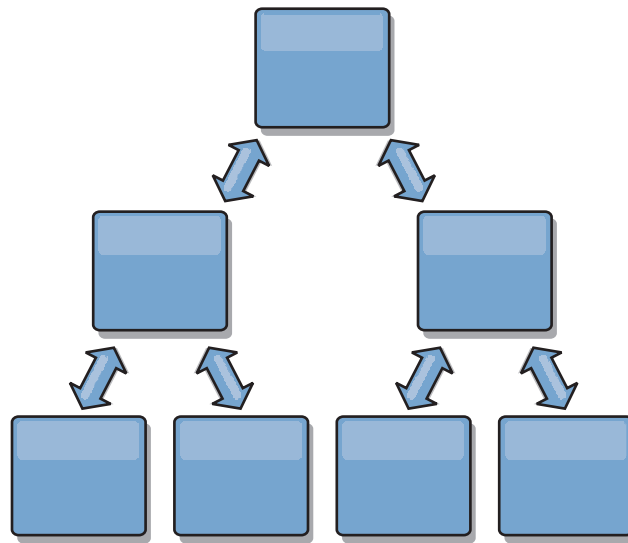
You can also use a strategy with fully replicated clients, a topology variation which uses a pair of servers that are running as a hub. Every client creates a self-contained single container data grid with a catalog in the client JVM. A client uses its data grid to connect to the hub catalog. This connection causes the client to synchronize with the hub as soon as the client obtains a connection to the hub.

Any changes made by the client are local to the client, and are replicated asynchronously to the hub. The hub acts as an arbitration catalog service domain, distributing changes to all connected clients. The fully replicated clients topology provides a reliable L2 cache for an object relational mapper, such as OpenJPA. Changes are distributed quickly among client JVMs through the hub. If the cache size can be contained within the available heap space, the topology is a reliable architecture for this style of L2.

Use multiple partitions to scale the hub catalog service domain on multiple JVMs, if necessary. Because all of the data still must fit in a single client JVM, multiple partitions increase the capacity of the hub to distribute and arbitrate changes. However, having multiple partitions does not change the capacity of a single catalog service domain.

Tree topologies

You can also use an acyclic directed tree. An acyclic tree has no cycles or loops, and a directed setup limits links to existing only between parents and children. This configuration is useful for topologies that have many catalog service domains. In these topologies, it is not practical to have a central hub that is connected to every possible spoke. This type of topology can also be useful when you must add child catalog service domains without updating the root catalog service domain.



A tree topology can still have a central point of reconciliation in the root catalog service domain. The second level can still function as a remote point of reconciliation for changes occurring in the catalog service domain beneath them. The root catalog service domain can arbitrate changes between the catalog service domains on the second level only. You can also use N-ary trees, each of which have N children at each level. Each catalog service domain connects out to n links.

Fully replicated clients

This topology variation involves a pair of servers that are running as a hub. Every client creates a self-contained single container data grid with a catalog in the client JVM. A client uses its data grid to connect to the hub catalog, causing the client to synchronize with the hub as soon as the client obtains a connection to the hub.

Any changes made by the client are local to the client, and are replicated asynchronously to the hub. The hub acts as an arbitration catalog service domain, distributing changes to all connected clients. The fully replicated clients topology provides a good L2 cache for an object relational mapper, such as OpenJPA. Changes are distributed quickly among client JVMs through the hub. As long as the cache size can be contained within the available heap space of the clients, this topology is a good architecture for this style of L2.

Use multiple partitions to scale the hub catalog service domain on multiple JVMs, if necessary. Because all of the data still must fit in a single client JVM, using multiple partitions increases the capacity of the hub to distribute and arbitrate changes, but it does not change the capacity of a single catalog service domain.

Configuration considerations for multi-master topologies

Consider the following issues when you are deciding whether and how to use multi-master replication topologies.

- **Map set requirements**

Map sets must have the following characteristics to replicate changes across catalog service domain links:

- The ObjectGrid name and map set name within a catalog service domain must match the ObjectGrid name and map set name of other catalog service domains. For example, ObjectGrid "og1" and map set "ms1" must be

configured in catalog service domain A and catalog service domain B to replicate the data in the map set between the catalog service domains.

- Is a FIXED_PARTITION data grid. PER_CONTAINER data grids cannot be replicated.
- Has the same number of partitions in each catalog service domain. The map set might or might not have the same number and types of replicas.
- Has the same data types being replicated in each catalog service domain.
- Contains the same maps and dynamic map templates in each catalog service domain.
- Does not use entity manager. A map set containing an entity map is not replicated across catalog service domains.
- Does not use write-behind caching support. A map set containing a map that is configured with write-behind support is not replicated across catalog service domains.

Any map sets with the preceding characteristics begin to replicate after the catalog service domains in the topology have been started.

- **Class loaders with multiple catalog service domains**

Catalog service domains must have access to all classes that are used as keys and values. Any dependencies must be reflected in all class paths for data grid container Java virtual machines (JVM) for all domains. If a CollisionArbiter plug-in retrieves the value for a cache entry, then the classes for the values must be present for the domain that is starting the arbiter.

Loader considerations in a multi-master topology

When you are using loaders in a multi-master topology, you must consider the possible collision and revision information maintenance challenges. The data grid maintains revision information about the items in the data grid so that collisions can be detected when other primary shards in the configuration write entries to the data grid. When entries are added from a loader, this revision information is not included and the entry takes on a new revision. Because the revision of the entry seems to be a new insert, a false collision could occur if another primary shard also changes this state or pulls the same information in from a loader.

Replication changes invoke the get method on the loader with a list of the keys that are not already in the data grid but are going to be changed during the replication transaction. When the replication occurs, these entries are collision entries. When the collisions are arbitrated and the revision is applied then a batch update is called on the loader to apply the changes to the database. All of the maps that were changed in the revision window are updated in the same transaction.

Preload conundrum

Consider a two data center topology with data center A and data center B. Both data centers have independent databases, but only data center A has a data grid that is running. When you establish a link between the data centers for a multi-master configuration, the data grids in data center A begin pushing data to the new data grids in data center B, causing a collision with every entry. Another major issue that occurs is with any data that is in the database in data center B but not in the database in data center A. These rows are not populated and arbitrated, resulting in inconsistencies that are not resolved.

Solution to the preload conundrum

Because data that resides only in the database cannot have revisions, you must always fully preload the data grid from the local database before establishing the multi-master link. Then, both data grids can revision and arbitrate the data, eventually reaching a consistent state.

Sparse cache conundrum

With a sparse cache, the application first attempts to find data in the data grid. If the data is not in the data grid, the data is searched for in the database using the loader. Entries are evicted from the data grid periodically to maintain a small cache size.

This cache type can be problematic in a multi-master configuration scenario because the entries within the data grid have revisioning metadata that help detect when collisions occur and which side has made changes. When links between the data centers are not working, one data center can update an entry and then eventually update the database and invalidate the entry in the data grid. When the link recovers, the data centers attempt to synchronize revisions with each other. However, because the database was updated and the data grid entry was invalidated, the change is lost from the perspective of the data center that went down. As a result, the two sides of the data grid are out of synch and are not consistent.

Solution to the sparse cache conundrum

Hub and spoke topology:

You can run the loader only in the hub of a hub and spoke topology, maintaining consistency of the data while scaling out the data grid. However, if you are considering this deployment, note that the loaders can allow the data grid to be partially loaded, meaning that an evictor has been configured. If the spokes of your configuration are sparse caches but have no loader, then any cache misses have no way to retrieve data from the database. Because of this restriction, you should use a fully populated cache topology with a hub and spoke configuration.

Invalidations and eviction

Invalidation creates inconsistency between the data grid and the database. Data can be removed from the data grid either programmatically or with eviction. When you develop your application, you must be aware that revision handling does not replicate changes that are invalidated, resulting in inconsistencies between primary shards.

Invalidation events are not cache state changes and do not result in replication. Any configured evictors run independently from other evictors in the configuration. For example, you might have one evictor configured for a memory threshold in one catalog service domain, but a different type of less aggressive evictor in your other linked catalog service domain. When data grid entries are removed due to the memory threshold policy, the entries in the other catalog service domain are not affected.

Database updates and data grid invalidation

Problems occur when you update the database directly in the background while calling the invalidation on the data grid for the updated entries in a multi-master configuration. This problem occurs because the data grid cannot replicate the change to the other primary shards until some type of cache access moves the entry into the data grid.

Multiple writers to a single logical database

When you are using a single database with multiple primary shards that are connected through a loader, transactional conflicts result. Your loader implementation must specially handle these types of scenarios.

Mirroring data using multi-master replication

You can configure independent databases that are connected to independent catalog service domains. In this configuration, the loader can push changes from one data center to the other data center.

Design considerations for multi-master replication

When implementing multi-master replication, you must consider aspects in your design such as: arbitration, linking, and performance.

Arbitration considerations in topology design

Change collisions might occur if the same records can be changed simultaneously in two places. Set up each catalog service domain to have about the same amount of processor, memory, network resources. You might observe that catalog service domains performing change collision handling (arbitration) use more resources than other catalog service domains. Collisions are detected automatically. They are handled with one of two mechanisms:

- **Default collision arbiter:** The default protocol is to use the changes from the lexically lowest named catalog service domain. For example, if catalog service domain A and B generate a conflict for a record, then the change from catalog service domain B is ignored. Catalog service domain A keeps its version and the record in catalog service domain B is changed to match the record from catalog service domain A. This behavior applies as well for applications where users or sessions are normally bound or have affinity with one of the data grids.
- **Custom collision arbiter:** Applications can provide a custom arbiter. When a catalog service domain detects a collision, it starts the arbiter. For information about developing a useful custom arbiter, see *Developing custom arbiters for multi-master replication*.

For topologies in which collisions are possible, consider implementing a hub-and-spoke topology or a tree topology. These two topologies are conducive to avoiding constant collisions, which can happen in the following scenarios:

1. Multiple catalog service domains experience a collision
2. Each catalog service domain handles the collision locally, producing revisions
3. The revisions collide, resulting in revisions of revisions

To avoid collisions, choose a specific catalog service domain, called an *arbitration catalog service domain* as the collision arbiter for a subset of catalog service domains. For example, a hub-and-spoke topology might use the hub as the collision handler. The spoke collision handler ignores any collisions that are detected by the spoke catalog service domains. The hub catalog service domain creates revisions, preventing unexpected collision revisions. The catalog service domain that is

assigned to handle collisions must link to all of the domains for which it is responsible for handling collisions. In a tree topology, any internal parent domains handle collisions for their immediate children. In contrast, if you use a ring topology, you cannot designate one catalog service domain in the ring as the arbiter.

The following table summarizes the arbitration approaches that are most compatible with various topologies.

Table 8. Arbitration approaches. This table states whether application arbitration is compatible with various technologies.

Topology	Application Arbitration?	Notes
A line of two catalog service domains	Yes	Choose one catalog service domain as the arbiter.
A line of three catalog service domains	Yes	The middle catalog service domain must be the arbiter. Think of the middle catalog service domain as the hub in a simple hub-and-spoke topology.
A line of more than three catalog service domains	No	Application arbitration is not supported.
A hub with N spokes	Yes	Hub with links to all spokes must be the arbitration catalog service domain.
A ring of N catalog service domains	No	Application arbitration is not supported.
An acyclic, directed tree (n-ary tree)	Yes	All root nodes must rate their direct descendants only.

Linking considerations in topology design

Ideally, a topology includes the minimum number of links while optimizing trade-offs among change latency, fault tolerance, and performance characteristics.

- **Change latency**

Change latency is determined by the number of intermediate catalog service domains a change must go through before arriving at a specific catalog service domain.

A topology has the best change latency when it eliminates intermediate catalog service domains by linking every catalog service domain to every other catalog service domain. However, a catalog service domain must perform replication work in proportion to its number of links. For large topologies, the sheer number of links to be defined can cause an administrative burden.

The speed at which a change is copied to other catalog service domains depends on additional factors, such as:

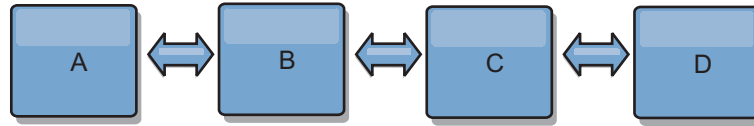
- Processor and network bandwidth on the source catalog service domain
- The number of intermediate catalog service domains and links between the source and target catalog service domain
- The processor and network resources available to the source, target, and intermediate catalog service domains

- **Fault tolerance**

Fault tolerance is determined by how many paths exist between two catalog service domains for change replication.

If you have only one link between a given pair of catalog service domains, a link failure disallows propagation of changes. Similarly, changes are not propagated between catalog service domains if any of the intermediate domains experiences link failure. Your topology could have a single link from one catalog service domain to another such that the link passes through intermediate domains. If so, then changes are not propagated if any of the intermediate catalog service domains is down.

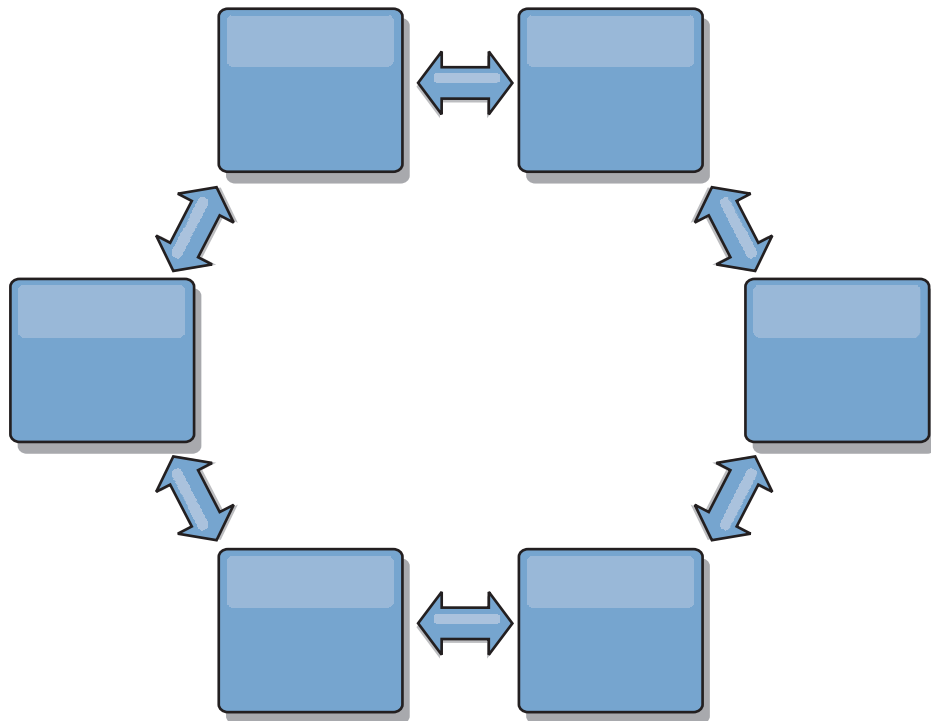
Consider the line topology with four catalog service domains A, B, C, and D:



If any of these conditions hold, Domain D does not see any changes from A:

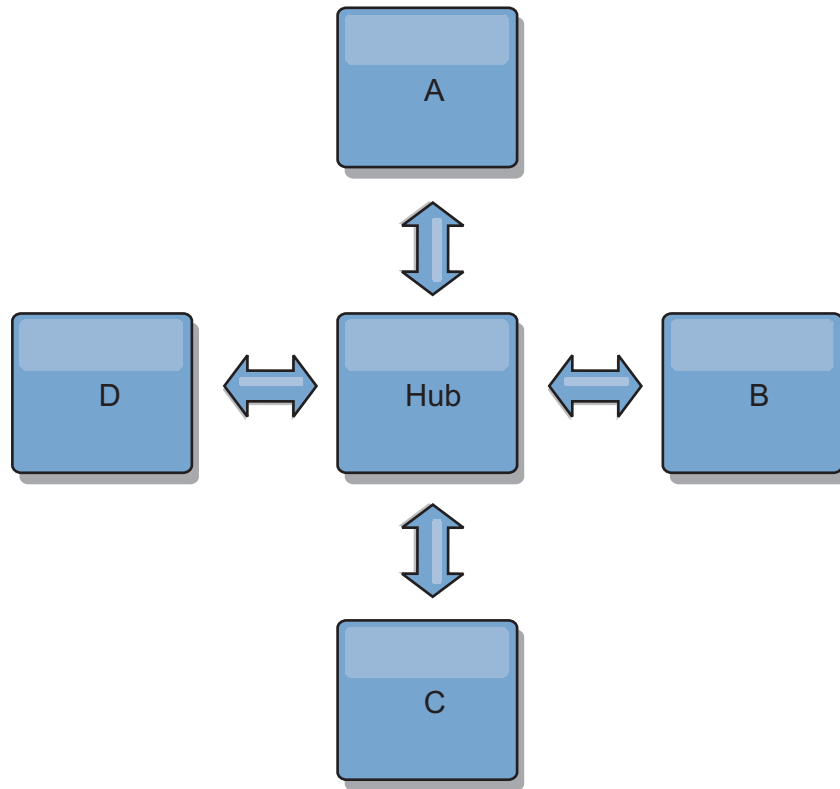
- Domain A is up and B is down
- Domains A and B are up and C is down
- The link between A and B is down
- The link between B and C is down
- The link between C and D is down

In contrast, with a ring topology, each catalog service domain can receive changes from either direction.



For example, if a given catalog service in your ring topology is down, then the two adjacent domains can still pull changes directly from each other.

All changes are propagated through the hub. Thus, as opposed to the line and ring topologies, the hub-and-spoke design is susceptible to break down if the hub fails.



A single catalog service domain is resilient to a certain amount of service loss. However, larger failures such as wide network outages or loss of links between physical data centers can disrupt any of your catalog service domains.

- **Linking and performance**

The number of links defined on a catalog service domain affects performance. More links use more resources and replication performance can drop as a result. The ability to retrieve changes for a domain A through other domains effectively offloads domain A from replicating its transactions everywhere. The change distribution load on a domain is limited by the number of links it uses, not how many domains are in the topology. This load property provides scalability, so the domains in the topology can share the burden of change distribution.

A catalog service domain can retrieve changes indirectly through other catalog service domains. Consider a line topology with five catalog service domains.

A <=> B <=> C <=> D <=> E

- A pulls changes from B, C, D, and E through B
- B pulls changes from A and C directly, and changes from D and E through C
- C pulls changes from B and D directly, and changes from A through B and E through D
- D pulls changes from C and E directly, and changes from A and B through C
- E pulls changes from D directly, and changes from A, B, and C through D

The distribution load on catalog service domains A and E is lowest, because they each have a link only to a single catalog service domain. Domains B, C, and D each have a link to two domains. Thus, the distribution load on domains B, C, and D is double the load on domains A and E. The workload depends on the number of links in each domain, not on the overall number of domains in the topology. Thus, the described distribution of loads would remain constant, even if the line contained 1000 domains.

Multi-master replication performance considerations

Take the following limitations into account when using multi-master replication topologies:

- **Change distribution tuning**, as discussed in the previous section.
- **Replication link performance** WebSphere eXtreme Scale creates a single TCP/IP socket between any pair of JVMs. All traffic between the JVMs occurs through the single socket, including traffic from multi-master replication. Catalog service domains are hosted on at least n container JVMs, providing at least n TCP links to peer catalog service domains. Thus, the catalog service domains with larger numbers of containers have higher replication performance levels. More containers require more processor and network resources.
- **TCP sliding window tuning and RFC 1323** RFC 1323 support on both ends of a link yields more data for a round trip. This support results in higher throughput, expanding the capacity of the window by a factor of about 16,000.

Recall that TCP sockets use a sliding window mechanism to control the flow of bulk data. This mechanism typically limits the socket to 64 KB for a round-trip interval. If the round-trip interval is 100 ms, then the bandwidth is limited to 640 KB/second without additional tuning. Fully using the bandwidth available on a link might require tuning that is specific to an operating system. Most operating systems include tuning parameters, including RFC 1323 options, to enhance throughput over high-latency links.

Several factors can affect replication performance:

- The speed at which eXtreme Scale retrieves changes.
- The speed at which eXtreme Scale can service retrieve replication requests.
- The sliding window capacity.
- With network buffer tuning on both sides of a link, eXtreme Scale retrieves changes over the socket efficiently.
- **Object Serialization** All data must be serializable. If a catalog service domain is not using COPY_TO_BYTES, then the catalog service domain must use Java serialization or ObjectTransformers to optimize serialization performance.
- **Compression** WebSphere eXtreme Scale compresses all data sent between catalog service domains by default. Disabling compression is not currently available.
- **Memory tuning** The memory usage for a multi-master replication topology is largely independent of the number of catalog service domains in the topology. Multi-master replication adds a fixed amount of processing per Map entry to handle versioning. Each container also tracks a fixed amount of data for each catalog service domain in the topology. A topology with two catalog service domains uses approximately the same memory as a topology with 50 catalog service domains. WebSphere eXtreme Scale does not use replay logs or similar queues in its implementation. Thus, there is no recovery structure ready in the case that a replication link is unavailable for a substantial period and later restarts.

Interoperability with other products

You can integrate WebSphere eXtreme Scale with other products, such as WebSphere Application Server and WebSphere Application Server Community Edition.

WebSphere Application Server

You can integrate WebSphere Application Server into various aspects of your WebSphere eXtreme Scale configuration. You can deploy data grid applications and use WebSphere Application Server to host container and catalog servers. Or, you can use a mixed environment that has WebSphere eXtreme Scale Client installed in the WebSphere Application Server environment with stand-alone catalog and container servers. You can also use WebSphere Application Server security in your WebSphere eXtreme Scale environment.

WebSphere Business Process Management and Connectivity products

WebSphere Business Process Management and Connectivity products, including WebSphere Integration Developer, WebSphere Enterprise Service Bus, and WebSphere Process Server, integrate with back end systems, such as CICS®, web services, databases, or JMS topics and queues. You can add WebSphere eXtreme Scale to the configuration to cache the output of these back end systems, increasing the overall performance of your configuration.

WebSphere Commerce

WebSphere Commerce can leverage WebSphere eXtreme Scale caching as a replacement to dynamic cache. By eliminating duplicate dynamic cache entries and the frequent invalidation processing necessary to keep the cache synchronized during high stress situations, you can improve performance, scaling, and high availability.

WebSphere Portal

You can persist HTTP sessions from WebSphere Portal into a data grid in WebSphere eXtreme Scale. In addition, IBM Web Content Manager in IBM WebSphere Portal can use dynamic cache instances to store rendered content that is retrieved from Web Content Manager when advanced caching is enabled. WebSphere eXtreme Scale offers an implementation of dynamic cache that stores cached content in an elastic data grid instead of using the default dynamic cache implementation.

WebSphere Application Server Community Edition

WebSphere Application Server Community Edition can share session state, but not in an efficient, scalable manner. WebSphere eXtreme Scale provides a high performance, distributed persistence layer that can be used to replicate state, but does not readily integrate with any application server outside of WebSphere Application Server. You can integrate these two products to provide a scalable session management solution.

WebSphere Real Time

With support for WebSphere Real Time, the industry-leading real-time Java offering, WebSphere eXtreme Scale enables Extreme Transaction Processing (XTP) applications to have more consistent and predictable response times.

Monitoring

WebSphere eXtreme Scale can be monitored using several popular enterprise monitoring solutions. Plug-in agents are included for IBM Tivoli Monitoring and Hyperic HQ, which monitor WebSphere eXtreme Scale using publicly accessible management beans. CA Wily Introscope uses Java method instrumentation to capture statistics.

.NET

8.6+

Microsoft Visual Studio, IIS, and .NET environments

For more information about supported Microsoft Visual Studio, IIS, and .NET environments, see Microsoft .NET considerations.

Chapter 3. Scenarios



Scenarios include real-world information to build a complete picture. Complete a scenario to understand new concepts or to accomplish common WebSphere eXtreme Scale tasks.

Scenario: Configuring an enterprise data grid

Configure an enterprise data grid when you want both Java and .NET applications to connect to the same data grid.

Before you begin

- Install the product. You must install both the server runtime and the clients. For clients, you can use both Java and .NET clients. For more information, see *Installing*.
- If you are upgrading from a previous release, you must have all of your container and catalog servers at the same release level. For more information, see *Upgrading and migrating WebSphere eXtreme Scale*.

Enterprise data grid overview

Enterprise data grids use the eXtremeIO transport mechanism and a new serialization format. With the new transport and serialization format, you can connect both Java and .NET clients to the same data grid.

With the enterprise data grid, you can create multiple types of applications, written in different programming languages, to access the same objects in the data grid. In prior releases, data grid applications had to be written in the Java programming language only. With the enterprise data grid function, you can write .NET applications that can create, retrieve, update, and delete objects from the same data grid as the Java application.

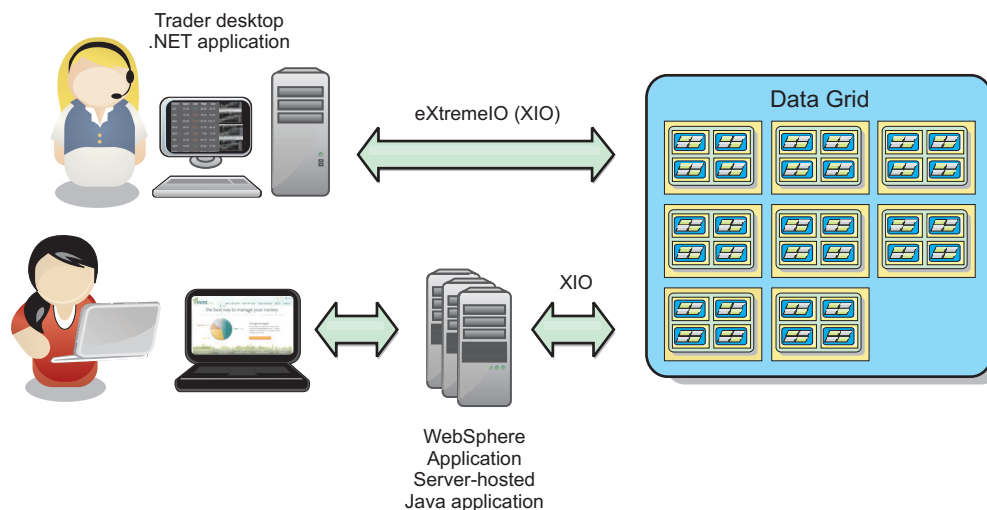


Figure 61. Enterprise data grid high-level overview

Object updates across different applications

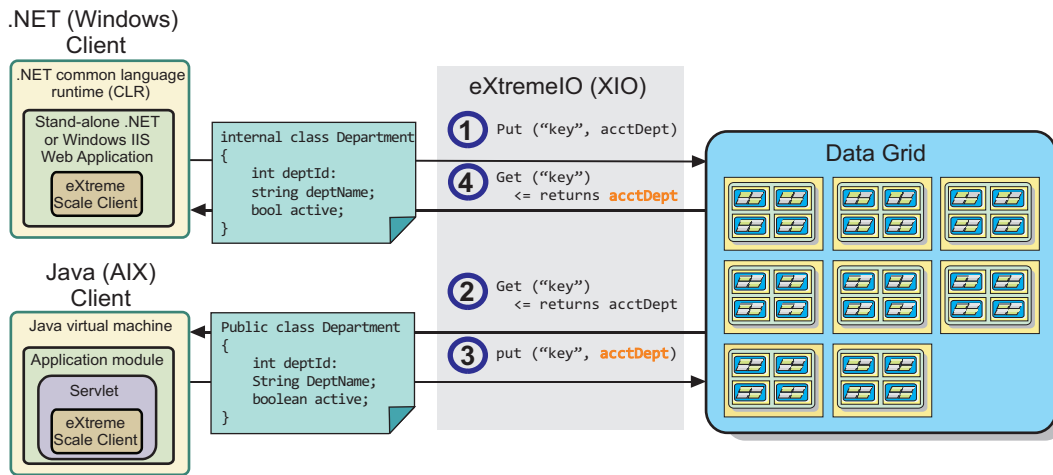


Figure 62. Enterprise data grid object update flow

1. The .NET client saves data in its format to the data grid.
2. The data is stored in a universal format, so that when the Java client requests this data it can be converted to Java format.
3. The Java client updates and re-saves the data.
4. The .NET client accesses the updated data, during which the data is converted to .NET format.

Transport mechanism

eXtremeIO (XIO) is a cross-platform transport protocol. XIO replaces the Java-bound Object Request Broker (ORB). With the ORB, WebSphere eXtreme Scale is bound to Java native client applications. XIO is a customized transport mechanism that is specifically targeted for data caching and enables client applications that are in different programming languages to connect to the data grid.

Serialization format

eXtreme data format (XDF) is a cross-platform serialization format. XDF replaces Java serialization on maps that have a copyMode attribute value of COPY_TO_BYTES in the ObjectGrid descriptor XML file. With XDF, performance is faster and data is more compact. In addition, the introduction of XDF enables client applications that are in different programming languages to connect to the same data grid.

Configuring IBM eXtremeIO (XIO)

IBM eXtremeIO (XIO) is a transport mechanism that replaces the Object Request Broker (ORB).

Before you begin

- If Version 8.6.0.2 is installed on your servers that are running XIO and you have COPY_TO_BYTES configured on the data grids, your WebSphere eXtreme Scale Client installations must also be at Version 8.6.0.2 or later.

- **8.6** To configure XIO, all of your catalog and container servers must be at the Version 8.6 release level. For more information, see Updating eXtreme Scale servers.

About this task

8.6+ You can configure XIO for all the container servers in your catalog service domain by enabling XIO in the catalog servers. The container servers discover the transport type of the catalog server and use that transport type.

Procedure

8.6+ How you enable XIO depends on the type of servers you are using:

- Enable XIO on your stand-alone catalog servers.
XIO is enabled by default when you start your catalog server with the **startXsServer** command. For more information, see Starting container servers that use the IBM eXtremeIO (XIO) transport.
- Enable XIO on your servers that are running in WebSphere Application Server.
You can enable XIO on your catalog service domain in the WebSphere Application Server administrative console. Click **System administration > WebSphere eXtreme Scale > Catalog service domains > catalog_service_domain**. Select **Enable IBM eXtremeIO (XIO) communication**. Apply your changes. For more information, see Configuring the catalog service in WebSphere Application Server.
- Enable XIO on your servers that run in the Liberty profile.
To enable XIO in a Liberty profile server, set transport attribute to XIO in your server.xml file. For example, see the highlighted property in the following code example:

```
<featureManager>
...
  <feature>eXtremeScale.server-1.1</feature>
</featureManager>

<xsServer isCatalog="true" transport="XIO" listenerPort="2809" ... />
```

Attention: The server must be a catalog server, and therefore, isCatalog must be set to true when you configure XIO. The listenerPort setting is not required; however, XIO can recognize this port if you enable it. If you do not enable XIO, then the ORB is used on that port instead.

Next, run the **start** command to start your Liberty profile servers. For more information, see Starting and stopping servers in the Liberty profile.

8.6+ You can use command-line arguments and server properties to configure XIO behavior:

- Optional: Update the server properties file for each container server in the configuration to enable XIO properties. After you decide on the properties that you want to set, you can set the values in the server properties file or programmatically with the ServerProperties interface. For more information about the properties you can set, see “Tuning IBM eXtremeIO (XIO)” on page 192.

8.6+

Results

The servers that you configured use the XIO transport. To verify that the configuration is correct, see [Displaying the transport type of your catalog service domain](#).

What to do next

You can also use IBM eXtremeMemory to help you avoid garbage collection pauses, leading to more constant performance and predicable response times. For more information, see [Configuring IBM eXtremeMemory](#).

Configuring data grids to use eXtreme data format (XDF)

If you are using an enterprise data grid, you must enable XDF so that both Java and .NET can access the same data grid objects. Use XDF to serialize and store keys and values in the data grid in a language-independent format.

Before you begin

Enable IBM eXtremeIO (XIO) in the environment. For more information, see [“Configuring IBM eXtremeIO \(XIO\)”](#) on page 184.

About this task

Enable eXtreme Data Format (XDF) to store serialized objects in a language independent manner. XDF is now the default serialization technology that is used when you are running XIO and have a map copy mode that is set to COPY_TO_BYTES. When you enable this feature, Java and C# objects can share data in the same data grid. You can set XDF mode for installations of WebSphere eXtreme Scale in a stand-alone environment and for installations of WebSphere eXtreme Scale within a WebSphere Application Server environment.

When you use XDF, you get the following benefits:

- Serialization of the data for sharing between Java, and C#/.NET applications.
- Indexing data on the server without requiring the user classes to be present, if field access is used.
- Automatic versioning of your classes so you can augment the class definitions when you add applications that require newer versions of the files. Older versions of the data can be used by taking advantage of the Mergable interface.
- Partitioning of the data with annotations in Java and C# to consistently partition from the application.

Restriction: You cannot use XDF if your data grids have EntityMetadata defined.

Procedure

In the ObjectGrid descriptor XML file, set the **CopyMode** attribute to XDF in the backingMap element of the ObjectGrid descriptor XML file.

```
<backingMap name="Employee" lockStrategy="PESSIMISTIC" copyMode="COPY_TO_BYTES">
```

What to do next

Develop applications that can share data. For more information, see [“Developing enterprise data grid applications”](#) on page 187.

Developing enterprise data grid applications

After you configure IBM eXtremeIO, you can write applications that access the enterprise data grid.

Before you begin

- Set up your development environment and view the API documentation. For more information, see *Getting started with developing applications*.
- You must have existing Java or .NET applications that access the data grid. For more information about getting started with writing applications, see *Getting started tutorial module 2: Create a client application*.

Class evolution

eXtreme data format (XDF) allows for class evolution. With class evolution, you can evolve the class definitions that are used in the data grid without affecting older applications that are using previous versions of the class. These older classes are accessing data in the same map as the new applications.

Overview

Class evolution is a further extension of the identification of classes and fields that determine whether two types are compatible enough to function together. Classes can function together when one of the classes has fewer fields than the other class. The following user scenarios are designed into the XDF implementation :

Multiple versions of the same object class

In this scenario, you have a map in a sales application that is used tracking customers. This map has two different interfaces. One interface is for the web purchases. The second interface is for the phone purchases. In version 2 of this sales application, you decide to give discounts to web shoppers based on their purchasing habits. This discount is stored with the Customer object. The phone sales employees are still using version 1 of the application, which is unaware of the new discount field in the web version. You want Customer objects from version 2 of the application to work with Customer objects that were created with the version 1 application and vice versa.

Multiple versions of a different object class

In this scenario, you have a sales application that is written in Java that keeps a map of Customer objects. You also have another application that is written in C# and is used to manage the inventory in the warehouse and ship goods to customers. These classes are currently compatible based on the names of the classes, fields, and types. In your Java sales application, you want to add an option to the Customer record to associate the sales person with a customer account. However, you do not want to update the warehouse application to store this field because it is not needed in the warehouse.

Multiple incompatible versions of the same class

In this scenario, your sales and inventory applications both contain a Customer object. The inventory application uses an ID field that is a string and the sales application uses an ID field that is an integer. These types are not compatible. As a result, the objects are probably not stored in the same map. The objects must be handled by the XDF serialization and treated as two distinct types. While this scenario is not really class evolution, it is a consideration that must be part of your overall application design.

Determination for evolution

XDF attempts to evolve a class when the class names match and the field names do not have conflicting types. Using the `ClassAlias` and `FieldAlias` annotations are useful when you are trying to match classes between C# and Java applications where the names of the classes or fields are slightly different. You can put these annotations on either the Java and C# application, or both. However, the lookup for the class in the Java application can be less efficient than defining the `ClassAlias` on the C# application. For more information about the `ClassAlias` and `FieldAlias` annotations, see “`ClassAlias` and `FieldAlias` annotations” on page 189

The effect of missing fields in serialized data

The constructor of the class is not invoked during deserialization, so any missing fields have a default that is assigned to it based on the language. The application that is adding new fields must be able to detect the missing fields and react when an older version of class is retrieved.

Updating the data is the only way for older applications to keep the newer fields

An application might run a fetch operation and update the map with an older version of the class that is missing some fields in the serialized value from the client. The server then merges the values on the server and determines whether any fields in the original version are merged into the new record. If an application runs a fetch operation, and then removes and inserts an entry, the fields from the original value are lost.

Merging capabilities

Objects within an array or collection are not merged by XDF. It is not always clear whether an update to an array or collection is intended to change the elements of that array or the type. If a merge occurs based on positioning, when an entry in the array is moved, XDF might merge fields that are not intended to be associated. As a result, XDF does not attempt to merge the contents of arrays or collections. However, if you add an array in a newer version of a class definition, the array gets merged back into the previous version of the class.

Defining `ClassAlias` and `FieldAlias` annotations to correlate Java and .NET classes

Use `ClassAlias` and `FieldAlias` annotations to enable sharing of data grid data between your Java and .NET classes.

Before you begin

- You must have IBM eXtremeIO configured. For more information, see “`Configuring IBM eXtremeIO (XIO)`” on page 184.
- Your `copyMode` attribute in your ObjectGrid descriptor XML file must be set to `COPY_TO_BYTES`. For more information, see “`Configuring data grids to use eXtreme data format (XDF)`” on page 186.

About this task

You might consider using `ClassAlias` and `FieldAlias` annotations if you have an existing Java class and want to create a corresponding C# class. In this scenario, you can add the annotations to your C# class that include the Java class name. For

more information about the `ClassAlias` and `FieldAlias` annotations, see “`ClassAlias` and `FieldAlias` annotations.”

Procedure

Use `ClassAlias` and `FieldAlias` annotations to correlate objects between a Java class and a C# class.

```
Java
.NET
@ClassAlias("Employee")
class com.company.department.Employee {
    @FieldAlias("id")
    int myId;
    String name;
}
```

Figure 63. Java example with `ClassAlias` and `FieldAlias` annotations

```
[ ClassAlias( "Employee" ) ]
class Com.MyCompany.Employee {
    [ FieldAlias("id" ) ]
    int identifier;
    string name;
}
```

Figure 64. .NET example with `ClassAlias` and `FieldAlias` attributes

ClassAlias and FieldAlias annotations:

Use `ClassAlias` and `FieldAlias` annotations to enable sharing of data grid data between classes. You can either share data between two Java classes or a Java and a .NET class.

If you define two classes with the same name and fields, the data grid data is automatically shared between the classes. For example, if you have a `Customer1` class in your Java application, and a `Customer1` class in your .NET application that has the same fields, the data is shared between the classes. This example assumes that the class name also includes the class qualifier, which is also the package name in Java and namespace name in C#. The package name and namespace name are automatically shared because the namespace and package names match. See the following example, where both names are case insensitive:

```
Java:
package com.mycompany.app
public class SampleClass {
    int field1;
    String field2;
}

C#
namespace Com.MyCompany.App
public class SampleClass {
    int field1;
    string field2;
}
```

However, you can also correlate data between classes that have different names. To correlate data to be stored in the data grid between different classes with different names, use `ClassAlias` or `FieldAlias` annotations.

Between two Java applications: You can define two different classes with different names in separate Java application environments. By marking the classes with the same `ClassAlias` annotation, and all fields and field types are matched between these two classes. The classes get correlated with the same class type ID even though they have the different class names. The same class type ID and the metadata can then be reused between the classes in the different Java application run times.

Between a Java application and a .NET application: You can use similar annotations in your C# application to correlate the C# class with a Java class. The `ClassAlias` attributes that are defined for the class C# and fields are matched to a Java class with the same `ClassAlias` annotation.

Mapping keys to partitions with `PartitionKey` annotations

A `PartitionKey` alias is used to identify the fields or attributes on which a hash code calculation is run to determine the partition to which data is saved. The `PartitionKey` annotation is only valid on key attributes.

Before you begin

You must be using eXtreme Data Format (XDF). For more information, see “Configuring data grids to use eXtreme data format (XDF)” on page 186.

About this task

You set a `PartitionKey` alias to ensure that multiple classes save data to the same partition. For example, if you set the `PartitionKey` value to be the `departmentID` key, employee records are collocated on the same partition.

The `PartitionableKey` interface is the existing Java interface and has precedence over the `PartitionableKey` annotation in C#.

Procedure

- **Java** Define `PartitionKey` annotations on a field in a Java application.

```
class Employee {
    int empId;

    @PartitionKey(order = 0)
    int deptId;
}
```

You can set `PartitionKey` annotations on multiple keys, or you can set the `PartitionKey` alias on a class. For more examples of how to set `PartitionKey` annotations in Java applications, see Java API documentation: Annotation Type `PartitionKeys`.

- **.NET** Define `PartitionKey` attributes on a field in a .NET application.


```

class Employee {
    int empId;

    [PartitionKey]
    int deptId;
}

```

You can set also PartitionKey attributes on .NET classes. For more information, see .NET API documentation: PartitionKeyAttribute Class.

Java and C# data type equivalents

When you develop enterprise data grid applications, data types between your Java and C# applications must be compatible.

Table 9. Data type equivalents between Java and C#

Java type	C# type
boolean	bool
java.lang.Boolean	bool?
byte	sbyte or byte
java.lang.Byte	sbyte?
short	short?, ushort
java.lang.Short	short?, ushort?
int	int, uint, ushort
java.lang.Integer	int?, uint?
long	long, ulong, uint
java.lang.Long	long?, ulong?, uint?
short or int	ushort
java.lang.Short or java.lang.Integer	ushort?
int or long	uint
java.lang.Integer or java.lang.Long	uint?
long or BigInteger	ulong
java.lang.Long or java.lang.BigInteger	ulong?
char, java.lang.Character	char
java.lang.Character	char?
float, java.lang.Float	float
java.lang.Foat	float?
double	double
java.lang.Double	double?
java.math.BigDecimal	decimal or decimal?
java.math.BigInteger	decimal, long or ulong?
java.lang.String	string
java.util.Date, java.util.Calendar	System.DateTime
java.util.Date(rounding), java.util.Calendar(rounding)	System.DateTime
java.util.ArrayList	System.Collections.ArrayList, System.Collections.Generic.List

Table 9. Data type equivalents between Java and C# (continued)

Java type	C# type
java.util.HashMap	System.Collections.Generic.Dictionary, System.Collections.Hashtable
java.util.LinkedList	System.Collections.Generic.LinkedList
java.util.ArrayList, java.util.Vector	System.Collections.Generic.List
java.util.Stack	System.Collections.Generic.Stack
java.util.Vector	System.Collections.ArrayList, System.Collections.Generic.List

Starting stand-alone servers (XIO)

When you are running a stand-alone configuration, the environment is comprised of catalog servers, container servers, and client processes. WebSphere eXtreme Scale servers can also be embedded within existing Java applications by using the embedded server API. You must manually configure and start these processes.

Before you begin

You can start WebSphere eXtreme Scale servers in an environment that does not have WebSphere Application Server installed. If you are using WebSphere Application Server, see *Configuring WebSphere eXtreme Scale with WebSphere Application Server*.

Tuning IBM eXtremeIO (XIO)

You can use XIO server properties to tune the behavior of the XIO transport in the data grid.

Server properties for tuning XIO

You can set the following properties in the server properties file:

maxXIONetworkThreads

Sets the maximum number of threads to allocate in the eXtremeIO transport network thread pool.

Default:256

minXIONetworkThreads

Sets the minimum number of threads to allocate in the eXtremeIO transport network thread pool.

Default:1

maxXIOWorkerThreads

Sets the maximum number of threads to allocate in the eXtremeIO transport request processing thread pool.

Default:256

minXIOWorkerThreads

Sets the minimum number of threads to allocate in the eXtremeIO transport request processing thread pool.

Default:1

8.6+ transport

Specifies the type of transport to use for all the servers in the catalog service domain. You can set the value to XIIO or ORB.

When you use the **start0gServer** or **startXsServer** commands, you do not need to set this property. The script overrides this property. However, if you start servers with another method, the value of this property is used.

This property applies to the catalog service only.

If you have both the **-transport** parameter on the start script and the **transport** server property defined on a catalog server, the value of the **-transport** parameter is used.

8.6+ xioTimeout

Sets the timeout for server requests that are using the IBM eXtremeIO (XIO) transport in seconds. The value can be set to any value greater than or equal to one second.

Default: 30 seconds

Scenario: Securing your data grid in eXtreme Scale

WebSphere eXtreme Scale data grids store information that is sensitive and must be protected.

Before you begin

- Install the product. You must install both the server runtime and the clients. For clients, you can use both Java and .NET clients. For more information, see *Installing*.
- If you are upgrading from a previous release, you must have all of your container and catalog servers at the same release level. For more information, see *Upgrading and migrating WebSphere eXtreme Scale*.

About this task

For a secure deployment, use several layers of protection for optimal security. The first element of protection is the use of firewalls to segment the network. The standard tiered model for web applications is comprised of web clients, a presentation tier of HTTP servers, an application tier comprised of application servers, a data tier, and a storage tier.

eXtreme Scale data grid servers are deployed as part of the data tier. Standard practice is to put the presentation layer servers in a demilitarized zone (DMZ) protected by one firewall, and to put the application, data, and storage tiers in network segments protected by additional firewalls. Do not deploy eXtreme Scale servers in a DMZ. eXtreme Scale servers must be protected as all elements of the data tier are, according to standard industry practice.

However, for optimal protection against security threats, use an in-depth defense mechanism, where a number of additional measures protect eXtreme Scale operation and the data that is stored in the data grid. These additional measures not only help in defending against external threats, but also prevent unauthorized data access by employees and contractors who might have access to network segments in which the eXtreme Scale servers reside.

Use the following end-to-end steps to configure security in WebSphere eXtreme Scale, whether you have stand-alone servers, the Liberty profile, the OSGi framework, or WebSphere Application Server installed in your environment:

Authenticating eXtreme Scale connections between servers

Connections between servers must be authenticated to prevent an unauthorized server from accessing grid data.

What to do next

“Authenticating requests from clients to servers” on page 197

Authenticating eXtreme Scale server connections in stand-alone environments

Connections between eXtreme Scale servers must be authenticated to prevent an unauthorized server from accessing the data grid.

About this task

The following settings in the `server.properties` file determine how servers authenticate to one another:

- `securityEnabled=true`
- `secureTokenManagerType=autoSecret`
- `authenticationSecret=OurGridServersExampleSecret`

All of the eXtreme Scale servers in a domain, as well as all of the servers in any linked domains, must use the same values for these three properties in the `server.properties` file, or communication fails. For more information about how to specify these properties in the server properties file, see `Server properties file`.

Procedure

1. Enable server to server authentication. Set the `securityEnabled` property to `true`; for example:

```
securityEnabled=true
```

The default value for this property is `false`.

2. Establish a secure server configuration.

The `secureTokenManagerType` is a property that you define in the `Server Properties` file.

One `secureTokenManagerType` that you can use for a secure configuration is `autoSecret`, which performs token encryption and signing using keys derived from the `authenticationSecret` property. Secure tokens are used in server-to-server authentication and also for client single sign-on tokens. A value of `none` for `secureTokenManagerType` is not secure because this setting prevents the creation of encrypted tokens.

You can also specify a setting of `secureTokenManagerType=default`. However, this option requires that you set up of a key store and related artifacts.

3. Specify a long string value for `authenticationSecret` (note: one word) that is difficult for others to guess. You can encode this value using the `FilePasswordEncoder` utility. For more information, see “Storing security artifacts for authorized users” on page 214. Do not use the `ObjectGridDefaultSecret` property, which is the value that is used in the `sampleServer.properties` file.

Results

When you start a stand-alone eXtreme Scale server, specify the name of the properties file is on the command line. By specifying the server properties file, the

authentication properties that you added are loaded when the server starts. For more information, see [Starting secure servers in a stand-alone environment](#).

What to do next

“Authenticating client requests in stand-alone environments” on page 197

Authenticating eXtreme Scale server connections in the Liberty profile

Connections between eXtreme Scale servers in the Liberty profile must be authenticated to prevent an unauthorized server from accessing the data grid.

About this task

The following settings in the `server.properties` file determine how servers authenticate to one another:

- `securityEnabled=true`
- `secureTokenManagerType=autoSecret`
- `authenticationSecret=OurGridServersExampleSecret`

All of the eXtreme Scale servers in a domain, as well as all of the servers in any linked domains, must use the same values for these properties in the `server.properties` file, or communication fails.

Procedure

1. Enable server to server authentication. Set the `securityEnable` property to `true`; for example:
`securityEnabled=true`

The default value for this property is `false`.

2. Establish a secure server configuration. One `secureTokenManagerType` that can be used for a secure configuration is `autoSecret`, which performs token encryption and signing using keys derived from the `authenticationSecret`. Secure tokens are used in server to server authentication and also for client single sign-on tokens. A value of `none` for `secureTokenManagerType` is not secure because this setting prevents the creation of encrypted tokens.
You can also specify a setting of `secureTokenManagerType=default`. However, this option requires that you set up of a keystore and related artifacts.
3. Specify a long and encrypted authentication secret that is difficult for others to decipher. Do not use the `ObjectGridDefaultSecret`, which is the value that is used in the `sampleServer.properties` file.
4. Configure the `server.xml` file using the same configuration that you might use for a stand-alone server configuration. In the `server.xml` file, specify the file path to the properties file in a `serverProps` attribute inside the `xsSever` element. See the following example from the `server.xml` file:

```
<server>
...
<xsSever ... serverProps="/path/to/myServerProps.properties" ... />
</server>
```

What to do next

“Authenticating client requests in the Liberty profile” on page 198

Authenticating eXtreme Scale server connections in the OSGi framework

Connections between eXtreme Scale servers in the OSGi framework must be authenticated to prevent an unauthorized server from accessing the data grid.

Before you begin

You must install the OSGi framework before you secure the data grid. For more information, see “Installing the Eclipse Equinox OSGi framework with Eclipse Gemini for clients and servers” on page 219.

About this task

The following settings in the `server.properties` file determine how servers authenticate to one another:

- **securityEnabled=true**
- **secureTokenManagerType=autoSecret**
- **authenticationSecret=OurGridServersExampleSecret**

All of the eXtreme Scale servers in a domain, as well as all of the servers in any linked domains, must use the same values for these properties in the `server.properties` file, or communication fails.

Procedure

1. Enable server to server authentication. Set the **securityEnabled** property to true in the server properties file; for example:

```
securityEnabled=true
```

The default value for this property is false.

2. Establish a secure server configuration. One `secureTokenManagerType` that may be used for a secure configuration is `autoSecret`, which performs token encryption and signing using keys derived from the `authenticationSecret`. Secure tokens are used in server to server authentication and also for client single sign-on tokens. A value of `none` for `secureTokenManagerType` is not secure because this setting prevents the creation of encrypted tokens.

You can also specify a setting of `secureTokenManagerType=default`. However, this option requires that you set up of a key store and related artifacts.

3. Specify a long, string value for the `authenticationSecret` element. This value should be difficult for others to guess. You can encode this value using the `FilePasswordEncoder` utility. Do not use the `ObjectGridDefaultSecret` element, which is the value that is used in the `sampleServer.properties` file.
4. Reference the server properties file. Create a managed, service persistent identifier (PID) for the server properties file in the OSGi console, by running the following commands:

```
osgi> cm create com.ibm.websphere.xs.server
osgi> cm put com.ibm.websphere.xs.server objectgrid.server.props /mypath/server.properties
```

What to do next

“Authenticating client requests in the OSGi framework” on page 200

Authenticating eXtreme Scale server connections in WebSphere Application Server

The eXtreme Scale servers running under WebSphere Application Server authenticate to one another in the same way as eXtreme Scale stand-alone servers.

Before you begin

About this task

Three settings in the `server.properties` file determine how servers authenticate to one another. All of the eXtreme Scale servers in a domain, as well as all of the servers in any linked domains, must use the same values for these three properties in the `server.properties` file, or communication fails. See Security descriptor XML file for more information about the security properties.

Procedure

1. Create the server properties file, and enable server to server authentication. Using this sample server properties file, create a server properties file that contains the **securityEnabled** property, which is set to `true`; for example:

```
securityEnabled=true
```

The default value for this property is `false`.

2. Establish a secure server configuration. One `secureTokenManagerType` that you can use for a secure configuration is `autoSecret`, which performs token encryption and signing using keys derived from the `authenticationSecret`. Secure tokens are used in server to server authentication and also for client single sign-on tokens. A value of `none` for `secureTokenManagerType` is not secure because this setting prevents the creation of encrypted tokens.

You can also specify a setting of `secureTokenManagerType=default`. However, this option requires that you set up of a key store and related artifacts.

3. Specify a long and encrypted authentication secret that is difficult for others to decipher. Do not use the `ObjectGridDefaultSecret`, which is the value that is used in the `sampleServer.properties` file.
4. Configure a server properties file to secure the server. Configure this properties file using the WebSphere Application Server administration console **WebSphere application servers > server_name > Java and Process Management > Process definition > Java virtual machine**. Add the following generic JVM argument:

```
-Dobjectgrid.server.props=<server property file name>
```

What to do next

“Authenticating client requests in WebSphere Application Server” on page 201

Authenticating requests from clients to servers

Your client applications must make secure requests across the network.

What to do next

“Authorizing access to the data grid” on page 202

Authenticating client requests in stand-alone environments

Unless clients are authenticated, access to grid data and JMX management operations that control the grid are left unprotected. This is true even if SSL is enabled.

About this task

The authentication behavior that eXtreme Scale servers require of eXtreme Scale clients is determined by the **credentialAuthentication=required** setting in the `server.properties` file.

When `credentialAuthentication` is set to `Required` or `Supported`, more configuration is needed, as described in the following steps. These steps are described in more detail, with examples of the changes to the configuration files in [Java SE security tutorial - Step 3](#).

Procedure

- Reference a security descriptor XML file in each catalog server.

When the catalog server is started in a stand-alone environment, you can point to this file using the `-clusterSecurityFile` parameter of the **startXsServer** or **startOgServer** command.

To enable security, this file must have `securityEnabled="true"` in the `security` element. The security descriptor XML file must also contain a descriptor of the authenticator that you want to use. WebSphere eXtreme Scale includes the `LDAPAuthenticator`, the `KeyStoreLoginAuthenticator`, and the `WSTokenAuthenticator`. You cannot use the `WSTokenAuthenticator` authenticator in the stand-alone environments. You can only use this authenticator when eXtreme Scale clients and servers are both running with WebSphere Application Server. Alternatively, you can develop custom authenticators and login modules, according to the interfaces described in the API documentation.

- Reference a JAAS configuration file in each catalog and container server using the `-Djava.security.auth.login.config="path_name"` JVM argument. For information about creating these files and configuring eXtreme Scale servers to use them, see the tutorial, [Tutorial: Configuring Java SE security](#). The JAAS configuration file specifies a `LoginModule`. You can use the `KeyStoreLoginModule` with the `KeyStoreLoginAuthenticator`. Use the `SimpleLDAPLoginModule` with the `LDAPAuthenticator`. See [Enabling LDAP authentication in eXtreme scale catalog and container servers in eXtreme Scale container and catalog servers](#), or [Enabling keystore authentication in eXtreme Scale container and catalog servers](#).
- Configure the client to pass the credentials that are required for authentication. This is typically done by the client loading a client security configuration that is defined in a client security properties file. For more information about enabling LDAP authentication in eXtreme Scale clients, see [Enabling LDAP authentication in eXtreme scale catalog and container servers](#), and for more information about enabling keystore authentication in eXtreme Scale clients, see [Enabling keystore authentication in eXtreme Scale container and catalog servers](#).

What to do next

[“Authorizing access to the data grid in stand-alone environments”](#) on page 203

Authenticating client requests in the Liberty profile

Unless clients are authenticated, access to grid data and JMX management operations that control the grid are left unprotected. This is true even if SSL is enabled in the Liberty profile.

About this task

The authentication behavior that is required by eXtreme Scale clients is determined by the **credentialAuthentication=required** setting in the `server.properties` file, the **KeyStoreLogin** setting in the `og_jaas.config` JAAS configuration file, and the **KeyStoreLoginAuthenticator** setting in the `security.xml` file.

The server properties file is loaded by referring to it in the `server.xml` file, as described in “Authenticating eXtreme Scale server connections in the Liberty profile” on page 195. For security, this file must have `credentialAuthentication=Required`, just as in stand-alone deployments.

Each of the configuration files is loaded by each catalog server. Container servers use the JAAS configuration file and the security deployment descriptor files only.

Use one of the following methods to authenticate clients.

Procedure

- Reference a security descriptor XML file in each catalog server.

When the catalog server is the Liberty profile, you can point to this file using the `clusterSecurityURL=` attribute in the `server.xml` file. See the following example, where `objectGridSecurity.xml` is the security descriptor XML file:

```
<server description="new server">
<!-- Enable features -->
<featureManager>
<feature>eXtremeScale.server-1.1</feature>
</featureManager>

<xsServer
isCatalog="true"
serverProps="server.xs.props"
clusterSecurityURL="file://C:/wlp/usr/servers/objectGridSecurity.xml"
/>
</server>
```

To enable security, this file must have `securityEnabled="true"` in the security element. The security descriptor XML file must also contain a descriptor of the authenticator that you want to use. WebSphere eXtreme Scale includes the `LDAPAuthenticator`, the `KeyStoreLoginAuthenticator`, and the `WSTokenAuthenticator`.

- Reference a JAAS configuration file in each catalog and container server using the `-Djava.security.auth.login.config="path_name"` JVM argument in the `jvm.options` file.

Edit or create the `jvm.options` file in the `wlp_install_dir/usr/servers/<server_name>` directory.

Note: If you need to create a `jvm.options` file at the server configuration level, you need to copy the version in the `wlp_install_root/etc/jvm.options` file. The `jvm.options` file has some options that are needed for eXtreme Scale to run in the Liberty profile.

When you create a `jvm.options` file at the server level and enter the JVM argument to reference the JAAS configuration file, your `jvm.options` files looks like this:

```
C:\wlp\usr\servers\simpCatalog>cat jvm.options
-Dorg.osgi.framework.bootdelegation=com.ibm.wsspi.runtime
-Djava.endorsed.dirs=C:\wlp\wxs\lib\endorsed
-Djava.security.auth.login.config=C:\wlp\usr\servers\ogjaas.config
```

For information about creating these files and configuring eXtreme Scale servers to use them, see the tutorial, Tutorial: Configuring Java SE security. The JAAS configuration file specifies a LoginModule. You can use the KeyStoreLoginModule with the KeyStoreLoginAuthenticator. Use the SimpleLDAPLoginModule with the LDAPAuthenticator. See Enabling LDAP authentication in eXtreme scale catalog and container servers in eXtreme Scale container and catalog servers, or Enabling keystore authentication in eXtreme Scale container and catalog servers.

- Configure the client to pass the credentials that are required for authentication. This is typically done by specifying values in a client properties file. For more information about enabling LDAP authentication in eXtreme Scale clients, see Enabling LDAP authentication in eXtreme scale catalog and container servers, and for more information about enabling keystore authentication in eXtreme Scale clients, see Enabling keystore authentication in eXtreme Scale container and catalog servers.

What to do next

“Authorizing access to the data grid in the Liberty profile” on page 203

Authenticating client requests in the OSGi framework

Unless clients are authenticated, access to grid data and JMX management operations that control the grid are left unprotected. This is true even if SSL is enabled in the OSGi framework.

Before you begin

You must install the OSGi framework before you secure the data grid. For more information, see “Installing the Eclipse Equinox OSGi framework with Eclipse Gemini for clients and servers” on page 219.

About this task

The authentication behavior that is required by eXtreme Scale clients is determined by the **credentialAuthentication=required** setting in the `server.properties` file, the **KeyStoreLogin** setting in the `og_jaas.config` JAAS configuration file, and the **KeyStoreLoginAuthenticator** setting in the `security.xml` file.

Use one of the following methods to authenticate clients.

Procedure

- Reference a security descriptor XML file in each catalog server using `-DclusterSecurityFile="path_name"` JVM argument.

Use this JVM argument on the OSGi command line when you start the catalog server.

To enable security, this file must have `securityEnabled="true"` in the `security` element. The security descriptor XML file must also contain a descriptor of the authenticator that you want to use. WebSphere eXtreme Scale includes the `LDAPAuthenticator`, the `KeyStoreLoginAuthenticator`, and the `WSTokenAuthenticator`. You cannot use the `WSTokenAuthenticator` authenticator in the stand-alone environments. You can only use this authenticator when eXtreme Scale clients and servers are both running with WebSphere Application Server. Alternatively, you can develop custom authenticators and login modules, according to the interfaces described in the API documentation.

- Reference a JAAS configuration file in each catalog and container server using the `-Djava.security.auth.login.config="path_name"` JVM argument. For information about creating these files and configuring eXtreme Scale servers to use them, see the tutorial, Tutorial: Configuring Java SE security. The JAAS configuration file specifies a LoginModule. You can use the `KeyStoreLoginModule` with the `KeyStoreLoginAuthenticator`. Use the `SimpleLDAPLoginModule` with the `LDAPAuthenticator`. See Enabling LDAP authentication in eXtreme scale catalog and container servers in eXtreme Scale container and catalog servers, or Enabling keystore authentication in eXtreme Scale container and catalog servers.
- Configure the client to pass the credentials that are required for authentication. This is typically done by specifying values in a client properties file. For more information about enabling LDAP authentication in eXtreme Scale clients, see Enabling LDAP authentication in eXtreme scale catalog and container servers, and for more information about enabling keystore authentication in eXtreme Scale clients, see Enabling keystore authentication in eXtreme Scale container and catalog servers.

What to do next

“Authorizing access to the data grid in the OSGi framework” on page 204

Authenticating client requests in WebSphere Application Server

Requests that WebSphere Application Server receives from the eXtreme Scale data grid must be authenticated.

Before you begin

Authentication requirements for eXtreme Scale clients are determined by the settings in the server properties file. A sample server properties file is provided in `was_root/optionalLibraries/ObjectGrid/properties/sampleServer.properties`.

About this task

You must configure authentication for eXtreme Scale servers that are running under WebSphere Application Server using the following steps.

Procedure

1. Create the server properties file. Using this sample server properties file, create a server properties file that contains the following lines:

```
securityEnabled=true
credentialAuthentication=Required
```

Unless the `credentialAuthentication=Required` property exists, the grid is not secure, and unauthenticated users can perform grid operations.

Restriction: You cannot specify the property, `credentialAuthentication=Required`, for the dynamic cache provider.

2. Create the security descriptor XML file. When the property, `credentialAuthentication`, is set to `Required` or `Supported`, you must specify a security descriptor XML file. See the following example:

```
<securityConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config/security ../objectGridSecurity.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config/security">

  <security securityEnabled="true">
    <authenticator
```

```

        className="com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenAuthenticator">
        </authenticator>
    </security>
</securityConfig>

```

The security descriptor XML file specifies the authenticator to be used. When all eXtreme Scale clients and servers are running under WebSphere Application Server, you can use the WSTokenAuthenticator authenticator. Two other authenticators are shipped with eXtreme Scale, the KeyStoreLoginAuthenticator and the LDAPLoginAuthenticator. For more information about configuring LDAP authentication for eXtreme Scale, see *Enabling LDAP authentication in eXtreme scale catalog and container servers*. To use the keystore and login authenticators with eXtreme Scale running under WebSphere Application Server, a JAAS configuration is needed. For more information about configuring keystore authentication for eXtreme Scale, see *Enabling keystore authentication in eXtreme Scale container and catalog servers*.

3. Create the JAAS configuration, unless you are using the WSTokenAuthenticator authenticator.
4. Point each catalog server at the server properties file using the following JVM arguments. Configure these properties using the WebSphere Application Server administration console **Servers > all servers > server_name > Process definition > Java virtual machine-generic JVM arguments**. The following arguments are required:


```

-Dobjectgrid.server.props=<server property file name>
-Dobjectgrid.cluster.security.xml.url=file://<security descriptor XML file>

```
5. Point each container server to the server properties file using this JVM argument:


```

-Dobjectgrid.server.props=<server property file name>

```

What to do next

WebSphere eXtreme Scale clients must be configured to pass appropriate credentials. Complete this configuration using a client properties file. See the following example of the WSTokenAuthenticator authenticator:

```

securityEnabled=true
credentialAuthentication=supported
credentialGeneratorClass=com.ibm.websphere.security.plugins.builtins.WSTokenCredentialGenerator

```

A client must be configured to use this file. When the client is running under WebSphere Application Server. Configure the client with the following JVM argument:

```

-Dobjectgrid.client.props=<client properties file>

```

To secure the grid deployment, set application security and Java 2 Security for WebSphere Application Server servers that are hosting eXtreme Scale servers. Use the WebSphere Application Server administrative console security configuration panel to enable these settings.

Now, you can proceed to the next step, “Authorizing access to the data grid in WebSphere Application Server” on page 205.

Authorizing access to the data grid

Enforce access control so that authenticated identities can only perform operations for which they are specifically authorized.

What to do next

“Authorizing access for special administrative operations” on page 206

Authorizing access to the data grid in stand-alone environments

Control which users have specific permissions to access the data grid through the policy file.

About this task

Even if a client is authenticated, that might not be enough to protect data grid access. If you use the `KeyStoreLoginAuthenticator`, usually you only define a few identities, and all of the identities might have full access to the data grid. In this case, authorization might not be necessary. However, if LDAP authentication is used, there might be many identities in the LDAP server that must not be granted access to grid data or operations.

Procedure

1. Enable access control for the data grid. Specify `securityEnabled="true"` in the `ObjectGrid.xml` file for the deployed data grid.
Specify this setting for each grid you define. After you configure this setting, no reads or writes are run on data grid entries except for identities that have been granted permissions in a policy file.

2. Create a policy file. See the following example policy file:

```
grant codebase "http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction"
    principal javax.security.auth.x500.X500Principal "CN=cashier,0=acme,OU=OGSample" {
    permission com.ibm.websphere.objectgrid.security.MapPermission "accounting.*", "read";
};

grant codebase "http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction"
    principal javax.security.auth.x500.X500Principal "CN=manager,0=acme,OU=OGSample" {
    permission com.ibm.websphere.objectgrid.security.MapPermission "accounting.*", "all";
};
```

Policy files can grant various permissions, depending on the authorization of the user. For more information about how to create this file, see [Java SE security tutorial - Step 5](#).

3. Configure each container server to load this policy file. You can complete this configuration by starting the container with the following JVM argument:
`-Djava.security.policy=<policy file>`

Tip: This policy file is also used in controlling administrative access to data grid servers. When you use this policy file to control administrative access, the policy file must contain `MBeanPermission` entries, and must be loaded by catalog servers and container servers.

What to do next

“Authorizing access for administrative operations in stand-alone environments” on page 206

Authorizing access to the data grid in the Liberty profile

Control which users have specific permissions to access the data grid in the Liberty profile through the policy file.

About this task

Even if a client is authenticated, that might not be enough to protect data grid access. If you use the `KeyStoreLoginAuthenticator` property, usually you define only a few identities, and all of the identities might have full access to the grid. In which case, authorization might not be necessary. Alternatively, if LDAP authentication is used, there might be many identities in the LDAP server that should not be granted access to grid data or operations.

Procedure

1. Enable access control for the data grid. Specify `securityEnabled="true"` in the `ObjectGrid.xml` file for the deployed data grid.
Specify this setting for each grid you define. After you configure this setting, no reads or writes are run on data grid entries except for identities that have been granted permissions in a policy file.

2. Create a policy file. See the following example policy file:

```
grant codebase "http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction"
    principal javax.security.auth.x500.X500Principal "CN=cashier,0=acme,OU=OGSample" {
    permission com.ibm.websphere.objectgrid.security.MapPermission "accounting.*", "read";
};

grant codebase "http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction"
    principal javax.security.auth.x500.X500Principal "CN=manager,0=acme,OU=OGSample" {
    permission com.ibm.websphere.objectgrid.security.MapPermission "accounting.*", "all";
};
```

Policy files can grant various permissions, depending on the authorization of the user. For more information about how to create this file, see [Java SE security tutorial - Step 5](#).

3. Configure each container server to load this policy file. You can complete this configuration by adding the following JVM argument to the `jvm.options` file in the `wlp_install_dir/usr/servers/<server_name>` directory:

```
-Djava.security.policy=<policy file>
```

Tip: This policy file is also used in controlling administrative access to data grid servers. When you use this policy file to control administrative access, the policy file must contain `MBeanPermission` entries, and must be loaded by catalog servers and container servers.

If you need to create a `jvm.options` file at the server configuration level, you need to copy the version in the `wlp_install_root/etc/jvm.options` file.

What to do next

“Authorizing access for administrative operations in the Liberty profile” on page 207

Authorizing access to the data grid in the OSGi framework

Control which users have specific permissions to access the data grid in the OSGi framework through the policy file.

Before you begin

You must install the OSGi framework before you secure the data grid. For more information, see “Installing the Eclipse Equinox OSGi framework with Eclipse Gemini for clients and servers” on page 219.

About this task

Even if a client is authenticated, that might not be enough to protect data grid access. If you use the `KeyStoreLoginAuthenticator` property, usually you only define a few identities, and all of the identities might have full access to the grid. In which case, authorization may not be necessary. Alternatively, if LDAP authentication is used, there might be many identities in the LDAP server that should not be granted access to grid data or operations.

Procedure

1. Enable access control for the data grid. Specify `securityEnabled="true"` in the `ObjectGrid.xml` file for the deployed data grid.
Specify this setting for each grid you define. After you configure this setting, no reads or writes are run on data grid entries except for identities that have been granted permissions in a policy file.

2. Create a policy file. Add the following lines of code to the security policy file to give `AllPermission` to the `osgi.jar` file for the deployed data grid.

```
grant codeBase "file:/opt/OSGI2/plugins/org.eclipse.osgi_3.7.1.R37x_v20110808-1106.jar" {  
    permission java.security.AllPermission;  
};
```

Specify this code for each grid you define. After you configure this setting, no reads or writes are run on data grid entries except for identities that have been specifically granted permissions in a policy file. Policy files can grant various permissions, depending on the authorization of the user. For more information about how to create this file, see [Java SE security tutorial - Step 5](#).

The policy file resembles the following example:

Remember: The policy file also typically contains `MapPermission` entries, as documented in [Java SE security tutorial - Step 5](#).

```
grant codeBase "file:${objectgrid.home}/lib/*" {  
    permission java.security.AllPermission;  
};
```

```
grant principal javax.security.auth.x500.X500Principal "CN=manager,O=acme,OU=OGSample"  
{  
    permission javax.management.MBeanPermission "*", "getAttribute,setAttribute,invoke,queryNames,addNotificationListener,removeNotificat  
};
```

3. Configure each container server to load this policy file. You can complete this configuration by starting the container with the following JVM argument:

```
-Djava.security.policy=<policy file>
```

Tip: This policy file is also used in controlling administrative access to data grid servers. When you use this policy file to control administrative access, the policy file must contain `MBeanPermission` entries, and must be loaded by catalog servers and container servers.

What to do next

“Authorizing access for administrative operations in the OSGi framework” on page 208

Authorizing access to the data grid in WebSphere Application Server

Control which users have specific permissions to access the data grid in WebSphere Application Server deployments in the same way that you control access to the data grid in stand-alone deployments.

About this task

Even if a client is authenticated, that might not be enough to protect data grid access. If you use the `KeyStoreLoginAuthenticator`, usually you only define a few identities, and all of the identities might have full access to the data grid. In this case, authorization might not be necessary. However, if LDAP authentication is used, there might be many identities in the LDAP server that must not be granted access to grid data or operations.

Attention: It is not necessary to specify `MBeanPermissions` for WebSphere Application Server deployments of eXtreme Scale servers because JMX access is controlled by the WebSphere Application Server, itself.

Procedure

1. Enable access control for the data grid. Specify `securityEnabled="true"` in the `ObjectGrid.xml` file for the deployed data grid.
Specify this setting for each grid you define. After you configure this setting, no reads or writes are run on data grid entries except for identities that have been granted permissions in a policy file.
2. Create a policy file. Policy files can grant various permissions, depending on the authorization of the user. For more information about how to create this file, see Lesson 4.2: Enable user-based authorization.
3. Configure each container server to load this policy file. You can specify the policy file in the Generic JVM arguments of the application server where the container runs. For more information about setting the server properties file with JVM properties, see Lesson 2.2: Configure catalog server security.
`-Djava.security.auth.policy=<policy file>`

What to do next

“Authorizing access for administrative operations in WebSphere Application Server” on page 209

Authorizing access for special administrative operations

Require special authorization for users to perform administrative operations on the data grid.

What to do next

“Securing data that flows between eXtreme Scale clients and servers with SSL encryption” on page 209

Authorizing access for administrative operations in stand-alone environments

Most data grid deployers restrict administrative access to only a subset of the users who can access grid data.

Procedure

You must run the catalog servers and container servers using the Java security manager, which requires a policy file.

The policy file is specified by passing the `-Djava.security.policy=<policy_file>` JVM argument.

The Java security manager is started by specifying the JVM argument,

-Djava.security.manager, when the eXtreme Scale server is started. Specify this argument for both container and catalog servers. The policy file resembles the following example:

Remember: The policy file also typically contains MapPermission entries, as documented in Java SE security tutorial - Step 5.

```
grant codeBase "file:${objectgrid.home}/lib/*" {
  permission java.security.AllPermission;
};

grant principal javax.security.auth.x500.X500Principal "CN=manager,O=acme,OU=OGSample"
{
  permission javax.management.MBeanPermission "*", "getAttribute,setAttribute,invoke,queryNames,addNotificationListener,removeNotificat
};
```

In this example, only the manager principal is authorized for administrative operations with the **xscmd** command. You can add other lines as necessary to give additional principals MBean permissions.

Enter the following command: UNIX Linux

```
startOgServer.sh <arguments> -jvmargs -Djava.security.auth.login.config=jaas.config
-Djava.security.manager -Djava.security.policy="auth.policy" -Dobjectgrid.home=$OBJECTGRID_HOME
```

UNIX Linux **8.6+**

```
startXsServer.sh <arguments> -jvmargs -Djava.security.auth.login.config=jaas.config
-Djava.security.manager -Djava.security.policy="auth.policy" -Dobjectgrid.home=$OBJECTGRID_HOME
```

Windows

```
startOgServer.bat <arguments> -jvmargs -Djava.security.auth.login.config=jaas.config
-Djava.security.manager -Djava.security.policy="auth.policy" -Dobjectgrid.home=%OBJECTGRID_HOME%
```

Windows **8.6+**

```
startXsServer.bat <arguments> -jvmargs -Djava.security.auth.login.config=jaas.config
-Djava.security.manager -Djava.security.policy="auth.policy" -Dobjectgrid.home=%OBJECTGRID_HOME%
```

What to do next

“Securing data that flows between eXtreme Scale servers in stand-alone environments with SSL encryption” on page 209

Authorizing access for administrative operations in the Liberty profile

Through administrative security, you can authorize users to access the data grid in the Liberty profile.

About this task

Most data grid deployers restrict administrative access to only a subset of the users who can access grid data.

Procedure

- Run the Java security manager, and specify a policy file that grants MBeanPermissions, to restrict administrative access when eXtreme Scale servers are running in the Liberty profile. This approach is the same as in stand-alone deployments. Enter the following lines to the `jvm.options` file for each Liberty profile server that is running an eXtreme Scale catalog or container server.
-Djava.security.manager
-Djava.security.policy="policy file"

- Configure the policy file to grant the Liberty profile and the eXtreme Scale code all permissions. This configuration allows the Liberty profile and the eXtreme Scale to work with the security manager. Add the following lines to the `jvm.options` file that is at the server level:

```
grant codeBase "file:${objectgrid.home}/lib/*" {
  permission java.security.AllPermission;
};
```

What to do next

“Securing data that flows between eXtreme Scale and the Liberty profile with SSL encryption” on page 211

Authorizing access for administrative operations in the OSGi framework

Through administrative security, you can authorize users to access the data grid in the OSGi framework.

Before you begin

You must install the OSGi framework before you secure the data grid. For more information, see “Installing the Eclipse Equinox OSGi framework with Eclipse Gemini for clients and servers” on page 219.

About this task

Most data grid deployers restrict administrative access to only a subset of the users who can access grid data.

Procedure

- You must run the catalog servers and container servers using the Java security manager, which requires a policy file.

The policy file is specified by passing the `-Djava.security.policy=<policy_file>` JVM argument.

The Java security manager is started by specifying the JVM argument, `-Djava.security.manager`, when the eXtreme Scale server is started. Specify this argument for both container and catalog servers.

The policy file resembles the following example:

Remember: The policy file also typically contains `MapPermission` entries, as documented in Java SE security tutorial - Step 5.

```
grant codeBase "file:${objectgrid.home}/lib/*" {
  permission java.security.AllPermission;
};

grant principal javax.security.auth.x500.X500Principal "CN=manager,O=acme,OU=OGSample"
{
  permission javax.management.MBeanPermission "*", "getAttribute,setAttribute,invoke,queryNames,addNotificationListener,removeNotificationListener";
};
```

In this example, only the manager principal is authorized for administrative operations with the `xs cmd` command. You can add other lines as necessary to give additional principals MBean permissions.

- Start the catalog and server containers by specifying the previous JVM arguments on the command line; for example:

```
/opt/XS86/java/jre/bin/java -DclusterSecurityFile=/og/security/secFiles_SA/objectGridSecurity.xml
```

What to do next

“Securing data that flows between eXtreme Scale and the OSGi framework with SSL encryption” on page 212

Authorizing access for administrative operations in WebSphere Application Server

Through administrative security, only WebSphere Application Server administrators can perform eXtreme Scale administrative operations.

About this task

Authorization for administrative access works differently in WebSphere Application Server deployments than in stand-alone environments. Only WebSphere Application Server users that are WebSphere Application Server administrators can perform eXtreme Scale administrative operations. You do not need to specify MbeanPermissions in the policy file.

Procedure

Enable administrative security in WebSphere Application Server. In the administrative console, click **Security > Global Security**. Click **Enable administrative security**, and select **Java 2 security** to restrict application access to local resources.

What to do next

“Securing data that flows between eXtreme Scale and WebSphere Application Server with SSL encryption” on page 214

Securing data that flows between eXtreme Scale clients and servers with SSL encryption

Java

Protect communication between WebSphere eXtreme Scale clients and servers with SSL encryption.

What to do next

“Storing security artifacts for authorized users” on page 214

Securing data that flows between eXtreme Scale servers in stand-alone environments with SSL encryption

Java

Configure SSL properties and JMX ports to secure sensitive information that flows between servers over the network.

About this task

When a data grid is deployed, the sensitive information it contains flows over the network. Also, the credentials that data grid clients use to authenticate to the data grid flow over the network. To protect data and credentials as they flow, use transport-level encryption using SSL to secure deployments.

The security of SSL depends on protecting the keystores and the truststores, so that only authorized users have access to the keystores and truststores. After you enable SSL encryption, you must specify a `JMXConnectorPort` and a `JMXServicePort` value in the server properties file to have SSL protection for JMX traffic.

The transport between the JMX client and server can be secured with transport layer security (TLS) or SSL. If the `transportType` of catalog server or container server is set to `SSL_Required` or `SSL_Supported`, then you must use SSL to connect to the JMX server.

Procedure

1. Specify SSL in the server properties file. Set the `transportType` property to `SSL-Required`; for example:

```
transportType=SSL-Required
```

2. Specify SSL properties in the server properties file.

```
alias=serverprivate
contextProvider=IBMJSSE2
protocol=SSL
keyStoreType=JKS
keyStore=etc/test/security/key.jks
keyStorePassword=serverpw
trustStoreType=JKS
trustStore=etc/test/security/trust.jks
trustStorePassword=public
clientAuthentication=false
```

Configure the truststore, truststore type, and truststore password. It is not necessary to specify a keystore, keystore type, and key store password for the client. The alias, keystore, keystore password, and keystore type are not needed on the client unless the server SSL properties includes `clientAuthentication=true`. This value is rarely used.

The client truststore must trust the server certificate. When the server certificate is self signed, as in the tutorial, that certificate must be imported into the client trust store. When the server certificate is issued by a local certificate authority, the signer certificate for that certificate authority must be imported into the client truststore. For more information about creating keystore and truststore files, see [Java SE security tutorial - Step 6](#).

3. Specify SSL in the client properties file when SSL is required. Set the `transportType` property to `SSL-Required` or `SSL-Supported`; for example:

```
transportType=SSL-Required
```

4. Specify SSL properties in the client properties file. For example, you can specify the following properties:

```
alias=clientprivate
contextProvider=IBMJSSE2
protocol=SSL
keyStoreType=JKS
keyStore=etc/test/security/client.private
keyStorePassword={xor}PDM20jErLyg\=
trustStoreType=JKS
trustStore=etc/test/security/server.public
trustStorePassword={xor}Lyo9MzY8
```

5. Set the JMX service port. Use the `-JMXServicePort` option on the `startOgServer` or `startXsServer` script.

The default value for the JMX service port on catalog servers is 1099. You must use a different port number for each JVM in your configuration. If you want to

use JMX/RMI, explicitly specify the **-JMXServicePort** option and port number, even if you want to use the default port value.

6. Set the JMX connector port.

Use the **-JMXConnectorPort** option on the **startOgServer** or **startXsServer** script.

Setting the JMX service port is required when you want to display container server information from the catalog server. For example, the port is required when you are using the **xscmd -c showMapSizes** command. Set the JMX connector port to avoid ephemeral port creation.

What to do next

“Storing security artifacts in stand-alone environments” on page 215

Securing data that flows between eXtreme Scale and the Liberty profile with SSL encryption

Java

Configure SSL properties and JMX ports to secure sensitive information that flows between WebSphere eXtreme Scale and the Liberty profile.

About this task

When a data grid is deployed, the sensitive information it contains flows over the network. Also, the credentials that data grid clients use to authenticate to the data grid flow over the network. To protect data and credentials as they flow, use transport-level encryption using SSL to secure deployments.

The security of SSL depends on protecting the keystores and the truststores, so that only authorized users have access to the keystores and truststores. After you enable SSL encryption, you must specify a **JMXConnectorPort** and a **JMXServicePort** value in the server properties file to have SSL protection for JMX traffic.

The transport between the JMX client and server can be secured with transport layer security (TLS) or SSL. If the **transportType** of catalog server or container server is set to **SSL_Required** or **SSL_Supported**, then you must use SSL to connect to the JMX server.

Procedure

1. Specify SSL in the server properties file. Set the **transportType** property to **SSL-Required**; for example:

```
transportType=SSL-Required
```

2. Specify SSL properties in the server properties file.

```
alias=serverprivate
contextProvider=IBMJSSE2
protocol=SSL
keyStoreType=JKS
keyStore=etc/test/security/key.jks
keyStorePassword=serverpw
trustStoreType=JKS
trustStore=etc/test/security/trust.jks
trustStorePassword=public
clientAuthentication=false
```

Configure the truststore, truststore type, and truststore password. It is not necessary to specify a keystore, keystore type, and key store password for the client. The alias, keystore, keystore password, and keystore type are not needed on the client unless the server SSL properties includes `clientAuthentication=true`. This value is rarely used.

The client truststore must trust the server certificate. When the server certificate is self signed, as in the tutorial, that certificate must be imported into the client trust store. When the server certificate is issued by a local certificate authority, the signer certificate for that certificate authority must be imported into the client truststore. For more information about creating keystore and truststore files, see Java SE security tutorial - Step 6.

3. Specify SSL in the client properties file when SSL is required. Set the `transportType` property to `SSL-Required` or `SSL-Supported`; for example:

```
transportType=SSL-Required
```

4. Specify SSL properties in the client properties file. For example, you can specify the following properties:

```
alias=clientprivate
contextProvider=IBMJSSE2
protocol=SSL
keyStoreType=JKS
keyStore=etc/test/security/client.private
keyStorePassword={xor}PDM20jErLyg\=
trustStoreType=JKS
trustStore=etc/test/security/server.public
trustStorePassword={xor}Lyo9MzY8
```

Specify the client properties file in the `jvm.options` file; for example:

```
-Dobjectgrid.client.props="D:\IDES\wxsEnvi\wlp\usr\servers\sessionAppServer\objectGridClient.prop
```

Remove the double quotation marks if you are using Linux operating systems.

5. Set the JMX service port in the server properties file.

The default value for the JMX service port on catalog servers is 1099. You must use a different port number for each JVM in your configuration. If you want to use JMX/RMI, explicitly specify the **server JMXServicePort** option and port number, even if you want to use the default port value.

6. Set the JMX connector port in the server properties file.

Setting the JMX service port is required when you want to display container server information from the catalog server. For example, the port is required when you are using the `xscmd -c showMapSizes` command. Set the JMX connector port to avoid ephemeral port creation.

What to do next

“Storing security artifacts in the Liberty profile” on page 215

Securing data that flows between eXtreme Scale and the OSGi framework with SSL encryption

Java

Configure SSL properties and JMX ports to secure sensitive information that flows between WebSphere eXtreme Scale and the OSGi framework.

Before you begin

You must install the OSGi framework before you secure the data grid. For more information, see “Installing the Eclipse Equinox OSGi framework with Eclipse

Gemini for clients and servers” on page 219.

About this task

When a data grid is deployed, the sensitive information it contains flows over the network. Also, the credentials that data grid clients use to authenticate to the data grid flow over the network. To protect data and credentials as they flow, use transport-level encryption using SSL to secure deployments.

The security of SSL depends on protecting the keystores and the truststores, so that only authorized users have access to the keystores and truststores. After you enable SSL encryption, you must specify a `JMXConnectorPort` and a `JMXServicePort` value in the server properties file to have SSL protection for JMX traffic.

The transport between the JMX client and server can be secured with transport layer security (TLS) or SSL. If the `transportType` of catalog server or container server is set to `SSL_Required` or `SSL_Supported`, then you must use SSL to connect to the JMX server.

Procedure

1. Specify SSL in the server properties file. Set the `transportType` property to `SSL-Required`; for example:
`transportType=SSL-Required`
2. To use SSL, you need to configure the truststore, truststore type, and truststore password on the MBean client with `-D` system properties; for example:
`-Djavax.net.ssl.trustStore=TRUST_STORE_LOCATION`
`-Djavax.net.ssl.trustStorePassword=TRUST_STORE_PASSWORD`
`-Djavax.net.ssl.trustStoreType=TRUST_STORE_TYPE`

If you use `com.ibm.websphere.ssl.protocol.SSLSocketFactory` as your SSL socket factory in your `java_home/jre/lib/security/java.security` file, then use the following properties:

```
-Dcom.ibm.ssl.trustStore=TRUST_STORE_LOCATION
-Dcom.ibm.ssl.trustStorePassword=TRUST_STORE_PASSWORD
-Dcom.ibm.ssl.trustStoreType=TRUST_STORE_TYPE
```

3. Set the JMX service port in the server properties file.
The default value for the JMX service port on catalog servers is 1099. You must use a different port number for each JVM in your configuration. If you want to use JMX/RMI, explicitly specify the `JMXServicePort` option and port number, even if you want to use the default port value.
4. Set the JMX connector port in the server properties file.
Setting the JMX service port is required when you want to display container server information from the catalog server. For example, the port is required when you are using the `xscmd c showMapSizes` command. Set the JMX connector port to avoid ephemeral port creation.
5. Specify the SSL port on the OSGi framework command line using the following JVM argument:
`-Dcom.ibm.CSI.SSL.Port=7602`

What to do next

“Storing security artifacts in the OSGi framework” on page 216

Securing data that flows between eXtreme Scale and WebSphere Application Server with SSL encryption

Java

WebSphere eXtreme Scale uses the Secure Sockets Layer (SSL) configuration in WebSphere Application Server .

About this task

To ensure that you have SSL protection for all data grid traffic that passes over the network, configure global security, configure CSIV2 inbound and outbound security in the WebSphere Application Server administrative console, and configure the SSL certificate and key management.

Procedure

1. Configure WebSphere Application Server global security. For more information about configuring global security, see Global security settings.
2. Configure CSIV2 inbound security. In the WebSphere Application Server administrative console, click **Security > Global Security > RMI/IIOP Security > CSIV2 inbound communications**. Click **SSL-Required**.
3. Configure CSIV2 outbound security. In the WebSphere Application Server administrative console, click **Security > Global Security > RMI/IIOP Security > CSIV2 inbound communications**. CSIV2 outbound communications must be **SSL-Supported** or **SSL-Required**.
4. Configure the SSL certificate and key management in WebSphere Application Server. When running only a WebSphere eXtreme Scale client in a WebSphere Application Server instance and the eXtreme Scale data grid servers are stand-alone. You must ensure that the keystore and truststore certificate information is included in the keystore and truststore files that are specified in the server properties file that is used to start your stand-alone catalog and containers serves.

When the client, catalog and container servers are all running in WebSphere Application Server processes, they use the WebSphere Application Server security configuration for the client-to-servers communication.

However, when multiple catalog servers are configured and running in a WebSphere Application Server process the catalog-to-catalog communication has its own proprietary transport paths that cannot be managed by the WebSphere Application Server Common Secure Interoperability Protocol Version 2 (CSIV2) transport settings. Therefore, you must configure the SSL properties in the server properties file for each catalog server. For more information, see Lesson 3.2: Add SSL properties to the catalog server properties file.

What to do next

“Storing security artifacts in WebSphere Application Server” on page 216

Storing security artifacts for authorized users

Key stores, passwords, shared secrets, and properties files must be stored in a directory that can only be accessed by authorized users.

What to do next

Starting and stopping secure servers

Storing security artifacts in stand-alone environments

Java

Protect secure passwords to prevent access from unauthorized users.

About this task

The FilePasswordEncoder utility is included with WebSphere eXtreme Scale Client to encode passwords in eXtreme Scale configuration files. The FilePasswordEncoder utility encodes passwords; however, it is possible to recover the passwords that are used to access the file. Therefore, you must protect the file system on which the client properties, the server properties, and the keystores and truststores are kept, so that only authorized users have access.

Procedure

Run the **FilePasswordEncoder.bat|sh** command to encode this property using an exclusive or (xor) algorithm to provide a measure of protection for passwords. Run the FilePasswordEncoder utility on the `client.properties` file and the `server.properties` file; for example:

```
./FilePasswordEncoder.sh <server properties file>  
./FilePasswordEncoder.sh <client properties file>
```

A sophisticated user can recover encoded passwords. These passwords are not encrypted because the eXtreme Scale code must be able to recover them to run. Therefore, ensure that only authorized persons can access the files where these passwords are stored.

What to do next

Starting secure servers in a stand-alone environment

Storing security artifacts in the Liberty profile

Java

Protect secure passwords to prevent access from unauthorized eXtreme Scale users in the Liberty profile.

About this task

The FilePasswordEncoder utility is included with WebSphere eXtreme Scale Client to encode passwords in eXtreme Scale configuration files.

Procedure

1. Run the Liberty profile **securityUtility.bat|sh** command to encode this property using an exclusive or (xor) algorithm to provide a measure of protection for passwords. Be aware that a sophisticated user can recover encoded passwords. These passwords are not encrypted because the eXtreme Scale code must be able to recover them to run. Therefore, ensure that only authorized persons can access the files where these passwords are stored.
2. Limit access to keystore files and truststore files by protecting access to the file system where they are stored.

What to do next

Starting and stopping secure servers in the Liberty profile

Storing security artifacts in the OSGi framework

Java

Protect secure passwords to prevent access from unauthorized users in the OSGi framework.

Before you begin

You must install the OSGi framework before you secure the data grid. For more information, see “Installing the Eclipse Equinox OSGi framework with Eclipse Gemini for clients and servers” on page 219.

About this task

The FilePasswordEncoder utility is included with WebSphere eXtreme Scale Client to encode passwords in eXtreme Scale configuration files.

Procedure

1. Run the **FilePasswordEncoder.bat|sh** command to encode this property using an exclusive or (xor) algorithm to provide a measure of protection for passwords. Be aware that a sophisticated user can recover encoded passwords. These passwords are not encrypted because the eXtreme Scale code must be able to recover them to run. Therefore, ensure that only authorized persons can access the files where these passwords are stored.
2. Limit access to keystore files and truststore files by protecting access to the file system where they are stored.

What to do next

Starting and stopping secure servers in the OSGi framework

Storing security artifacts in WebSphere Application Server

Java

Protect secure passwords to prevent access from unauthorized users in WebSphere Application Server deployments.

About this task

Passwords and the authenticationSecret in the server properties and client properties files must be encoded.

Procedure

Invoke the PropFilePasswordEncoder to encode passwords and the authentication secret. Run the following commands *was_root/bin/PropFilePasswordEncoder.sh* command, or on Windows, run the *was_root\bin\PropFilePasswordEncoder.bat* command; for example:

```
./PropFilePasswordEncoder <properties_file> <property_to_encode>
```

Properties that should be encoded include the **keyStorePassword**, **trustStorePassword**, **credentialGeneratorProps**, and **authenticationSecret**. Even when these properties are encoded, it is possible to recover the original values. The file system on which the properties files, key stores, and trust stores are kept must be protected, so only authorized users can access them. See the WebSphere Application Server documentation for more information.

What to do next

Starting secure servers in WebSphere Application Server

Scenario: Using an OSGi environment to develop and run eXtreme Scale plug-ins

Use these scenarios to complete common tasks in an OSGi environment. For example, the OSGi framework is ideal for starting servers and clients in an OSGi container, which allows you to dynamically add and update WebSphere eXtreme Scale plug-ins to the runtime environment.

Before you begin

Read the “OSGi framework overview” on page 33 topic to learn more about OSGi support and the benefits that it can offer.

About this task

The following scenarios are about building and running dynamic plug-ins, which allows you to dynamically install, start, stop, modify, and uninstall plug-ins. You might also complete another likely scenario, which allows you to use the OSGi framework without dynamic capabilities. You can still package your applications as bundles, which are defined by and communicated through services. These service-based bundles offer multiple benefits, which include more efficient development and deployment capabilities.

Scenario goals

After completing this scenario, you will know how to complete the goals:

- Build eXtreme Scale dynamic plug-ins to use in an OSGi environment.
- Run eXtreme Scale containers in an OSGi environment without dynamic capabilities.

OSGi framework overview

OSGi defines a dynamic module system for Java. The OSGi service platform has a layered architecture, and is designed to run on various standard Java profiles. You can start WebSphere eXtreme Scale servers and clients in an OSGi container.

Benefits of running applications in the OSGi container

WebSphere eXtreme Scale OSGi support allows you to deploy the product in the Eclipse Equinox OSGi framework. Previously, if you wanted to update the plug-ins used by eXtreme Scale, you had to restart the Java virtual machine (JVM) to apply the new versions of the plug-ins. With the dynamic update capability that the OSGi framework provides, now you can update the plug-in classes without

restarting the JVM. These plug-ins are exported by user bundles as services. WebSphere eXtreme Scale accesses the service or services by looking them up the OSGi registry.

eXtreme Scale containers can be configured to start more easily and dynamically using either the OSGi configuration admin service or with OSGi Blueprint. If you want to deploy a new data grid with its placement strategy, you can do so by creating an OSGi configuration or by deploying a bundle with eXtreme Scale descriptor XML files. With OSGi support, application bundles containing eXtreme Scale configuration data can be installed, started, stopped, updated, and uninstalled without restarting the whole system. With this capability, you can upgrade the application without disrupting the data grid.

Plug-in beans and services can be configured with custom shard scopes, allowing sophisticated integration options with other services running in the data grid. Each plug-in can use OSGi Blueprint rankings to verify that every instance of the plug-in is activated is at the correct version. An OSGi-managed bean (MBean) and **xscmd** utility are provided, which allow you to query the eXtreme Scale plug-in OSGi services and their rankings.

This capability allows administrators to quickly recognize potential configuration and administration errors and upgrade the plug-in service rankings in use by eXtreme Scale .

OSGi bundles

To interact with and deploy plug-ins in the OSGi framework, you must use *bundles*. In the OSGi service platform, a bundle is a Java archive (JAR) file that contains Java code, resources, and a manifest that describes the bundle and its dependencies. The bundle is the unit of deployment for an application. The eXtreme Scale product supports the following bundle types:

Server bundle

The server bundle is the `objectgrid.jar` file and is installed with the eXtreme Scale stand-alone server installation and is required for running eXtreme Scale servers and can also be used for running eXtreme Scale clients, or local, in-memory caches. The bundle ID for the `objectgrid.jar` file is `com.ibm.websphere.xs.server_<version>`, where the version is in the format: `<Version>.<Release>.<Modification>`. For example, the server bundle for eXtreme Scale version 7.1.1 is `com.ibm.websphere.xs.server_7.1.1`.

Client bundle

The client bundle is the `ogclient.jar` file and is installed with the eXtreme Scale stand-alone and client installations and is used to run eXtreme Scale clients or local, in-memory caches. The bundle ID for the `ogclient.jar` file is `com.ibm.websphere.xs.client_<version>`, where the version is in the format: `<Version>.<Release>.<Modification>`. For example, the client bundle for eXtreme Scale version 7.1.1 is `com.ibm.websphere.xs.client_7.1.1`.

Limitations

You cannot restart the eXtreme Scale bundle because you cannot restart the object request broker (ORB) or eXtremeIO (XIO). To restart the eXtreme Scale server, you must restart the OSGi framework.

Installing the Eclipse Equinox OSGi framework with Eclipse Gemini for clients and servers

Java

If you want to deploy WebSphere eXtreme Scale in the OSGi framework, then you must set up the Eclipse Equinox Environment.

About this task

The task requires that you download and install the Blueprint framework, which allows you to later configure JavaBeans and expose them as services. The use of services is important because you can expose plug-ins as OSGi services so they can be used by the eXtreme Scale run time environment. The product supports two blueprint containers within the Eclipse Equinox core OSGi framework: Eclipse Gemini and Apache Aries. Use this procedure to set up the Eclipse Gemini container.

Procedure

1. Download Eclipse Equinox SDK Version 3.6.1 or later from the Eclipse website. Create a directory for the Equinox framework, for example: `/opt/equinox`. These instructions refer to this directory as `equinox_root`. Extract the compressed file in the `equinox_root` directory.
2. Download the gemini-blueprint incubation 1.0.0 compressed file from the Eclipse website. Extract the file contents into a temporary directory, and copy the following extracted files to the `equinox_root/plugins` directory:

```
dist/gemini-blueprint-core-1.0.0.jar
dist/gemini-blueprint-extender-1.0.0.jar
dist/gemini-blueprint-io-1.0.0.jar
```

Attention: Depending on the location where you download the compressed Blueprint file, the extracted files might have the extension, `RELEASE.jar`, much like the Spring framework JAR files in the next step. You must verify that the file names match the file references in the `config.ini` file.

3. Download the Spring Framework Version 3.0.5 from the following SpringSource web page: <http://www.springsource.com/download/community>. Extract it into a temporary directory, and copy the following extracted files to the `equinox_root/plugins` directory:

```
org.springframework.aop-3.0.5.RELEASE.jar
org.springframework.asm-3.0.5.RELEASE.jar
org.springframework.beans-3.0.5.RELEASE.jar
org.springframework.context-3.0.5.RELEASE.jar
org.springframework.core-3.0.5.RELEASE.jar
org.springframework.expression-3.0.5.RELEASE.jar
```
4. Download the AOP Alliance Java archive (JAR) file from the SpringSource web page. Copy the `com.springsource.org.aopalliance-1.0.0.jar` to the `equinox_root/plugins` directory.
5. Download the Apache commons logging 1.1.1 JAR file from the SpringSource web page. Copy the `com.springsource.org.apache.commons.logging-1.1.1.jar` file to the `equinox_root/plugins` directory.
6. Download the Luminis OSGi Configuration Admin command-line client. Use this JAR file bundle to manage OSGi administrative configurations. Copy the `net.luminis.cmc-0.2.5.jar` to the `equinox_root/plugins` directory.

7. Download the Apache Felix file installation Version 3.0.2 bundle from the following web page: <http://felix.apache.org/site/index.html>. Copy the `org.apache.felix.fileinstall-3.0.2.jar` file to the `equinox_root/plugins` directory.
8. Create a configuration directory inside `equinox_root/plugins` directory; for example:

```
mkdir equinox_root/plugins/configuration
```

9. Create the following `config.ini` file in the `equinox_root/plugins/configuration` directory, replacing `equinox_root` with the absolute path to your `equinox_root` directory and removing all trailing spaces after the backslash on each line. You must include a blank line at the end of the file; for example:

```
osgi.noShutdown=true
osgi.java.profile.bootdelegation=none
org.osgi.framework.bootdelegation=none
eclipse.ignoreApp=true
osgi.bundles=\
org.eclipse.osgi.services_3.2.100.v20100503.jar@1:start, \
org.eclipse.osgi.util_3.2.100.v20100503.jar@1:start, \
org.eclipse.equinox.cm_1.0.200.v20100520.jar@1:start, \
com.springsource.org.apache.commons.logging-1.1.1.jar@1:start, \
com.springsource.org.aopalliance-1.0.0.jar@1:start, \
org.springframework.aop-3.0.5.RELEASE.jar@1:start, \
org.springframework.asm-3.0.5.RELEASE.jar@1:start, \
org.springframework.beans-3.0.5.RELEASE.jar@1:start, \
org.springframework.context-3.0.5.RELEASE.jar@1:start, \
org.springframework.core-3.0.5.RELEASE.jar@1:start, \
org.springframework.expression-3.0.5.RELEASE.jar@1:start, \
org.apache.felix.fileinstall-3.0.2.jar@1:start, \
net.luminis.cmc-0.2.5.jar@1:start, \
geminiblueprint-core-1.0.0.jar@1:start, \
geminiblueprint-extender-1.0.0.jar@1:start, \
geminiblueprint-io-1.0.0.jar@1:start
```

If you have already set up the environment, you can clean up the Equinox plug-in repository by removing the following directory: `equinox_root\plugins\configuration\org.eclipse.osgi`.

10. Run the following commands to start equinox console.

If you are running a different version of Equinox, then your JAR file name is different from the one in the following example:

```
java -jar plugins\org.eclipse.osgi_3.6.1.R36x_v20100806.jar -console
```

Installing eXtreme Scale bundles

Java

WebSphere eXtreme Scale includes bundles that can be installed into an Eclipse Equinox OSGi framework. These bundles are required to start eXtreme Scale servers or use eXtreme Scale clients in OSGi. You can install the eXtreme Scale bundles using the Equinox console or using the `config.ini` configuration file.

Before you begin

This task assumes that you have installed the following products:

- Eclipse Equinox OSGi framework
- eXtreme Scale stand-alone client or server

About this task

eXtreme Scale includes two bundles. Only one of the following bundles is required in an OSGi framework:

objectgrid.jar

The server bundle is the `objectgrid.jar` file and is installed with the eXtreme Scale stand-alone server installation and is required for running eXtreme Scale servers and can also be used for running eXtreme Scale

clients, or local, in-memory caches. The bundle ID for the `objectgrid.jar` file is `com.ibm.websphere.xs.server_<version>`, where the version is in the format: `<Version>.<Release>.<Modification>`. For example, the server bundle for this release is `com.ibm.websphere.xs.server_8.5.0`.

ogclient.jar

The `ogclient.jar` bundle is installed with the eXtreme Scale stand-alone and client installations and is used to run eXtreme Scale clients or local, in-memory caches. The bundle ID for `ogclient.jar` file is `com.ibm.websphere.xs.client_<version>`, where the version is in the format: `<Version>_<Release>_<Modification>`. For example, the client bundle for this release is `com.ibm.websphere.xs.server_8.5.0`.

For more information about developing eXtreme Scale plug-ins, see the System APIs and Plug-ins topic.

Install the eXtreme Scale client or server bundle into the Eclipse Equinox OSGi framework using the Equinox console:

Procedure

1. Start the Eclipse Equinox framework with the console enabled; for example:

```
java_home/bin/java -jar <equinox_root>/plugins/  
org.eclipse.osgi_3.6.1.R36x_v20100806.jar -console
```

2. Install the eXtreme Scale client or server bundle in the Equinox console:

```
osgi> install file:///<path to bundle>
```

3. Equinox displays the bundle ID for the newly installed bundle:

```
Bundle id is 25
```

4. Start the bundle in the Equinox console, where `<id>` is the bundle ID assigned when the bundle was installed:

```
osgi> start <id>
```

5. Retrieve the service status in the Equinox console to verify that the bundle has started; for example:

```
osgi> ss
```

When the bundle starts successfully, the bundle displays the ACTIVE state; for example:

```
25      ACTIVE      com.ibm.websphere.xs.server_8.5.0
```

Install the eXtreme Scale client or server bundle into the Eclipse Equinox OSGi framework using the `config.ini` file:

Procedure

1. Copy the eXtreme Scale client or server (`objectgrid.jar` or `ogclient.jar`) bundle from the `<wxs_install_root>/ObjectGrid/lib` to the Eclipse Equinox plug-ins directory; for example: `<equinox_root>/plugins`
2. Edit the Eclipse Equinox `config.ini` configuration file, and add the bundle to the `osgi.bundles` property; for example:

```
osgi.bundles=\  
org.eclipse.osgi.services_3.2.100.v20100503.jar@1:start, \  
org.eclipse.osgi.util_3.2.100.v20100503.jar@1:start, \  
org.eclipse.equinox.cm_1.0.200.v20100520.jar@1:start, \  
objectgrid.jar@1:start
```

Important: Verify that a blank line exists after the last bundle name. Each bundle is separated by a comma.

3. Start the Eclipse Equinox framework with the console enabled; for example:


```
java_home/bin/java -jar <equinox_root>/plugins/  
org.eclipse.osgi_3.6.1.R36x_v20100806.jar -console
```

4. Retrieve the service status in the Equinox console to verify that the bundle has started:

```
osgi> ss
```

When the bundle starts successfully, the bundle displays the ACTIVE state; for example:

```
25      ACTIVE      com.ibm.websphere.xs.server_8.5.0
```

Results

The eXtreme Scale server or client bundle is installed and started in your Eclipse Equinox OSGi framework.

Running eXtreme Scale containers with non-dynamic plug-ins in an OSGi environment

If you do not need to use the dynamic capability of an OSGi environment, you can still take advantage of tighter coupling, declarative packaging, and service dependencies that the OSGi framework offers.

Before you begin

1. Develop your application using WebSphere eXtreme Scale APIs and plug-ins.
2. Package the application in one or more OSGi bundles with the appropriate import or export dependencies that are declared in one or more bundle manifests. Ensure that all classes or packages that are required for the plug-ins, agents, data objects, and so on, are exported.

About this task

With dynamic plug-ins, you can upgrade your plug-ins without stopping the grid. To use this capability, the original and new plug-ins must be compatible. If you do not need to update plug-ins, or can afford to stop the grid to upgrade them, then you may not need the complexity of dynamic plug-ins. However, there are still good reasons to run your eXtreme Scale application in an OSGi environment. These reasons include the tighter coupling, declarative package, service dependencies, and so on.

One concern with hosting the grid or client in an OSGi environment without using dynamic plug-ins (more specifically, without declaring the plug-ins using OSGi services) is how the eXtreme Scale bundle loads the plug-in classes. The eXtreme Scale bundle relies on OSGi services to load plug-in classes, which allows the bundle to invoke object methods on classes in other bundles without directly importing the packages of those classes.

When the plug-ins are not made available via OSGi services, the eXtreme Scale bundle must be able to load the plug-in classes directly. Rather than modifying the manifest of the eXtreme Scale bundle to import user classes and packages, create a bundle fragment that adds the necessary package imports. The fragment can also import the classes needed for other non-plug-in user classes, such as data objects and agent classes.

Procedure

1. Create an OSGi fragment that uses the eXtreme Scale bundle (client or server, depending on the intended deployment environment) as its host. The fragment declares dependencies (Import-Package) on all of the packages that one or more plug-ins must load. For example, if you are installing a serializer plug-in whose classes reside in the `com.mycompany.myapp.serializers` package and depends on classes in the `com.mycompany.myapp.common` package, then your fragment META-INF/MANIFEST.MF file resembles the following example:

```
Bundle-ManifestVersion: 2
Bundle-Name: Plug-in fragment for XS serializers
Bundle-SymbolicName: com.mycompany.myapp.myfragment; singleton:=true
Bundle-Version: 1.0.0
Fragment-Host: com.ibm.websphere.xs.server; bundle-version=7.1.1
Manifest-Version: 1.0
Import-Package: com.mycompany.myapp.serializers,
               com.mycompany.myapp.common
...
```

This manifest must be packaged in a fragment JAR file, which in this example is `com.mycompany.myapp.myfragment_1.0.0.jar`.

2. Deploy both the newly created fragment, the eXtreme Scale bundle, and application bundles to your OSGi environment. Now, start the bundles.

Results

You can now test and run your application in the OSGi environment without using OSGi services to load user classes, such as plug-ins and agents.

Administering eXtreme Scale servers and applications in an OSGi environment

Use this topic to install the WebSphere eXtreme Scale server bundle, an optional fragment that allows loading of your application bundles and non-dynamic user classes, such as plug-ins, agents, data objects, and so on.

Before you begin

1. Install and start a supported OSGi framework. Currently Equinox is the only supported OSGi implementation. If your application uses Blueprint, make sure to install and start a supported Blueprint implementation. Apache Aries and Eclipse Gemini are both supported.
2. Open the OSGi console.

Procedure

1. Install the eXtreme Scale server bundle. You must know the file URL of the bundle Java archive (JAR) file. For example:

```
osgi> install file:///home/user1/my0sgiEnv/plugins/objectgrid.jar
Bundle id is 41
```

```
osgi>
```

The eXtreme Scale bundle is now installed, but not yet resolved.

2. If the eXtreme Scale server must load user classes directly, rather than using dynamic plug-ins exposed via OSGi services, then you must also install a user-developed fragment that either provides those classes or imports them. If you are using dynamic plug-ins and not using agents, you can skip this step. Here is an example of how to install a custom fragment:

```
osgi> install file:///home/user1/myOsgiEnv/plugins/myFragment.jar
Bundle id is 42
```

```
osgi> ss
```

```
Framework is launched.
```

```
id State      Bundle
...
41 INSTALLED  com.ibm.websphere.xs.server_7.1.1
42 INSTALLED  com.mycompany.myfragment_1.0.0
```

```
osgi>
```

Now the eXtreme Scale server bundle and the custom fragment that attaches to the bundle are both installed.

3. Start the eXtreme Scale server bundle; for example:

```
osgi> start 41
```

```
osgi> ss
```

```
Framework is launched.
```

```
id State      Bundle
...
41 ACTIVE      com.ibm.websphere.xs.server_7.1.1
                Fragments=42
42 RESOLVED    com.mycompany.myfragment_1.0.0
                Master=41
```

```
osgi>
```

4. Now install and start all user application bundles using the same previously mentioned commands. To start a grid on this server, the server and container definition must be declared using Blueprint, or the application must start the server and container programmatically from a bundle activator or some other mechanism.

Results

The eXtreme Scale server bundle and application are deployed, started, and ready to accept work.

Building and running eXtreme Scale dynamic plug-ins for use in an OSGi environment

All eXtreme Scale plug-ins can be configured for an OSGi environment. The primary benefit of dynamic plug-ins is that they allow you to upgrade them without shutting down the grid. This allows you to evolve an application without restarting the grid container processes.

About this task

WebSphere eXtreme Scale OSGi support allows you to deploy the product in an OSGi framework, such as Eclipse Equinox. Previously, if you wanted to update the plug-ins used by eXtreme Scale, you had to restart the Java virtual machine (JVM) to apply the new versions of the plug-ins. With the dynamic plug-in support provided by eXtreme Scale and the ability to update bundles that the OSGi framework provides, you can now update the plug-in classes without restarting the JVM. These plug-ins are exported by *bundles* as services. WebSphere eXtreme Scale accesses the service by looking up the OSGi registry. In the OSGi service

platform, a bundle is a Java archive (JAR) file that contains Java code, resources, and a manifest that describes the bundle and its dependencies. The bundle is the unit of deployment for an application.

Procedure

1. Build eXtreme Scale dynamic plug-ins.
2. Configure eXtreme Scale plug-ins with OSGi Blueprint.
3. Install and starting OSGi-enabled plug-ins.

Building eXtreme Scale dynamic plug-ins

Java

WebSphere eXtreme Scale includes ObjectGrid and BackingMap plug-ins. These plug-ins are implemented in Java and are configured using the ObjectGrid descriptor XML file. To create a dynamic plug-in that can be dynamically upgraded, they need to be aware of ObjectGrid and BackingMap life cycle events because they might need to complete some actions during an update. Enhancing a plug-in bundle with life cycle callback methods, event listeners, or both allows the plug-in to complete those actions at the appropriate times.

Before you begin

This topic assumes that you have built the appropriate plug-in. For more information about developing eXtreme Scale plug-ins, see the System APIs and plug-ins topic.

About this task

All eXtreme Scale plug-ins apply to either a BackingMap or ObjectGrid instance. Many plug-ins also interact with other plug-ins. For example, a Loader and TransactionCallback plug-in work together to properly interact with a database transaction and the various database JDBC calls. Some plug-ins might also need to cache configuration data from other plug-ins to improve performance.

The BackingMapLifecycleListener and ObjectGridLifecycleListener plug-ins provide life cycle operations for the respective BackingMap and ObjectGrid instances. This process allows plug-ins to be notified when the parent BackingMap or ObjectGrid and their respective plug-ins might be changed. BackingMap plug-ins implement the BackingMapLifecycleListener interface, and ObjectGrid plug-ins implement the ObjectGridLifecycleListener interface. These plug-ins are automatically invoked when the life cycle of the parent BackingMap or ObjectGrid changes. For more information about life cycle plug-ins, see the Managing plug-in life cycles topic.

You can expect to enhance bundles using the life cycle methods or event listeners in the following common tasks:

- Starting and stopping resources, such as threads or messaging subscribers.
- Specifying that a notification occur when peer plug-ins have been updated, allowing direct access to the plug-in and detecting any changes.

Whenever you access another plug-in directly, access that plug-in through the OSGi container to ensure that all parts of the system reference the correct plug-in. If, for example, some component in the application directly references, or caches, an instance of a plug-in, it will maintain its reference to that version of the plug-in, even after that plug-in has been dynamically updated. This behavior can cause application-related problems as well as memory leaks. Therefore, write code that

depends on dynamic plug-ins that obtain its reference using OSGi, getService() semantics. If the application must cache one or more plug-ins, it listens for life cycle events using ObjectGridLifecycleListener and BackingMapLifecycleListener interfaces. The application must also be able to refresh its cache when necessary, in a thread safe manner.

All eXtreme Scale plug-ins used with OSGi must also implement the respective BackingMapPlugin or ObjectGridPlugin interfaces. New plug-ins such as the MapSerializerPlugin interface enforce this practice. These interfaces provide the eXtreme Scale runtime environment and OSGi a consistent interface for injecting state into the plug-in and controlling its life cycle.

Use this task to specify that a notification occurs when peer plug-ins are updated, you might create a listener factory that produces a listener instance.

Procedure

- Update the ObjectGrid plug-in class to implement the ObjectGridPlugin interface. This interface includes methods that allow eXtreme Scale to initialize, set the ObjectGrid instance and destroy the plug-in. See the following code example:

```
package com.mycompany;
import com.ibm.websphere.objectgrid.plugins.ObjectGridPlugin;
...

public class MyTranCallback implements TransactionCallback, ObjectGridPlugin {

    private ObjectGrid og = null;

    private enum State {
        NEW, INITIALIZED, DESTROYED
    }

    private State state = State.NEW;

    public void setObjectGrid(ObjectGrid grid) {
        this.og = grid;
    }

    public ObjectGrid getObjectGrid() {
        return this.og;
    }

    void initialize() {
        // Handle any plug-in initialization here. This is called by
        // eXtreme Scale, and not the OSGi bean manager.
        state = State.INITIALIZED;
    }

    boolean isInitialized() {
        return state == State.INITIALIZED;
    }

    public void destroy() {
        // Destroy the plug-in and release any resources. This
        // can be called by the OSGi Bean Manager or by eXtreme Scale.
        state = State.DESTROYED;
    }

    public boolean isDestroyed() {
        return state == State.DESTROYED;
    }
}
```

- Update the ObjectGrid plug-in class to implement the ObjectGridLifecycleListener interface. See the following code example:

```
package com.mycompany;
import com.ibm.websphere.objectgrid.plugins.ObjectGridLifecycleListener;
import com.ibm.websphere.objectgrid.plugins.ObjectGridLifecycleListener.LifecycleEvent;
...

public class MyTranCallback implements TransactionCallback, ObjectGridPlugin, ObjectGridLifecycleListener{
    public void objectGridStateChanged(LifecycleEvent event) {
        switch(event.getState()) {
            case NEW:
            case DESTROYED:
            case DESTROYING:
            case INITIALIZING:
                break;
            case INITIALIZED:
                // Lookup a Loader or MapSerializerPlugin using
```

```

        // OSGi or directly from the ObjectGrid instance.
        lookupOtherPlugins()
        break;
    case STARTING:
    case PRELOAD:
        break;
    case ONLINE:
        if (event.isWritable()) {
            startupProcessingForPrimary();
        } else {
            startupProcessingForReplica();
        }
        break;
    case QUIESCE:
        if (event.isWritable()) {
            quiesceProcessingForPrimary();
        } else {
            quiesceProcessingForReplica();
        }
        break;
    case OFFLINE:
        shutdownShardComponents();
        break;
    }
}
...
}

```

- Update a BackingMap plug-in. Update the BackingMap plug-in class to implement the BackingMap plu-in interface. This interface includes methods that allow eXtreme Scale to initialize, set the BackingMap instance, and destroy the plug-in. See the following code example:

```

package com.mycompany;
import com.ibm.websphere.objectgrid.plugins.BackingMapPlugin;
...

public class MyLoader implements Loader, BackingMapPlugin {

    private BackingMap bmap = null;

    private enum State {
        NEW, INITIALIZED, DESTROYED
    }

    private State state = State.NEW;

    public void setBackingMap(BackingMap map) {
        this.bmap = map;
    }

    public BackingMap getBackingMap() {
        return this.bmap;
    }
    void initialize() {
        // Handle any plug-in initialization here. This is called by
        // eXtreme Scale, and not the OSGi bean manager.
        state = State.INITIALIZED;
    }
    boolean isInitialized() {
        return state == State.INITIALIZED;
    }

    public void destroy() {
        // Destroy the plug-in and release any resources. This
        // can be called by the OSGi Bean Manager or by eXtreme Scale.
        state = State.DESTROYED;
    }

    public boolean isDestroyed() {
        return state == State.DESTROYED;
    }
}

```

- Update the BackingMap plug-in class to implement the BackingMapLifecycleListener interface. See the following code example:

```

package com.mycompany;

import com.ibm.websphere.objectgrid.plugins.BackingMapLifecycleListener;
import com.ibm.websphere.objectgrid.plugins.BackingMapLifecycleListener.LifecycleEvent;
...

public class MyLoader implements Loader, ObjectGridPlugin, ObjectGridLifecycleListener{
    ...
    public void backingMapStateChanged(LifecycleEvent event) {
        switch(event.getState()) {
            case NEW:
            case DESTROYED:
            case DESTROYING:
            case INITIALIZING:
                break;
        }
    }
}

```

```

    case INITIALIZED:
        // Lookup a MapSerializerPlugin using
        // OSGi or directly from the ObjectGrid instance.
        lookupOtherPlugins()
        break;
    case STARTING:
    case PRELOAD:
        break;
    case ONLINE:
        if (event.isWritable()) {
            startupProcessingForPrimary();
        } else {
            startupProcessingForReplica();
        }
        break;
    case QUIESCE:
        if (event.isWritable()) {
            quiesceProcessingForPrimary();
        } else {
            quiesceProcessingForReplica();
        }
        break;
    case OFFLINE:
        shutdownShardComponents();
        break;
    }
    ...
}

```

Results

By implementing the `ObjectGridPlugin` or `BackingMapPlugin` interface, eXtreme Scale can control the life cycle of your plug-in at the right times.

By implementing the `ObjectGridLifecycleListener` or `BackingMapLifecycleListener` interface, the plug-in is automatically registered as a listener of the associated `ObjectGrid` or `BackingMap` life cycle events. The `INITIALIZING` event is used to signal that all of the `ObjectGrid` and `BackingMap` plug-ins have been initialized and are available for lookup and use. The `ONLINE` event is used to signal that the `ObjectGrid` is online and ready to start processing events.

Configuring eXtreme Scale plug-ins with OSGi Blueprint

Java

All eXtreme Scale `ObjectGrid` and `BackingMap` plug-ins can be defined as OSGi beans and services using the OSGi Blueprint Service available with Eclipse Gemini or Apache Aries.

Before you begin

Before you can configure your plug-ins as OSGi services, you must first package your plug-ins in an OSGi bundle, and understand the fundamental principles of the required plug-ins. The bundle must import the WebSphere eXtreme Scale server or client packages and other dependent packages required by the plug-ins, or create a bundle dependency on the eXtreme Scale server or client bundles. This topic describes how to configure the Blueprint XML to create plug-in beans and expose them as OSGi services for eXtreme Scale to use.

About this task

Beans and services are defined in a Blueprint XML file, and the Blueprint container discovers, creates, and wires the beans together and exposes them as services. The process makes the beans available to other OSGi bundles, including the eXtreme Scale server and client bundles.

When creating custom plug-in services for use with eXtreme Scale, the bundle that is to host the plug-ins, must be configured to use Blueprint. In addition, a Blueprint XML file must be created and stored within the bundle. Read about building OSGi applications with the Blueprint Container specification for a general understanding of the specification.

Procedure

1. Create a Blueprint XML file. You can name the file anything. However, you must include the blueprint namespace:

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">
  ...
</blueprint>
```

2. Create bean definitions in the Blueprint XML file for each eXtreme Scale plug-in.

Beans are defined using the `<bean>` element and can be wired to other bean references and can include initialization parameters.

Important: When defining a bean, you must use the correct scope. Blueprint supports the singleton and prototype scopes. eXtreme Scale also supports a custom shard scope.

Define most eXtreme Scale plug-ins as prototype or shard-scoped beans, since all of the beans must be unique for each ObjectGrid shard or BackingMap instance it is associated with. Shard-scoped beans can be useful when using the beans in other contexts to allow retrieving the correct instance.

To define a prototype-scoped bean, use the `scope="prototype"` attribute on the bean:

```
<bean id="myPluginBean" class="com.mycompany.MyBean" scope="prototype">
  ...
</bean>
```

To define a shard-scoped bean, you must add the `objectgrid` namespace to the XML schema, and use the `scope="objectgrid:shard"` attribute on the bean:

```
<?xml version="1.0" encoding="UTF-8"?>

<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
  xmlns:objectgrid="http://www.ibm.com/schema/objectgrid"

  xsi:schemaLocation="http://www.ibm.com/schema/objectgrid
    http://www.ibm.com/schema/objectgrid/objectgrid.xsd">

  <bean id="myPluginBean" class="com.mycompany.MyBean"
scope="objectgrid:shard">
    ...
  </bean>

  ...
```

3. Create `PluginServiceFactory` bean definitions for each plug-in bean. All eXtreme Scale beans must have a `PluginServiceFactory` bean defined so that the correct bean scope can be applied. eXtreme Scale includes a `BlueprintServiceFactory` that you can use. It includes two properties that must be set. You must set the `blueprintContainer` property to the `blueprintContainer` reference, and the `beanId` property must be set to the bean identifier name. When eXtreme Scale looks up the service to instantiate the appropriate beans, the server looks up the bean component instance using the Blueprint container.

```

bean id="myPluginBeanFactory"
  class="com.ibm.websphere.objectgrid.plugins.osgi.BluePrintServiceFactory">
  <property name="blueprintContainer" ref="blueprintContainer" />
  <property name="beanId" value="myPluginBean" />
</bean>

```

4. Create a service manager for each PluginServiceFactory bean. Each service manager exposes the PluginServiceFactory bean, using the <service> element. The service element identifies the name to expose to OSGi, the reference to the PluginServiceFactory bean, the interface to expose, and the ranking of the service. eXtreme Scale uses the service manager ranking to perform service upgrades when the eXtreme Scale grid is active. If the ranking is not specified, the OSGi framework assumes a ranking of 0. Read about updating service rankings for more information.

Blueprint includes several options for configuring service managers. To define a simple service manager for a PluginServiceFactory bean, create a <service> element for each PluginServiceFactory bean:

```

<service ref="myPluginBeanFactory"
  interface="com.ibm.websphere.objectgrid.plugins.osgi.PluginServiceFactory"
  ranking="1">
</service>

```

5. Store the Blueprint XML file in the plug-ins bundle. The Blueprint XML file must be stored in the OSGI-INF/blueprint directory for the Blueprint container to be discovered.

To store the Blueprint XML file in a different directory, you must specify the following Bundle-Blueprint manifest header:

```
Bundle-Blueprint: OSGI-INF/blueprint.xml
```

Results

The eXtreme Scale plug-ins are now configured to be exposed in an OSGi Blueprint container. In addition, the ObjectGrid descriptor XML file is configured to reference the plug-ins using the OSGi Blueprint service.

Installing and starting OSGi-enabled plug-ins

In this task, you install the dynamic plug-in bundle into the OSGi framework. Then, you start the plug-in.

Before you begin

This topic assumes that the following tasks have been completed:

- The eXtreme Scale server or client bundle has been installed into the Eclipse Equinox OSGi framework. See “Installing eXtreme Scale bundles” on page 220.
- One or more dynamic BackingMap or ObjectGrid plug-ins have been implemented. See “Building eXtreme Scale dynamic plug-ins” on page 225.
- The dynamic plug-ins have been packaged as OSGi services in OSGi bundles.

About this task

This task describes how to install the bundle using the Eclipse Equinox console. The bundle can be installed using several different methods, including modifying the config.ini configuration file. Products that embed Eclipse Equinox include alternative methods for managing bundles. For more information on how to add bundles in the config.ini file in Eclipse Equinox, see the Eclipse runtime options.

OSGi allows bundles to be started that have duplicate services. WebSphere eXtreme Scale uses the latest service ranking. When starting multiple OSGi frameworks in an eXtreme Scale data grid, you must make sure that the correct service rankings are started on each server. Failure to do so causes the grid to be started with a mixture of different versions.

To see which versions are in-use by the data grid, use the `xscmd` utility to check the current and available rankings. For more information about available service rankings see [Updating OSGi services for eXtreme Scale plug-ins with `xscmd`](#).

Procedure

Install the plug-in bundle into the Eclipse Equinox OSGi framework using the OSGi console.

1. Start the Eclipse Equinox framework with the console enabled; for example:

```
<java_home>/bin/java -jar <equinox_root>/plugins/org.eclipse.osgi_3.6.1.R36x_v20100806.jar -console
```

2. Install the plug-in bundle in the Equinox console.

```
osgi> install file:///<path to bundle>
```

Equinox displays the bundle ID for the newly installed bundle:

```
Bundle id is 17
```

3. Enter the following line to start the bundle in the Equinox console, where `<id>` is the bundle ID assigned when the bundle was installed:

```
osgi> start <id>
```

4. Retrieve the service status in the Equinox console to verify that the bundle has started:

```
osgi> ss
```

When the bundle has started successfully, the bundle displays the ACTIVE state; for example:

```
17      ACTIVE      com.mycompany.plugin.bundle_VRM
```

Install the plug-in bundle into the Eclipse Equinox OSGi framework using the `config.ini` file.

5. Copy the plug-in bundle into the Eclipse Equinox plug-ins directory; for example:

```
<equinox_root>/plugins
```

6. Edit the Eclipse Equinox `config.ini` configuration file, and add the bundle to the `osgi.bundles` property; for example:

```
osgi.bundles=\
org.eclipse.osgi.services_3.2.100.v20100503.jar@1:start, \
org.eclipse.osgi.util_3.2.100.v20100503.jar@1:start, \
org.eclipse.equinox.cm_1.0.200.v20100520.jar@1:start, \
com.mycompany.plugin.bundle_VRM.jar@1:start
```

Important: Verify there is a blank line after the last bundle name. Each bundle is separated by a comma.

7. Start the Eclipse Equinox framework with the console enabled; for example:

```
<java_home>/bin/java -jar <equinox_root>/plugins/org.eclipse.osgi_3.6.1.R36x_v20100806.jar -console
```

8. Retrieve the service status in the Equinox console to verify that the bundle has started; for example:

```
osgi> ss
```

When the bundle has started successfully, the bundle displays the ACTIVE state; for example:

```
17      ACTIVE      com.mycompany.plugin.bundle_VRM
```

Results

The plug-in bundle is now installed and started. The eXtreme Scale container or client can now be started. For more information on developing eXtreme Scale plug-ins, see the System APIs and Plug-ins topic.

Running eXtreme Scale containers with dynamic plug-ins in an OSGi environment

If your application is hosted in the Eclipse Equinox OSGi framework with Eclipse Gemini or Apache Aries, then you can use this task to help you install and configure your WebSphere eXtreme Scale application in OSGi.

Before you begin

Before you start this task, be sure to complete the following tasks:

- Install the Eclipse Equinox OSGi framework with Eclipse Gemini
- Build and run eXtreme Scale dynamic plug-ins for use in an OSGi environment

About this task

With dynamic plug-ins, you can dynamically upgrade the plug-in while the grid is still active. This allows you to update an application without restarting the grid container processes. For more information about developing eXtreme Scale plug-ins, see System APIs and Plug-ins.

Procedure

1. Configure OSGi-enabled plug-ins using the ObjectGrid descriptor XML file.
2. Start eXtreme Scale container servers using the Eclipse Equinox OSGi framework.
3. Administer OSGi services for eXtreme Scale plug-ins with the xscmd utility.
4. Configure servers with OSGi Blueprint.

Configuring OSGi-enabled plug-ins using the ObjectGrid descriptor XML file

Java

In this task, you add existing OSGi services to a descriptor XML file so that WebSphere eXtreme Scale containers can recognize and load the OSGi-enabled plug-ins correctly.

Before you begin

To configure your plug-ins, be sure to:

- Create your package, and enable dynamic plug-ins for OSGi deployment.
- Have the names of the OSGi services that represent your plug-ins available.

About this task

You have created an OSGi service to wrap your plug-in. Now, these services must be defined in the `objectgrid.xml` file so that eXtreme Scale containers can load and configure the plug-in or plug-ins successfully.

Procedure

1. Any grid-specific plug-ins, such as `TransactionCallback`, must be specified under the `objectGrid` element. See the following example from the `objectgrid.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>

<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">

  <objectGrids>
    <objectGrid name="MyGrid" txTimeout="60">
      <bean id="myTranCallback" osgiService="myTranCallbackFactory"/>
      ...
    </objectGrid>
    ...
  </objectGrids>
  ...
</objectGridConfig>
```

Important: The `osgiService` attribute value must match the `ref` attribute value that is specified in the blueprint XML file, where the service was defined for `myTranCallback PluginServiceFactory`.

2. Any map-specific plug-ins, such as loaders or serializers, for example, must be specified in the `backingMapPluginCollections` element and referenced from the `backingMap` element. See the following example from the `objectgrid.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>

objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
    <objectGrid name="MyGrid" txTimeout="60">
      <backingMap name="MyMap1" lockStrategy="PESSIMISTIC"
        copyMode="COPY_TO_BYTES" nullValuesSupported="false"
        pluginCollectionRef="myPluginCollectionRef1"/>
      <backingMap name="MyMap2" lockStrategy="PESSIMISTIC"
        copyMode="COPY_TO_BYTES" nullValuesSupported="false"
        pluginCollectionRef="myPluginCollectionRef2"/>
      ...
    </objectGrid>
    ...
  </objectGrids>
  ...
  <backingMapPluginCollections>
    <backingMapPluginCollection id="myPluginCollectionRef1">
      <bean id="MapSerializerPlugin" osgiService="mySerializerFactory"/>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="myPluginCollectionRef2">
      <bean id="MapSerializerPlugin" osgiService="myOtherSerializerFactory"/>
      <bean id="Loader" osgiService="myLoader"/>
    </backingMapPluginCollection>
    ...
  </backingMapPluginCollections>
  ...
</objectGridConfig>
```

Results

The `objectgrid.xml` file in this example tells eXtreme Scale to create a grid called `MyGrid` with two maps, `MyMap1` and `MyMap2`. The `MyMap1` map uses the serializer wrapped by the OSGi service, `mySerializerFactory`. The `MyMap2` map uses a serializer from the OSGi service, `myOtherSerializerFactory`, and a loader from the OSGi service, `myLoader`.

Starting eXtreme Scale servers using the Eclipse Equinox OSGi framework

WebSphere eXtreme Scale container servers can be started in an Eclipse Equinox OSGi framework using several methods.

Before you begin

Before you can start an eXtreme Scale container, you must have completed the following tasks:

1. The WebSphere eXtreme Scale server bundle must be installed into Eclipse Equinox.
2. Your application must be packaged as an OSGi bundle.
3. Your WebSphere eXtreme Scale plug-ins (if any) must be packaged as an OSGi bundle. They can be bundled in the same bundle as your application or as separate bundles.
4. If your container servers are using IBM eXtremeMemory, you must first configure the native libraries. For more information, see [Configuring IBM eXtremeMemory](#).

About this task

This task describes how to start an eXtreme Scale container server in an Eclipse Equinox OSGi framework. You can use any of the following methods to start container servers using the Eclipse Equinox implementation:

- OSGi Blueprint service

You can include all configuration and metadata in an OSGi bundle. See the following image to understand the Eclipse Equinox process for this method:

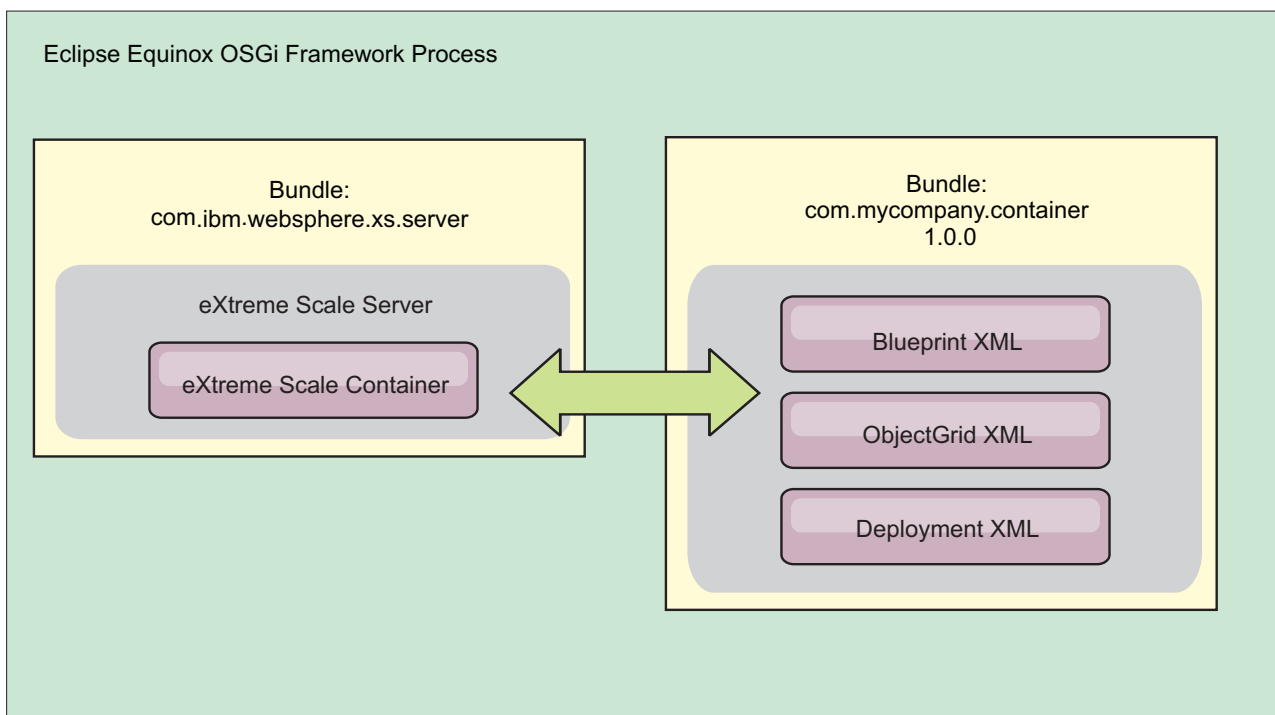


Figure 65. Eclipse Equinox process for including all configuration and metadata in an OSGi bundle

- OSGi Configuration Admin service

You can specify configuration and metadata outside of an OSGi bundle. See the following image to understand the Eclipse Equinox process for this method:

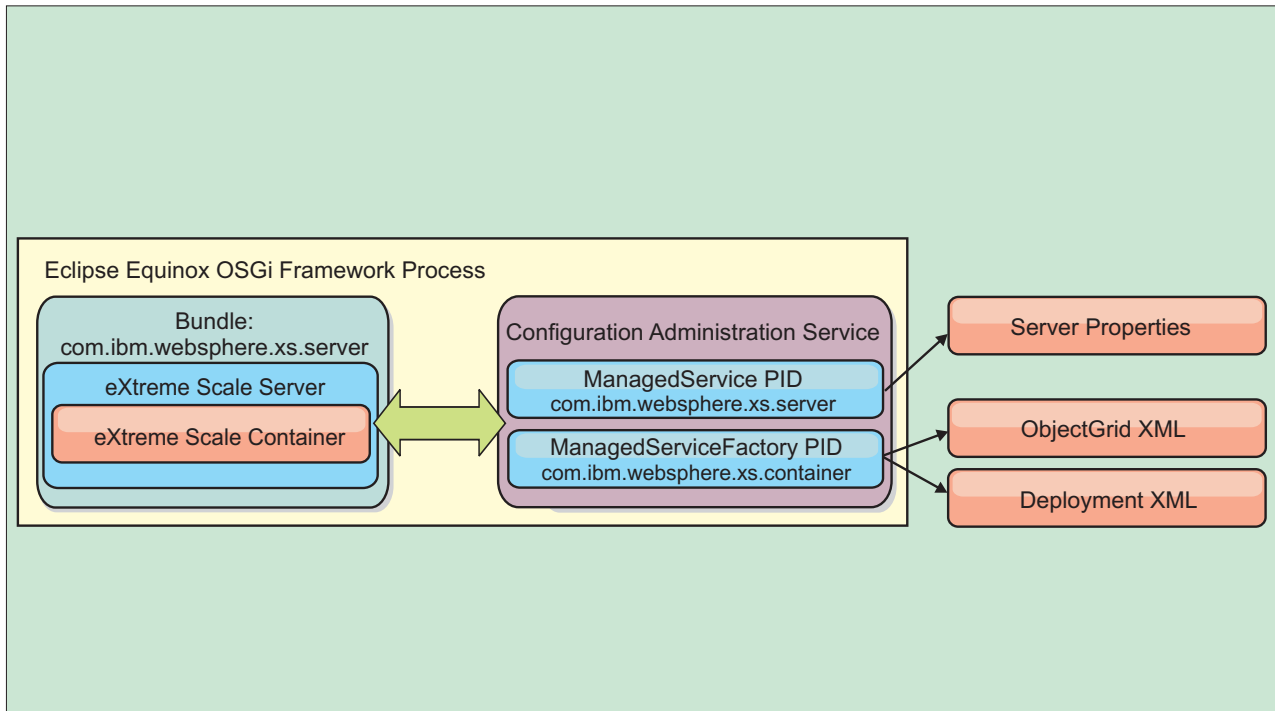


Figure 66. Eclipse Equinox process for specify configuration and metadata outside of an OSGi bundle

- Programmatically
Supports customized configuration solutions.

In each case, an eXtreme Scale server singleton is configured and one or more containers are configured.

The eXtreme Scale server bundle, `objectgrid.jar`, includes all of the required libraries to start and run an eXtreme Scale grid container in an OSGi framework. The server runtime environment communicates with user-supplied plug-ins and data objects using the OSGi service manager.

Important: After an eXtreme Scale server bundle is started and the eXtreme Scale server is initialized, it cannot be restarted. The Eclipse Equinox process must be restarted to restart an eXtreme Scale server.

You can use eXtreme Scale support for Spring namespace to configure eXtreme Scale container servers in a Blueprint XML file. When the server and container XML elements are added to the Blueprint XML file, the eXtreme Scale namespace handler automatically starts a container server using the parameters that are defined in the Blueprint XML file when the bundle is started. The handle stops the container when the bundle is stopped.

To configure eXtreme Scale container servers with Blueprint XML, complete the following steps:

Procedure

- Start an eXtreme Scale container server using OSGi blueprint.
 1. Create a container bundle.
 2. Install the container bundle into the Eclipse Equinox OSGi framework. See “Installing and starting OSGi-enabled plug-ins” on page 230.
 3. Start the container bundle.
- Start an eXtreme Scale container server using OSGi configuration admin.
 1. Configure the server and container using config admin.
 2. When the eXtreme Scale server bundle is started, or the persistent identifiers are created with config admin, the server and container automatically start.
- Start an eXtreme Scale container server using the ServerFactory API. See the server API documentation.
 1. Create an OSGi bundle activator class, and use the eXtreme Scale ServerFactory API to start a server.

Administering OSGi-enabled services using the xscmd utility

You can use the **xscmd** utility to complete administrator tasks, such as viewing services and their rankings that are being used by each container, and updating the runtime environment to use new versions of the bundles.

About this task

With the Eclipse Equinox OSGi framework, you can install multiple versions of the same bundle, and you can update those bundles during run time. WebSphere eXtreme Scale is a distributed environment that runs the container servers in many OSGi framework instances.

Administrators are responsible for manually copying, installing, and starting bundles into the OSGi framework. eXtreme Scale includes an OSGi ServiceTrackerCustomizer to track any services that have been identified as eXtreme Scale plug-ins in the ObjectGrid descriptor XML file. Use the **xscmd** utility to validate which version of the plug-in is used, which versions are available to be used, and to perform bundle upgrades.

eXtreme Scale uses the service ranking number to identify the version of each service. When two or more services are loaded with the same reference, eXtreme Scale automatically uses the service with the highest ranking.

Procedure

- Run the **osgiCurrent** command, and verify that each eXtreme Scale server is using the correct plug-in service ranking.

Since eXtreme Scale automatically chooses the service reference with the highest ranking, it is possible that the data grid may start with multiple rankings of a plug-in service.

If the command detects a mismatch of rankings or if it is unable to find a service, a non-zero error level is set. If the command completed successfully then the error level is set to 0.

The following example shows the output of the **osgiCurrent** command when two plug-ins are installed in the same grid on four servers. The loaderPlugin plug-in is using ranking 1, and the txCallbackPlugin is using ranking 2.

```
OSGi Service Name Current Ranking ObjectGrid Name MapSet Name Server Name
-----
```

loaderPlugin	1	MyGrid	MapSetA	server1
--------------	---	--------	---------	---------

loaderPlugin	1	MyGrid	MapSetA	server2
loaderPlugin	1	MyGrid	MapSetA	server3
loaderPlugin	1	MyGrid	MapSetA	server4
txCallbackPlugin	2	MyGrid	MapSetA	server1
txCallbackPlugin	2	MyGrid	MapSetA	server2
txCallbackPlugin	2	MyGrid	MapSetA	server3
txCallbackPlugin	2	MyGrid	MapSetA	server4

The following example shows the output of the **osgiCurrent** command when server2 was started with a newer ranking of the loaderPlugin:

```
OSGi Service Name Current Ranking ObjectGrid Name MapSet Name Server Name
-----
```

loaderPlugin	1	MyGrid	MapSetA	server1
loaderPlugin	2	MyGrid	MapSetA	server2
loaderPlugin	1	MyGrid	MapSetA	server3
loaderPlugin	1	MyGrid	MapSetA	server4
txCallbackPlugin	2	MyGrid	MapSetA	server1
txCallbackPlugin	2	MyGrid	MapSetA	server2
txCallbackPlugin	2	MyGrid	MapSetA	server3
txCallbackPlugin	2	MyGrid	MapSetA	server4

- Run the **osgiAll** command to verify that the plug-in services have been correctly started on each eXtreme Scale container server.

When bundles start that contain services that an ObjectGrid configuration is referencing, the eXtreme Scale runtime environment automatically tracks the plug-in, but does not immediately use it. The **osgiAll** command shows which plug-ins are available for each server.

When run without any parameters, all services are shown for all grids and servers. Additional filters, including the **-serviceName** <service_name> filter can be specified to limit the output to a single service or a subset of the data grid.

The following example shows the output of the **osgiAll** command when two plug-ins are started on two servers. The loaderPlugin has both rankings 1 and 2 started and the txCallbackPlugin has ranking 1 started. The summary message at the end of the output confirms that both servers see the same service rankings:

```
Server: server1
OSGi Service Name Available Rankings
-----
```

loaderPlugin	1, 2
txCallbackPlugin	1

```
Server: server2
OSGi Service Name Available Rankings
-----
```

loaderPlugin	1, 2
txCallbackPlugin	1

Summary - All servers have the same service rankings.

The following example shows the output of the **osgiAll** command when the bundle that includes the loaderPlugin with ranking 1 is stopped on server1. The summary message at the bottom of the output confirms that server1 is now missing the loaderPlugin with ranking 1:

```
Server: server1
OSGi Service Name Available Rankings
-----
```

loaderPlugin	2
txCallbackPlugin	1

```
Server: server2
OSGi Service Name Available Rankings
-----
```

loaderPlugin	1, 2
txCallbackPlugin	1

Summary - The following servers are missing service rankings:

```
Server  OSGi Service Name Missing Rankings
-----  -----
server1 loaderPlugin      1
```

The following example shows the output if the service name is specified with the **-sn** argument, but the service does not exist:

```
Server: server2
  OSGi Service Name Available Rankings
  -----
  invalidPlugin      No service found
```

```
Server: server1
  OSGi Service Name Available Rankings
  -----
  invalidPlugin      No service found
```

Summary - All servers have the same service rankings.

- Run the **osgiCheck** command to check sets of plug-in services and rankings to see if they are available.

The **osgiCheck** command accepts one or more sets of service rankings in the form: **-serviceRankings <service name>;<ranking>[,<serviceName>;<ranking>]**

When the rankings are all available, the method returns with an error level of 0. If one or more rankings are not available, a non-zero error level is set. A table of all of the servers that do not include the specified service rankings is displayed. Additional filters can be used to limit the service check to a subset of the available servers in the eXtreme Scale domain.

For example, if the specified ranking or service is absent, the following message is displayed:

```
Server  OSGi Service Unavailable Rankings
-----  -----
server1 loaderPlugin 3
server2 loaderPlugin 3
```

- Run the **osgiUpdate** command to update the ranking of one or more plug-ins for all servers in a single ObjectGrid and MapSet in a single operation.

The command accepts one or more sets of service rankings in the form: **-serviceRankings <service name>;<ranking>[,<serviceName>;<ranking>] -g <grid name> -ms <mapset name>**

With this command, you can complete the following operations:

- Verify that the specified services are available for updating on each of the servers.
- Change the state of the grid to offline using the StateManager interface. See Managing ObjectGrid availability for more information. This process quiesces the grid and waits until any running transactions have completed and prevents any new transactions from starting. This process also signals any ObjectGridLifecycleListener and BackingMapLifecycleListener plug-ins to discontinue any transactional activity. See Plug-ins for providing event listeners for information about event listener plug-ins.
- Update each eXtreme Scale container running in an OSGi framework to use the new service versions.
- Changes the state of the grid to online, allowing transactions to continue.

The update process is idempotent so that if a client fails to complete any one task, it results in the operation being rolled back. If a client is unable to perform the rollback or is interrupted during the update process, the same command can be issued again, and it continues at the appropriate step.

If the client is unable to continue, and the process is restarted from another client, use the `-force` option to allow the client to perform the update. The `osgiUpdate` command prevents multiple clients from updating the same map set concurrently. For more details about the `osgiUpdate` command, see Updating OSGi services for eXtreme Scale plug-ins with `xscmd`.

Configuring servers with OSGi Blueprint

Java

You can configure WebSphere eXtreme Scale container servers using an OSGi blueprint XML file, allowing simplified packaging and development of self-contained server bundles.

Before you begin

This topic assumes that the following tasks have been completed:

- The Eclipse Equinox OSGi framework has been installed and started with either the Eclipse Gemini or Apache Aries blueprint container.
- The eXtreme Scale server bundle has been installed and started.
- The eXtreme Scale dynamic plug-ins bundle has been created.
- The eXtreme Scale ObjectGrid descriptor XML file and deployment policy XML file have been created.

About this task

This task describes how to configure an eXtreme Scale server with a container using a blueprint XML file. The result of the procedure is a container bundle. When the container bundle is started, the eXtreme Scale server bundle will track the bundle, parse the server XML and start a server and container.

A container bundle can optionally be combined with the application and eXtreme Scale plug-ins when dynamic plug-in updates are not required or the plug-ins do not support dynamic updating.

Procedure

1. Create a Blueprint XML file with the `objectgrid` namespace included. You can name the file anything. However, it must include the blueprint namespace:

```
<?xml version="1.0" encoding="UTF-8"?>

<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:objectgrid="http://www.ibm.com/schema/objectgrid"
  xsi:schemaLocation="http://www.ibm.com/schema/objectgrid
  http://www.ibm.com/schema/objectgrid/objectgrid.xsd">
  ...
</blueprint>
```

2. Add the XML definition for the eXtreme Scale server with the appropriate server properties. See the Spring descriptor XML file for details on all available configuration properties. See the following example of the XML definition:

```
<objectgrid:server id="xsServer" tracespec="ObjectGridOSGi=all=enabled"
  tracefile="logs/osgi/wxserver/trace.log" jmxport="1199" listenerPort="2909">
  <objectgrid:catalog host="catserver1.mycompany.com" port="2809" />
  <objectgrid:catalog host="catserver2.mycompany.com" port="2809" />
</objectgrid:server>
```

3. Add the XML definition for the eXtreme Scale container with the reference to the server definition and the ObjectGrid descriptor XML and ObjectGrid deployment XML files embedded in the bundle; for example:

```
<objectgrid:container id="container"
  objectgridxml="/META-INF/objectGrid.xml"
  deploymentxml="/META-INF/objectGridDeployment.xml"
  server="xsServer" />
```

4. Store the Blueprint XML file in the container bundle. The Blueprint XML must be stored in the OSGI-INF/blueprint directory for the Blueprint container to be found.

To store the Blueprint XML in a different directory, you must specify the Bundle-Blueprint manifest header; for example:

```
Bundle-Blueprint: OSGI-INF/blueprint.xml
```

5. Package the files into a single bundle JAR file. See the following example of a bundle directory hierarchy:

```
MyBundle.jar
  /META-INF/manifest.mf
  /META-INF/objectGrid.xml
  /META-INF/objectGridDeployment.xml
  /OSGI-INF/blueprint/blueprint.xml
```

Results

An eXtreme Scale container bundle is now created and can be installed in Eclipse Equinox. When the container bundle is started, the eXtreme Scale server runtime environment in the eXtreme Scale server bundle, will automatically start the singleton eXtreme Scale server using the parameters defined in the bundle, and starts a container server. The bundle can be stopped and started, which results in the container stopping and starting. The server is a singleton and does not stop when the bundle is started the first time.

Scenario: Using JCA to connect transactional applications to eXtreme Scale clients

Java

The following scenario is about connecting clients to applications that participate in transactions.

Before you begin

Read the Transaction processing in the Java EE applications overview topic to learn more about transaction support.

About this task

The Java EE Connector Architecture (JCA) provides support for clients that are using Java Transaction API (JTA). Through JTA, client management is simplified and accomplished using Java Platform, Enterprise Edition (Java EE). The JCA specification also supports resource adapters that you can use to connect applications to eXtreme Scale clients. A resource adapter is a system-level software driver that a Java application uses to connect to an enterprise information system (EIS). A resource adapter plugs into an application server and provides

connectivity between the EIS, the application server, and the enterprise application. WebSphere eXtreme Scale provides its own resource adapter, which you can install without any required configuration.

As with previous versions of the product, you can use transactions to process a single unit of work to the data grid. With the support of JCA, when you commit those transactions you can enlist resources for that transaction in one-phase commit, which has the following benefits:

- Simplified eXtreme Scale application development. Previously, developers coordinated eXtreme Scale transactions with resources, such as enterprise beans, servlets, and web containers. Because no rollback mechanism existed, developers had no simple way to recover failures.
- Tighter integration exists with WebSphere Application Server, which includes last participant support to enlist in global transactions if necessary.

Scenario goals

After completing this scenario, you will know how to complete the following goals:

- Use Java Transaction API (JTA) support to develop application components that use transactions.
- Connect your applications with eXtreme Scale clients.

Transaction processing in Java EE applications

WebSphere eXtreme Scale provides its own resource adapter that you can use to connect applications to the data grid and process local transactions.

Through support from the eXtreme Scale resource adapter, Java Platform, Enterprise Edition (Java EE) applications can look up eXtreme Scale client connections and demarcate local transactions using Java EE local transactions or using the eXtreme Scale APIs. When the resource adapter is configured, you can complete the following actions with your Java EE applications:

- Look up or inject eXtreme Scale resource adapter connection factories within a Java EE application component.
- Obtain standard connection handles to the eXtreme Scale client and share them between application components using Java EE conventions.
- Demarcate eXtreme Scale transactions using either the `javax.resource.cci.LocalTransaction` API or the `com.ibm.websphere.objectgrid.Session` interface.
- Use the entire eXtreme Scale client API, such as the `ObjectMap` API and `EntityManager` API.

The following additional capabilities are available with WebSphere Application Server:

- Enlist eXtreme Scale connections with a global transaction as a last participant with other two-phase commit resources. The eXtreme Scale resource adapter provides local transaction support, with a single-phase commit resource. With WebSphere Application Server, your applications can enlist one, single-phase commit resource into a global transaction through last participant support.
- Automatic resource adapter installation when the profile is augmented.
- Automatic security principal propagation.

Administrator responsibilities

The eXtreme Scale resource adapter is installed into the Java EE application server or embedded with the application. After you install the resource adapter, the administrator creates one or more resource adapter connection factories for each catalog service domain and optionally each data grid instance. The connection factory identifies the properties that are required to communicate with the data grid.

Applications reference the connection factory, which establishes the connection to the remote data grid. Each connection factory hosts a single eXtreme Scale client connection that is reused for all application components.

Important: Because the eXtreme Scale client connection might include a near cache, applications must not share a connection. A connection factory must exist for a single application instance to avoid problems sharing objects between applications.

The connection factory hosts an eXtreme Scale client connection, which is shared between all referencing application components. You can use a managed bean (MBean) to access information about the client connection or to reset the connection when it is no longer needed.

Application developer responsibilities

An application developer creates resource references for managed connection factories in the application deployment descriptor or with annotations. Each resource reference includes a local reference for the eXtreme Scale connection factory, as well as the resource-sharing scope.

Important: Enabling resource sharing is important because it allows the local transaction to be shared between application components.

Applications can inject the connection factory into the Java EE application component, or it can look up the connection factory using Java Naming Directory Interface (JNDI). The connection factory is used to obtain connection handles to the eXtreme Scale client connection. The eXtreme Scale client connection is managed independently from the resource adapter connection and is established on first use, and reused for all subsequent connections.

After finding the connection, the application retrieves an eXtreme Scale session reference. With the eXtreme Scale session reference, the application can use the entire eXtreme Scale client APIs and features.

You can demarcate transactions in one of the following ways:

- Use the `com.ibm.websphere.objectgrid.Session` transaction demarcation methods.
- Use the `javax.resource.cci.LocalTransaction` local transaction.
- Use a global transaction, when you use WebSphere Application Server with last participant support enabled. When you select this approach for demarcation, you must:
 - Use an application-managed global transaction with the `javax.transaction.UserTransaction`.
 - Use a container-managed transaction.

Application deployer responsibilities

The application deployer binds the local reference to the resource adapter connection factory that the application developer defines to the resource adapter connection factories that the administrator defines. The application deployer must assign the correct connection factory type and scope to the application and ensure that the connection factory is not shared between applications to avoid Java object sharing. The application deployer is also responsible for configuring and mapping other appropriate configuration information that is common to all connection factories.

Installing an eXtreme Scale resource adapter

The WebSphere eXtreme Scale resource adapter is Java Connector Architecture (JCA) 1.5 compatible and can be installed on a Java 2 Platform, Enterprise Edition (J2EE) 1.5 1.6 or later, or on an application server such as WebSphere Application Server.

Before you begin

The resource adapter is in the `wxsra.rar` resource adapter archive (RAR) file, which is available in all installations of eXtreme Scale. The RAR file is in the following directories:

- For WebSphere Application Server installations: `wxs_install_root/optionalLibraries/ObjectGrid`
- For stand-alone installations: `wxs_install_root/ObjectGrid/lib` directory

The resource adapter is coupled with the eXtreme Scale runtime environment. It requires the eXtreme Scale runtime JAR files in the correct classpath. In general, you can upgrade the eXtreme Scale runtime environment without updating the resource adapter. Upgrading the eXtreme Scale runtime environment also upgrades the resource adapter runtime environment. The resource adapter supports version 8.5 and up to two versions later of the eXtreme Scale runtime environment. Later versions of the resource adapter might require later versions of the eXtreme Scale runtime environment as they become available.

The `wxsra.rar` file requires one of the eXtreme Scale client runtime JAR files to operate. For details about which client runtime JAR file is appropriate, see Runtime files for WebSphere eXtreme Scale stand-alone installation and Runtime files for WebSphere eXtreme Scale integrated with WebSphere Application Server, which include details about the available runtime JAR files.

About this task

You can install the eXtreme Scale resource adapter using several options that allow for flexible deployment scenarios. The resource adapter can be embedded with the Java Platform, Enterprise Edition (Java EE) application, or it can be installed as a stand-alone RAR file that is shared between applications.

Embedding the resource adapter with the application simplifies deployment because connection factories are only created within the scope of the application and cannot be shared between applications. With the resource adapter embedded in the application, you can also embed the cache objects and ObjectGrid client plug-in classes within the application. Embedding the resource adapter also protects the application from inadvertently sharing cache objects between applications, which can result in `java.lang.ClassCastException` exceptions.

By installing the wxsra.rar file as a stand-alone resource adapter, you can create resource manager connection factories at the node scope. This option is useful in the following situations:

- When it is not practical to embed the wxsra.rar file inside the application
- When the version of eXtreme Scale is not known at build time
- When you want to share an eXtreme Scale client connection with multiple applications

Important: In multiple versions of WebSphere Application Server, up to Version 8.0.2, you cannot install the eXtreme Scale resource adapter in an application EAR file and in the stand-alone server simultaneously. The result, when you use the enterprise archive (EAR) file that also has the RAR file installed, is that the application experiences an exception, such as `ClassCastException`: `com.ibm.websphere.xs.ra.XSConnectionFactory` incompatible with `com.ibm.websphere.xs.ra.XSConnectionFactory`. The following example WebSphere Application Server message and call stack for this error are displayed when a servlet encounters this exception:

```
SRVE0068E: An exception was thrown by one of the service methods of the servlet [ClientServlet]
in application [JTASampleClientEAR]. Exception created : [java.lang.ClassCastException:
com.ibm.websphere.xs.ra.XSConnectionFactory incompatible with com.ibm.websphere.xs.ra.XSConnectionFactory
at com.ibm.websphere.xs.sample.jtasample.WXSClientServlet.connectClient(WXSClientServlet.java:484)
at com.ibm.websphere.xs.sample.jtasample.WXSClientServlet.doGet(WXSClientServlet.java:200)
at javax.servlet.http.HttpServlet.service(HttpServlet.java:575)
at javax.servlet.http.HttpServlet.service(HttpServlet.java:668)
at com.ibm.ws.webcontainer.servlet.ServletWrapper.service(ServletWrapper.java:1214)
at com.ibm.ws.webcontainer.servlet.ServletWrapper.handleRequest(ServletWrapper.java:774)
at com.ibm.ws.webcontainer.servlet.ServletWrapper.handleRequest(ServletWrapper.java:456)
```

Procedure

- **Install an embedded eXtreme Scale resource adapter.** When the wxsra.rar file is embedded in the application EAR file, the resource adapter must have access to the eXtreme Scale runtime libraries.

For applications that run in WebSphere Application Server, the following choices and subsequent actions are available:

Option	Description
If eXtreme Scale is integrated with the WebSphere Application Server node	The runtime library files are already available in the system classpath, and no other action is required.
If eXtreme Scale is not integrated with the WebSphere Application Server node	You must include the wsogclient.jar file in the wxsra.rar classpath.

For applications that do not run in WebSphere Application Server, the client runtime library file, `ogclient.jar`, or the server runtime library file, `objectgrid.jar`, must be in the classpath of the RAR file.

- **Install a stand-alone eXtreme Scale resource adapter.** When you install the wxsra.rar file as a stand-alone resource adapter, it must have access to the eXtreme Scale runtime libraries.

For applications that run in WebSphere Application Server, the following choices and subsequent actions are available:

Option	Description
If eXtreme Scale is integrated with the WebSphere Application Server node	The runtime library files are already available in the system classpath, and no other action is required.
If eXtreme Scale is not integrated with the WebSphere Application Server node	You must include the wsogclient.jar file in the wxsra.rar classpath.

For applications that do not run in WebSphere Application Server, the client runtime library file, `ogclient.jar`, or the server runtime library file, `objectgrid.jar`, must be in the classpath of the RAR file.

1. Give the resource adapter access to any shared classes. All ObjectGrid plug-in classes and the applications that use them must share a class loader. Since the resource adapter is shared by multiple applications, all classes must be accessible by the same class loader. You can create this access by using a shared library between all applications that interact with the resource adapter.

What to do next

Now that you have installed the eXtreme Scale resource adapter, you can configure connection factories so that your Java EE applications can connect to a remote eXtreme Scale data grid.

Configuring eXtreme Scale connection factories

Java

An eXtreme Scale connection factory allows Java EE applications to connect to a remote WebSphere eXtreme Scale data grid. Use custom properties to configure resource adapters.

Before you begin

Before you create the connection factories, you must install the resource adapter.

About this task

After you install the resource adapter, you can create one or more resource adapter connection factories that represent eXtreme Scale client connections to remote data grids. Complete the following steps to configure a resource adapter connection factory and use it within an application.

You can create an eXtreme Scale connection factory at the node scope for stand-alone resource adapters or within the application for embedded resource adapters. See the related topics for information about how to create connection factories in WebSphere Application Server.

Procedure

1. Using the WebSphere Application Server administrative console to create an eXtreme Scale connection factory that represents an eXtreme Scale client connection. See [Configuring Java EE Connector connection factories](#) in the administrative console. After you specify properties for the connection factory in the General Properties panel, you must click **Apply** for the Custom properties link to become active.
2. Click **Custom properties** in the administrative console. Set the following custom properties to configure the client connection to the remote data grid.

Table 10. Custom properties for configuring connection factories

Property Name	Type	Description
ConnectionName	String	(Optional) The name of the eXtreme Scale client connection. The ConnectionName helps identify the connection when exposed as a managed bean. This property is optional. If not specified, the ConnectionName is undefined.
CatalogServiceEndpoints	String	(Optional) The catalog service domain end points in the format: <host>:<port>[,<host><port>]. For more information, see Catalog service domain settings. This property is required if the catalog service domain is not set.
CatalogServiceDomain	String	(Optional) The catalog service domain name that is defined in WebSphere Application Server. For more information, see Configuring catalog servers and catalog service domains. This property is required if the CatalogServiceEndpoints property is not set.
ObjectGridName	String	(Optional) The name of the data grid that this connection factory connects to. If not specified, then the application must supply the name when obtaining the connection from the connection factory.
ObjectGridURL	String	(Optional) The URL of the client data grid, override XML file. This property is not valid if the ObjectGridResource is also specified. For more information, see Configuring Java clients.
ObjectGridResource	String	The resource path of the client data grid, override XML file. This property is optional and invalid if ObjectGridURL is also specified. For more information, see Configuring Java clients.
ClientPropertiesURL	String	(Optional) The URL of the client properties file. This property is not valid if the ClientPropertiesResource is also specified. For more information, see Client properties file for more information.
ClientPropertiesResource	String	(Optional) The resource path of the client properties file. This property is not valid if the ClientPropertiesURL is also specified. For more information, see Client properties file for more information.

WebSphere Application Server also allows other configuration options for adjusting connection pools and managing security. See the related information for links to WebSphere Application Server Information Center topics.

What to do next

Create an eXtreme Scale connection factory reference in the application. See “Configuring applications to connect with eXtreme Scale” on page 247 for more information.

Configuring Eclipse environments to use eXtreme Scale connection factories

Java

The eXtreme Scale resource adapter includes custom connection factories. To use these interfaces in your eXtreme Scale Java Platform, Enterprise Edition (Java EE) applications, you must import the wxsra.rar file into your workspace and link it to your application project.

Before you begin

- You must install Rational® Application Developer Version 7 or later or Eclipse Java EE IDE for Web Developers Version 1.4 or later.
- A server runtime environment must be configured.

Procedure

1. Import the wxsra.rar file into your project by selecting **File > Import**. The Import window is displayed.

2. Select **Java EE > RAR file**. The Connector Import window is displayed.
3. To specify the connector file, click **Browse** to locate the `wxsra.rar` file. The `wxsra.rar` file is installed when you install a resource adapter. You can find the resource adapter archive (RAR) file in the following location:
 - For WebSphere Application Server installations: `wxs_install_root/optionalLibraries/ObjectGrid`
 - For stand-alone installations: `wxs_install_root/ObjectGrid/lib` directory
4. Create a name for the new connector project in the **Connector project** field. You can use `wxsra`, which is the default name.
5. Choose a Target runtime, which references a Java EE server runtime environment.
6. Optionally select **Add project to EAR** to embed the RAR into an existing EAR project.

Results

The RAR file is now imported into your Eclipse workspace.

What to do next

You can reference the RAR project from your other Java EE projects using the following steps:

1. Right click on the project and click **Properties**.
2. Select **Java Build Path**.
3. Select the Projects tab.
4. Click **Add**.
5. Select the `wxsra` connector project, and click **OK**.
6. Click **OK** again to close the Properties window.

The eXtreme Scale resource adapter classes are now in the classpath. To install product runtime JAR files using the Eclipse console, see Setting up a stand-alone development environment in Eclipse for more information.

Configuring applications to connect with eXtreme Scale

Applications use an eXtreme Scale connection factory to create connection handles to an eXtreme Scale client connection. You can configure resource adapter connection factory references using this task.

Before you begin

Create a Java Platform, Enterprise Edition (Java EE) application component, such as an Enterprise JavaBeans (EJB) container or servlet.

Procedure

Create a `javax.resource.cci.ConnectionFactory` resource reference in the application component. Resource references are declared in the deployment descriptor by the application provider. The connection factory represents an eXtreme Scale client connection that can be used to communicate with one or more named data grids that are available in the catalog service domain.

Securing J2C client connections

Use the Java 2 Connector (J2C) architecture to secure connections between WebSphere eXtreme Scale clients and your applications.

About this task

Applications reference the connection factory, which establishes the connection to the remote data grid. Each connection factory hosts a single eXtreme Scale client connection that is reused for all application components.

Important: Since the eXtreme Scale client connection might include a near cache, it is important that applications do not share a connection. A connection factory must exist for a single application instance to avoid problems sharing objects between applications.

You can set the credential generator with the API or in the client properties file. In the client properties file, the `securityEnabled` and `credentialGenerator` properties are used.

Attention: In the following example, some lines of code are continued on the next line for publication purposes.

```
securityEnabled=true
credentialGeneratorClass=com.ibm.websphere.objectgrid.security.plugins.builtins.
    UserPasswordCredentialGenerator
credentialGeneratorProps=operator XXXXXX
```

The credential generator and credential in the client properties file are used for the eXtreme Scale connect operation and the default J2C credentials. Therefore, the credentials that are specified with the API are used at J2C connect time for the J2C connection. However, if no credentials are specified at J2C connect time, then the credential generator in the client properties file is used.

Procedure

1. Set up secure access where the J2C connection represents the eXtreme Scale client. Use the `ClientPropertiesResource` connection factory property or the `ClientPropertiesURL` connection factory property to configure client authentication.

If you are using WebSphere eXtreme Scale with WebSphere Application Server, then specify the client properties on the catalog service domain configuration. When the connection factory references the domain, it automatically uses this configuration.

2. Configure the client security properties to use the connection factory that references the appropriate credential generator object for eXtreme Scale. These properties are also compatible with eXtreme Scale server security. For example, use the `WSTokenCredentialGenerator` credential generator for WebSphere credentials when eXtreme Scale is installed with WebSphere Application Server. Alternatively, use the `UserPasswordCredentialGenerator` credential generator when you run the eXtreme Scale in a stand-alone environment. In the following example, credentials are passed programmatically using the API call instead of using the configuration in the client properties:

```
XSConnectionSpec spec = new XSConnectionSpec();
spec.setCredentialGenerator(new UserPasswordCredentialGenerator("operator", "xxxxxx"));
Connection conn = connectionFactory.getConnection(spec);
```

3. (Optional) Disable the near cache, if required.

All J2C connections from a single connection factory share a single near cache. Grid entry permissions and map permissions are validated on the server, but

not on the near cache. When an application uses multiple credentials to create J2C connections, and the configuration uses specific permissions for grid entries and maps for those credentials, then disable the near cache. Disable the near cache using the connection factory property, ObjectGridResource or ObjectGridURL. For more information about disabling the near cache, see Configuring the near cache.

4. (Optional) Set security policy settings, if required.

If the J2EE application contains the embedded eXtreme Scale resource adapter archive (RAR) file configuration, you might be required to set additional security policy settings in the security policy file for the application. For example, these policies are required:

```
permission com.ibm.websphere.security.WebSphereRuntimePermission "accessRuntimeClasses";
permission java.lang.RuntimePermission "accessDeclaredMembers";
permission javax.management.MBeanTrustPermission "register";
permission java.lang.RuntimePermission "getClassLoader";
```

Additionally, any property or resource files used by connection factories require file or other permissions, such as `permission java.io.FilePermission "filePath";` For WebSphere Application Server, the policy file is `META-INF/was.policy`, and it is located in the J2EE EAR file.

Results

The client security properties that you configured on the catalog service domain are used as default values. The values that you specify override any properties that are defined in the `client.properties` files.

What to do next

Use eXtreme Scale data access APIs to develop client components that you want to use transactions.

Developing eXtreme Scale client components to use transactions

Java

The WebSphere eXtreme Scale resource adapter provides client connection management and local transaction support. With this support, Java Platform, Enterprise Edition (Java EE) applications can look up eXtreme Scale client connections and demarcate local transactions with Java EE local transactions or the eXtreme Scale APIs.

Before you begin

Create an eXtreme Scale connection factory resource reference.

About this task

There are several options for working with eXtreme Scale data access APIs. In all cases, the eXtreme Scale connection factory must be injected into the application component, or looked up in Java Naming Directory Interface (JNDI). After the connection factory is looked up, you can demarcate transactions and create connections to access the eXtreme Scale APIs.

You can optionally cast the `javax.resource.cci.ConnectionFactory` instance to a `com.ibm.websphere.xs.ra.XSConnectionFactory` that provides additional options for retrieving connection handles. The resulting connection handles must be cast to the `com.ibm.websphere.xs.ra.XSConnection` interface, which provides the `getSession` method. The `getSession` method returns a `com.ibm.websphere.objectgrid.Session` object handle that allows applications to use any of the eXtreme Scale data access APIs, such as the `ObjectMap` API and `EntityManager` API.

The Session handle and any derived objects are valid for the life of the `XSConnection` handle.

The following procedures can be used to demarcate eXtreme Scale transactions. You cannot mix each of the procedures. For example, you cannot mix global transaction demarcation and local transaction demarcation in the same application component context.

Procedure

- Use autocommit, local transactions. Use the following steps to automatically commit data access operations or operations that do not support an active transaction:
 1. Retrieve a `com.ibm.websphere.xs.ra.XSConnection` connection outside of the context of a global transaction.
 2. Retrieve and use the `com.ibm.websphere.objectgrid.Session` session to interact with the data grid.
 3. Invoke any data access operation that supports autocommit transactions.
 4. Close the connection.
- Use an `ObjectGrid` session to demarcate a local transaction. Use the following steps to demarcate an `ObjectGrid` transaction using the `Session` object:
 1. Retrieve a `com.ibm.websphere.xs.ra.XSConnection` connection.
 2. Retrieve the `com.ibm.websphere.objectgrid.Session` session.
 3. Use the `Session.begin()` method to start the transaction.
 4. Use the session to interact with the data grid.
 5. Use the `Session.commit()` or `rollback()` methods to end the transaction.
 6. Close the connection.
- Use a `javax.resource.cci.LocalTransaction` transaction to demarcate a local transaction. Use the following steps to demarcate an `ObjectGrid` transaction using the `javax.resource.cci.LocalTransaction` interface:
 1. Retrieve a `com.ibm.websphere.xs.ra.XSConnection` connection.
 2. Retrieve the `javax.resource.cci.LocalTransaction` transaction using the `XSConnection.getLocalTransaction()` method.
 3. Use the `LocalTransaction.begin()` method to start the transaction.
 4. Retrieve and use the `com.ibm.websphere.objectgrid.Session` session to interact with the data grid.
 5. Use the `LocalTransaction.commit()` or `rollback()` methods to end the transaction.
 6. Close the connection.
- Enlist the connection in a global transaction. This procedure also applies to container-managed transactions:
 1. Begin the global transaction through the `javax.transaction.UserTransaction` interface or with a container-managed transaction.

2. Retrieve a `com.ibm.websphere.xs.ra.XSConnection` connection.
 3. Retrieve and use the `com.ibm.websphere.objectgrid.Session` session.
 4. Close the connection.
 5. Commit or roll back the global transaction.
- **8.6+** Configure a connection to write multiple partitions in a transaction. Use the following steps to demarcate an `ObjectGrid` transaction using the `Session` object:
 1. Create a new `com.ibm.websphere.xs.ra.XSConnectionSpec` object.
 2. Call the `XSConnectionSpec` method and the `setMultiPartitionSupportEnabled` method with an argument of `true`.
 3. Retrieve the `com.ibm.websphere.xs.ra.XSConnection` connection to pass the `XSConnectionSpec` to the `ConnectionFactory.getConnection` method.
 4. Retrieve and use the `com.ibm.websphere.objectgrid.Session` session.

Example

See the following code example, which demonstrates the previous steps for demarcating eXtreme Scale transactions.

```
// (C) Copyright IBM Corp. 2001, 2012.
// All Rights Reserved. Licensed Materials - Property of IBM.
package com.ibm.ws.xs.ra.test.ee;

import javax.naming.InitialContext;
import javax.resource.cci.Connection;
import javax.resource.cci.ConnectionFactory;
import javax.resource.cci.LocalTransaction;
import javax.transaction.Status;
import javax.transaction.UserTransaction;

import junit.framework.TestCase;

import com.ibm.websphere.objectgrid.ObjectMap;
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.xs.ra.XSConnection;

/**
 * This sample requires that it runs in a J2EE context in your
 * application server. For example, using the JUnitEE framework servlet.
 *
 * The code in these test methods would typically reside in your own servlet,
 * EJB, or other web component.
 *
 * The sample depends on a configured WebSphere eXtreme Scale connection
 * factory registered at of JNDI Name of "eis/embedded/wxscf" that defines
 * a connection to a grid containing a Map with the name "Map1".
 *
 * The sample does a direct lookup of the JNDI name and does not require
 * resource injection.
 */
public class DocSampleTests extends TestCase {
    public final static String CF_JNDI_NAME = "eis/embedded/wxscf";
    public final static String MAP_NAME = "Map1";

    Long          key = null;
    Long          value = null;
    InitialContext ctx = null;
    ConnectionFactory cf = null;

    public DocSampleTests() {
    }
    public DocSampleTests(String name) {
        super(name);
    }
    protected void setUp() throws Exception {
        ctx = new InitialContext();
        cf = (ConnectionFactory)ctx.lookup(CF_JNDI_NAME);
        key = System.nanoTime();
        value = System.nanoTime();
    }
}
```

```

}
/**
 * This example runs when not in the context of a global transaction
 * and uses autocommit.
 */
public void testLocalAutocommit() throws Exception {
    Connection conn = cf.getConnection();
    try {
        Session session = ((XSCConnection)conn).getSession();
        ObjectMap map = session.getMap(MAP_NAME);
        map.insert(key, value); // Or various data access operations
    }
    finally {
        conn.close();
    }
}

/**
 * This example runs when not in the context of a global transaction
 * and demarcates the transaction using session.begin()/session.commit()
 */
public void testLocalSessionTransaction() throws Exception {
    Session session = null;
    Connection conn = cf.getConnection();
    try {
        session = ((XSCConnection)conn).getSession();
        session.begin();
        ObjectMap map = session.getMap(MAP_NAME);
        map.insert(key, value); // Or various data access operations
        session.commit();
    }
    finally {
        if (session != null && session.isTransactionActive()) {
            try { session.rollback(); }
            catch (Exception e) { e.printStackTrace(); }
        }
        conn.close();
    }
}

/**
 * This example uses the LocalTransaction interface to demarcate
 * transactions.
 */
public void testLocalTranTransaction() throws Exception {
    LocalTransaction tx = null;
    Connection conn = cf.getConnection();
    try {
        tx = conn.getLocalTransaction();
        tx.begin();
        Session session = ((XSCConnection)conn).getSession();
        ObjectMap map = session.getMap(MAP_NAME);
        map.insert(key, value); // Or various data access operations
        tx.commit(); tx = null;
    }
    finally {
        if (tx != null) {
            try { tx.rollback(); }
            catch (Exception e) { e.printStackTrace(); }
        }
        conn.close();
    }
}

/**
 * This example depends on an externally managed transaction,
 * the externally managed transaction might typically be present in
 * an EJB with its transaction attributes set to REQUIRED or REQUIRES_NEW.
 * NOTE: If there is NO global transaction active, this example runs in auto-commit
 * mode because it doesn't verify a transaction exists.
 */
public void testGlobalTransactionContainerManaged() throws Exception {
    Connection conn = cf.getConnection();
    try {
        Session session = ((XSCConnection)conn).getSession();
        ObjectMap map = session.getMap(MAP_NAME);
        map.insert(key, value); // Or various data access operations
    }
    catch (Throwable t) {

```

```

        t.printStackTrace();
        UserTransaction tx = (UserTransaction)ctx.lookup("java:comp/UserTransaction");
        if (tx.getStatus() != Status.STATUS_NO_TRANSACTION) {
            tx.setRollbackOnly();
        }
    }
    finally {
        conn.close();
    }
}

/**
 * This example demonstrates starting a new global transaction using the
 * UserTransaction interface. Typically the container starts the global
 * transaction (for example in an EJB with a transaction attribute of
 * REQUIRES_NEW), but this sample will also start the global transaction
 * using the UserTransaction API if it is not currently active.
 */
public void testGlobalTransactionTestManaged() throws Exception {
    boolean started = false;
    UserTransaction tx = (UserTransaction)ctx.lookup("java:comp/UserTransaction");
    if (tx.getStatus() == Status.STATUS_NO_TRANSACTION) {
        tx.begin();
        started = true;
    }
    // else { called with an externally/container managed transaction }
    Connection conn = null;
    try {
        conn = cf.getConnection(); // Get connection after the global tran starts
        Session session = ((XSCConnection)conn).getSession();
        ObjectMap map = session.getMap(MAP_NAME);
        map.insert(key, value); // Or various data access operations
        if (started) {
            tx.commit(); started = false; tx = null;
        }
    }
    finally {
        if (started) {
            try { tx.rollback(); }
            catch (Exception e) { e.printStackTrace(); }
        }
        if (conn != null) { conn.close(); }
    }
}

/**
 * This example demonstrates a multi-partition transaction.
 */

public void testGlobalTransactionTestManagedMultiPartition() throws Exception {
    boolean started = false;
    XSCConnectionSpec connSpec = new XSCConnectionSpec();
    connSpec.setWriteToMultiplePartitions(true);
    UserTransaction tx = (UserTransaction)ctx.lookup("java:comp/UserTransaction");
    if (tx.getStatus() == Status.STATUS_NO_TRANSACTION) {
        tx.begin();
        started = true;
    }
    // else { called with an externally/container managed transaction }
    Connection conn = null;
    try {
        conn = cf.getConnection(connSpec); // Get connection after the global tran starts
        Session session = ((XSCConnection)conn).getSession();
        ObjectMap map = session.getMap(MAP_NAME);
        map.insert(key, value); // Or various data access operations
        if (started) {
            tx.commit(); started = false; tx = null;
        }
    }
    finally {
        if (started) {
            try { tx.rollback(); }
            catch (Exception e) { e.printStackTrace(); }
        }
        if (conn != null) { conn.close(); }
    }
}
}

```

Administering J2C client connections

Java

The WebSphere eXtreme Scale connection factory includes an eXtreme Scale client connection that can be shared between applications and persisted through application restarts.

About this task

The client connection includes a management bean that provides connection status information and lifecycle management operations.

Procedure

Maintain client connections. When the first connection is obtained from the `XSCConnectionFactory` connection factory object, an eXtreme Scale client connection is established to the remote data grid and the `ObjectGridJ2CConnection` MBean is created. The client connection is maintained for the life of the process. To end a client connection, invoke one of the following events::

- Stop the resource adapter. A resource adapter can be stopped, for example, when it is embedded in an application and the application is stopped.
- Invoke the `resetConnection` MBean operation on the `ObjectGridJ2CConnection` MBean. When the connection is reset, all connections are invalidated, transactions completed, and the `ObjectGrid` client connection is destroyed. Subsequent calls to the `getConnection` methods on the connection factory result in a new client connection.

WebSphere Application Server also provides additional management beans for managing J2C connections, monitoring connection pools, and performance.

Scenario: Configuring HTTP session failover in the Liberty profile

You can configure a web application server so that, when the web server receives an HTTP request for session replication, the request is forwarded to one or more servers that run in the Liberty profile.

Before you begin

To complete this task, you must install the Liberty profile. For more information, see [Installing the Liberty profile](#).

About this task

The Liberty profile does not include session replication. However, if you use WebSphere eXtreme Scale with the Liberty profile, then you can replicate sessions. Therefore, if a server fails, then application users do not lose session data.


When you add the `webApp` feature to the server definition and configure the session manager, you can use session replication in your eXtreme Scale applications that run in the Liberty profile.

Enabling the eXtreme Scale web feature in the Liberty profile

Java

You can enable the web feature to use HTTP session failover in the Liberty profile.

About this task

 The web feature is deprecated. Use the webApp feature instead. When you add the webApp feature to the server definition and configure the session manager, you can use session replication in your WebSphere eXtreme Scale applications that run in the Liberty profile.

When you install the WebSphere Application Server Liberty profile, it does not include session replication. However, if you use WebSphere eXtreme Scale with the Liberty profile, then you can replicate sessions so that if a server goes down, the application users do not lose session data.

When you add the web feature to the server definition and configure the session manager, you can use session replication in your eXtreme Scale applications that run in the Liberty profile.

Procedure

Define a web application to run in the Liberty profile.

What to do next

Next, configure a web server plug-in to forward HTTP requests to multiple servers in the Liberty profile.

Enabling the eXtreme Scale webGrid feature in the Liberty profile

Use the webGrid feature to automatically start a container to host the clients for HTTP session replication in the Liberty profile.

About this task

When you install the WebSphere Application Server Liberty profile, it does not include session replication. However, if you use WebSphere eXtreme Scale with the Liberty profile, then you can replicate sessions so that if a server goes down, the application users do not lose session data.

When you add the webGrid feature to the server definition and configure the session manager, you can use session replication in your eXtreme Scale applications that run in the Liberty profile.

Procedure

Add the following webGrid feature to the Liberty profile `server.xml` file. The webGrid feature includes the client feature and the server feature. You likely want to separate your web applications from the data grids. For example, you have one Liberty profile server for your web applications and a different Liberty profile server for hosting the data grid.

```
<featureManager>
<feature>eXtremeScale.webGrid-1.1</feature>
</featureManager>
```

Results

Your web applications can now persist its session data in a WebSphere eXtreme Scale grid.

Example

The webGrid feature has meta type properties that you can set on the xsWebGrid element of the server.xml file. See the following example of a server.xml file, which contains the webGrid feature that you use when you connect to the data grid remotely.

```
<server description="Airport Entry eXtremeScale Getting Started Client Web Server">
<!--
This sample program is provided AS IS and may be used, executed, copied and modified
without royalty payment by customer
(a) for its own instruction and study,
(b) in order to develop applications designed to run with an IBM WebSphere product,
either for customer's own internal use or for redistribution by customer, as part of such an
application, in customer's own products.
Licensed Materials - Property of IBM
5724-X67, 5655-V66 (C) COPYRIGHT International Business Machines Corp. 2012
-->
<!-- Enable features -->
<featureManager>
<feature>eXtremeScale.webGrid-1.1</feature>
</featureManager>

<xsServer catalogServer="true"/>

<xsWebGrid objectGridName="session" catalogHostPort="remoteHost:2809" securityEnabled="false" />

</server>
```

Enabling the eXtreme Scale webApp feature in the Liberty profile

A Liberty profile server can host a data grid that caches data for applications to replicate HTTP session data for fault tolerance.

About this task

When you install the WebSphere Application Server Liberty profile, it does not include session replication. However, if you use WebSphere eXtreme Scale with the Liberty profile, then you can replicate sessions so that if a server goes down, the application users do not lose session data.

When you add the webApp feature to the server definition and configure the session manager, you can use session replication in your eXtreme Scale applications that run in the Liberty profile.

Procedure

Add the following webApp feature to the Liberty profile server.xml file. The webApp feature includes the client feature; however, it does not include the server feature. You likely want to separate your web applications from the data grids. For example, you have one Liberty profile server for your web applications and a different Liberty profile server for hosting the data grid.

```
<featureManager>
<feature>eXtremeScale.webapp-1.1</feature>
</featureManager>
```

Results

Your web applications can now persist its session data in a WebSphere eXtreme Scale grid.

Example

See the following example of a server.xml file, which contains the webApp feature that you use when you connect to the data grid remotely.

```
<server description="Airport Entry eXtremeScale Getting Started Client Web Server">
<!--
This sample program is provided AS IS and may be used, executed, copied and modified
without royalty payment by customer
(a) for its own instruction and study,
(b) in order to develop applications designed to run with an IBM WebSphere product,
either for customer's own internal use or for redistribution by customer, as part of such an
application, in customer's own products.
Licensed Materials - Property of IBM
5724-X67, 5655-V66 (C) COPYRIGHT International Business Machines Corp. 2012
-->
<!-- Enable features -->
<featureManager>
<feature>eXtremeScale.webapp-1.1</feature>
</featureManager>

<httpEndpoint id="defaultHttpEndpoint"
host="*"
httpPort="{default.http.port}"
httpsPort="{default.https.port}" />

<xsWebApp objectGridName="session" catalogHostPort="remoteHost:2809" securityEnabled="false" />
</server>
```

What to do next

The webApp feature has meta type properties that you can set on the xsWebApp element of the server.xml file. For more information, see Liberty profile xsWebApp feature properties.

Configuring a web server plug-in to forward requests to multiple servers in the Liberty profile

Java

Use this task to configure the web server plug-in to distribute HTTP server requests between multiple servers in the Liberty profile.

Before you begin

Before you configure the web server plug-in to route HTTP requests to multiple server, complete the following task:

- “Enabling the eXtreme Scale webApp feature in the Liberty profile” on page 256

About this task

Configure the web server plug-in so that the web server receives an HTTP request for dynamic resources, the request is forwarded to multiple servers that run in the Liberty profile.

Procedure

See *Configuring the Liberty profile with a web server plug-in in the WebSphere Application Server Information Center* to complete this task.

What to do next

Next, merge the `plugin-cfg.xml` files from multiple application server cells. You must also ensure that unique clone IDs exist for each application server that runs in the Liberty profile.

Merging plug-in configuration files for deployment to the application server plug-in

Java

Generate plug-in configuration files after you configure a unique clone ID in the Liberty `server.xml` configuration file.

Before you begin

If you are generating and merging plug-in configuration files to configure HTTP session failover in a Liberty profile, then you must complete the following tasks:

- “Enabling the eXtreme Scale web feature in the Liberty profile” on page 255
- “Configuring a web server plug-in to forward requests to multiple servers in the Liberty profile” on page 257

About this task

Use the WebSphere Application Server administrative console to complete this task.

Procedure

1. Merge the `plugin-cfg.xml` files from multiple application server cells. You can either manually merge the `plugin-cfg.xml` files or use the `pluginCfgMerge` tool to automatically merge the `plugin-cfg.xml` file from multiple application server profiles into a single output file. The `pluginCfgMerge.bat` and `pluginCfgMerge.sh` files are in the `install_root/bin` directory.

For more information about manually merging the `plugin-cfg.xml` files, see the technote about merging `plugin-cfg.xml` files from multiple application server profiles.

2. Ensure that the `cloneID` value for each application server is unique. Examine the `cloneID` value for each application server in the merged file to ensure that this value is unique for each application server. If the `cloneID` values in the merged file are not all unique, or if you are running with memory to memory session replication in peer to peer mode, use the administrative console to configure unique HTTP session `cloneIDs`.

To configure a unique HTTP session clone ID with the WebSphere Application Server administrative console, complete the following steps:

- a. Click **Servers > Server Types > WebSphere application servers > *server_name***.
- b. Under Container Settings, click **Web Container Settings > Web container**.
- c. Under Additional Properties, click **Custom properties > New**.

- d. Enter `HttpSessionCloneId` in the **Name** field, and enter a unique value for the server in the **Value** field. The unique value must be eight to nine alphanumeric characters in length. For example, `test1234` is a valid `cloneID` value.
 - e. Click **Apply** or **OK**.
 - f. Click **Save** to save the configuration changes to the master configuration.
3. Copy the merged `plugin-cfg.xml` file to the `plugin_installation_root/config/web_server_name` directory on the web server host.
 4. Ensure that you defined the correct operating system, file access permissions for the merged `plugin-cfg.xml` file. These file access permissions allow the HTTP server plug-in process to read the file.

Results

When you complete this task, you have one plug-in configuration file for multiple application server cells, and your eXtreme Scale applications that run in the Liberty profile are enabled for session replication.

Scenario: Running grid servers in the Liberty profile using Eclipse tools

You can use Eclipse tools to run WebSphere eXtreme Scale servers in the WebSphere Application Server Liberty profile. The Eclipse tools offer a convenient way of running your servers in the same Eclipse environment where you develop, configure, and deploy your eXtreme Scale applications.

About this task

With the Eclipse tools, you can configure eXtreme Scale servers to run in the Liberty profile. If you complete this task manually, you add the supported Liberty features to the `server.xml` file. However, when you use the Eclipse tools, you can complete this task and other development tasks using Eclipse Java EE IDE for Web Developers, Version: Indigo Service Release 1.

Installing the Liberty profile developer tools for WebSphere eXtreme Scale

Eclipse provides a graphical user interface (GUI) that you can use to run WebSphere eXtreme Scale servers in the Liberty profile. To use this GUI, you must install WebSphere eXtreme Scale Version 8.5 Liberty profile tools.

About this task

You can install the tools using one of the following methods:

- Install from Eclipse Marketplace. Click **Help** > **Eclipse Marketplace**.
- Install by dragging an **Install** icon to a running workbench. This option is only available for installing the developer tools on Eclipse IDE for Java EE Developers 3.7, or later.

You must install IBM WebSphere Application Server V8.5 Liberty Profile Developer Tools to use IBM WebSphere eXtreme Scale V8.5 Liberty Profile Developer Tools. Therefore, the steps in this task include the installation of both developer tools.

Procedure

- Install from Eclipse Marketplace.
 1. Start your Eclipse workbench.
 2. Click Help > Eclipse Marketplace.
 3. In the Find field, type WebSphere.
 4. In the list of results, locate **IBM WebSphere Application Server V8.5 Liberty Profile Developer Tools**, and click **Install**.
 5. The Confirm Selected Features page opens. Continue with the installation procedure in the "Complete the installation procedure" step.
 6. Complete each of the previous steps to install **IBM WebSphere eXtreme Scale V8.5 Liberty Profile Developer Tools**.
- Complete the installation procedure.
 1. Expand the node for the tooling that you installed.
 2. Select **IBM WebSphere Application Server V8.5 Liberty Profile Developer Tools** or **IBM WebSphere eXtreme Scale V8.5 Liberty Profile Developer Tools**.
 3. Select any of the optional features that you want to install. When you are finished, click **Next**.

Remember: If you want to install any of the additional optional installation features, such as the WebSphere Application Server tools features for Version 8.5, 8.0, or 7.0, a separate set of installation instructions are available in the IBM WebSphere Application Server Developer Tools for Eclipse overview Version 8.5 topic in the WebSphere Application Server Information Center.

4. On the Review Licenses page, review the license text.
5. If you agree to the terms, click **I accept the terms of the license agreement** and then click **Finish**. The installation process starts.
6. When the installation process completes, restart the workbench.

Setting up your development environment within Eclipse

After you install the Liberty profile Eclipse tooling for WebSphere eXtreme Scale, you must configure your eXtreme Scale servers in the Liberty profile and generate an Eclipse project in which you can begin development tasks.

Configuring eXtreme Scale in the Liberty profile using Eclipse tools

You must configure your WebSphere eXtreme Scale servers to run in the WebSphere Application Server Liberty profile. Complete this task to configure eXtreme Scale servers with Eclipse tools.

Before you begin

You must define a Liberty profile server in Eclipse. To complete this task, see [Creating a Liberty profile server using developer tools](#).

About this task

Configuring the eXtreme Scale server entails specifying the server properties and including those properties in the Liberty profile server.xml file in the `wlp_home/usr/servers/your_server_name` directory. This server definition is required to run eXtreme Scale in the Liberty profile.

This procedure also includes adding the configuration from the eXtreme Scale server properties file, `xsServerConfig.xml`, to the `server.xml` file.

Procedure

1. Generate the eXtreme Scale server properties file.
 - a. Click **File** > **New** > **Other**.
 - b. Expand **WebSphere eXtreme Scale**, and select **Container server configuration file**. Click **Next**. The eXtreme Scale Server Configuration File window is displayed.
 - c. Click **Browse** to specify where the Liberty profile is installed. Then, select the Liberty profile server definition for which you want to configure for your eXtreme Scale servers. Click **Next**. The General Server Configuration window is displayed.
 - d. Complete the server configuration. Click **Next**. The Container Server Configuration window is displayed.
 - e. Complete the container server configuration. Click **Next**.
 - f. If you included the catalog server configuration, then another window is displayed, where you specify the catalog server settings. Click **Next**. The Server Logging Configuration window is displayed.
 - g. Complete the logging information pages, and click **Next** until the Security Configuration window is displayed.
 - h. Optional: Specify the location of the `objectGridSecurity.xml` file, which describes the security properties that are common to all servers, including catalog servers and container servers. An example of the defined security properties is the authenticator configuration, which represents the user registry and authentication mechanism. The file name specified for this property must be in a URL format, such as `file:///tmp/og/objectGridSecurity.xml`.
 - i. Click **Finish**.

A configuration file is generated in the Liberty profile.

2. Include the configuration from the eXtreme Scale server properties file in the `server.xml` file.
 - a. Open the Servers view in Eclipse.
 - b. Expand Liberty Server to locate your server configuration XML file.
 - c. Double-click the entry for your server configuration to open the file.
 - d. Click **Add**, and select **Include** to add an include statement to the `server.xml` file. Click **OK**.
 - e. Under Include Details, click **Browse**. The Browse for Include File window is displayed.
 - f. Select `xsServerConfig.xml`, to include the server configuration settings that you created in step 1. Click **OK**.

What to do next

The eXtreme Scale server configuration file, `xsServerConfig.xml`, is now included in the Liberty profile `server.xml` file. Now, you are ready to start the Liberty profile server, where your eXtreme Scale servers will run.

Creating an OSGi bundle project for eXtreme Scale grid development

To use Eclipse as the development environment for your WebSphere eXtreme Scale servers in the Liberty profile, you must create an Eclipse project within the supported Open Services Gateway initiative (OSGi) framework.

Procedure

1. Create the OSGi bundle project in Eclipse.
 - a. Click **File > New > Project**. The "Select a wizard" window is displayed.
 - b. Expand the WebSphere eXtreme Scale folder, and select the **Object grid** project. The "Object grid project" window is displayed.
 - c. Click **Add**, and enter a backing map name to add the object grid map for which you want to complete development activities. You can enter multiple maps on this page. Click **Next**.
 - d. Specify object grid parameters for each map that you entered. Click **Next**.
 - e. Specify the deployment parameters, and click **Finish**.

The OSGi bundle project is created, and you can access eXtreme Scale APIs to complete development activities in the Liberty profile. The bundle includes the `gridBlueprint.xml` file. This file includes the location of the eXtreme Scale configuration files, `objectGrid.xml` and `gridDeployment.xml`. These configuration files include the map or maps that you created in the step c.

2. Export the bundle project, and place the bundle in the grids folder. You must export the project to deploy eXtreme Scale applications in the Liberty profile. When you export the project, it is exported as a bundle Java archive (JAR) file to the `Liberty_profile_Server_Definition/grids` folder, which you must manually create before you export the bundle.
 - a. Right-click the project that you just created, and select **Export > OSGi Bundle or Fragment**. The OSGi Application Export window is displayed.
 - b. Specify where you want to export the bundle JAR file. Click **Finish**.

Migrating a WebSphere Application Server memory-to-memory replication or database session to use WebSphere eXtreme Scale session management

Java

You can migrate any previously set memory-to-memory replication session or database session to use WebSphere eXtreme Scale session management.

Before you begin

- For session support for client applications running on WebSphere Application Server in the cluster, WebSphere eXtreme Scale must be installed on top of the WebSphere Application Server node deployments, including the deployment manager node. See *Installing WebSphere eXtreme Scale or WebSphere eXtreme Scale Client with WebSphere Application Server*.
- A WebSphere eXtreme Scale grid environment, that consists of one or more catalog and container servers must be started. For more information, see *Starting and stopping stand-alone servers*.
- If the catalog servers within your catalog service domain have Secure Sockets Layer (SSL) enabled or you want to use SSL for a catalog service domain with SSL supported, then global security must be enabled in the WebSphere

Application Server administrative console. You require SSL for a catalog server by setting the `transportType` attribute to `SSL-Required` in the `Server` properties file. For more information, see *Global security settings*.

About this task

The steps in this scenario are for Version 8.5 of the WebSphere Application Server administrative console. This information may vary slightly depending on the version of WebSphere Application Server you are using.

Note: WebSphere eXtreme Scale Version 8.6 is not supported on versions of WebSphere Application Server prior to Version 7.0.

Taking note of previous configuration settings in WebSphere Application Server administrative console

Java

As part of migration to a WebSphere eXtreme Scale session, you should take note of your previous configuration settings in WebSphere Application Server administrative console. When migrating to a WebSphere eXtreme Scale session, the configuration settings have to reflect what you already had configured for your database or memory-to-memory session.

About this task

There are specific settings in WebSphere Application Server administrative console that you should take note of. You will need these values when updating the `splicer.properties` file. The steps in this procedure are for Version 8.5 of the WebSphere Application Server administrative console. This information may vary slightly depending on the version of WebSphere Application Server you are using.

Note: WebSphere eXtreme Scale Version 8.6 is not supported on versions of WebSphere Application Server prior to Version 7.0.

Procedure

1. Start the WebSphere Application Server administrative console.
 - If you have previously configured settings at the server level, then go to:
 - a. **Servers > Server Types > WebSphere application servers**
 - b. In the **Application servers** area, select **your server name**
 - c. In the **Container Settings** area, click **Session management**
 - If you have previously configured settings at the application level, then go to:
 - a. **Applications > All applications.**
 - b. In the **Application servers** area, select **your application name.**
 - c. In the **Web Module Properties** area, click **Session management**
2. In the **General Properties**, select the **Allow Overflow** check box.
3. In the **General Properties** area, take note of the WebSphere Application Server settings. You will need these values later to update the properties in the `splicer.properties` file.

Table 11. Configuration settings to update the `splicer.properties` file

Settings in the WebSphere Application Server administration console	Properties to update in the <code>splicer.properties</code> file
Enable cookies	<code>useCookies</code>
Enable URL rewriting	<code>useURLEncoding</code>
Maximum in-memory session count	<code>sessionTableSize</code>

- In the **General Properties** area, if the **Enable cookies** check box is selected, then click it and take note of the WebSphere Application Server settings. You will need these values later to update the properties in the `splicer.properties` file.

Table 12. Configuration settings for the properties in the `splicer.properties` file

Settings in the WebSphere Application Server administration console	Properties to update in the <code>splicer.properties</code> file
Cookie domain	<code>cookieDomain</code>
Cookie path	<code>cookiePath</code>

- Click **Session management** and in the **Additional Properties** area, click **Distributed environment settings**.
- In the **Distributed Sessions** area, change your previous database or memory-to-memory replication configuration to **None**.
- Click **Custom Tuning Properties** and take note of the WebSphere Application Server settings. You will need these values later to update the properties in the `splicer.properties` file

Table 13. Configuration settings for the properties in the `splicer.properties` file

Settings in the WebSphere Application Server administration console	Properties to update in the <code>splicer.properties</code> file
Write frequency	<code>replicationInterval</code>
Write contents	<code>fragmentedSession</code>

What to do next

Next, create the catalog service domain for a WebSphere eXtreme Scale session.

Creating the catalog service domain for WebSphere eXtreme Scale session management

Java

As part of migration to a WebSphere eXtreme Scale session, you must create a catalog service domain in the WebSphere Application Server administrative console.

About this task

The steps in this procedure are for Version 8.5 of the WebSphere Application Server administrative console. This information may vary slightly depending on the version of WebSphere Application Server you are using.

Note: WebSphere eXtreme Scale Version 8.6 is not supported on versions of WebSphere Application Server prior to Version 7.0. Create the catalog service domain for the data grid in the WebSphere Application Server administrative console. For more information, see *Creating catalog service domains in WebSphere Application Server*.

Procedure

1. Start the WebSphere Application Server administrative console.
2. In the top menu, click **System administration > WebSphere eXtreme Scale > Catalog service domains**

Note: If you do not see WebSphere eXtreme Scale, then your WebSphere Application Server profile has not been augmented for the data grid. For more information, see *Creating and augmenting profiles for WebSphere eXtreme Scale*.

3. Click **New**.
4. Specify a name for the catalog service in the **Name** box.
5. In the **Catalog Servers** area, choose **Remote Server** and specify the location or the name of the remote server in the box.
6. Specify a port number in the **Listener Port** box.
7. Click **Apply** or **OK** and save the configuration.

What to do next

Next, use the previous configuration settings that you noted in the WebSphere Application Server administration console to associate either an application or an application server to WebSphere eXtreme Scale session management.

Configuring WebSphere eXtreme Scale to use your previous configuration settings

Java

Using your previous configuration settings that you noted in the WebSphere Application Server administration console, you must use these settings to associate either an application or an application server to WebSphere eXtreme Scale session management.

About this task

The steps in this procedure are for Version 8.5 of the WebSphere Application Server administrative console. This information may vary slightly depending on the version of WebSphere Application Server you are using.

Note: WebSphere eXtreme Scale Version 8.6 is not supported on versions of WebSphere Application Server prior to Version 7.0.

Procedure

- If you want to configure an application so that it is associated with WebSphere eXtreme Scale session management, follow these steps:
 1. Start the WebSphere Application Server administrative console.
 2. In the top menu, click **Applications > All applications**.
 3. In the **WebSphere Enterprise Applications** area, select **application name**.

4. In the **Web Module** properties area, click **Session management**
5. Click **eXtreme Scale session management settings**.
6. If you do not see WebSphere eXtreme Scale, then your WebSphere Application Server profile has not been augmented for WebSphere eXtreme Scale. For more information, see *Creating and augmenting profiles for WebSphere eXtreme Scale*.
7. To configure an application for WebSphere eXtreme Scale in a stand-alone environment, follow these steps:
 - a. In the **Manage session persistence by** list, select **Remote eXtreme Scale data grid**
 - b. Select the catalog service domain you had created from the list.
 - c. Click **Browse** to select the grid.
8. Click **Apply** or **OK** and save the configuration.
9. A new `splicer.properties` file is created for this application. The location of the `splicer.properties` file is the value of the a new property `{application name},com.ibm.websphere.xs.sessionFilterProps`. To locate the custom property, go to **System administration > Cell** and click **Custom properties**.
10. Update the `splicer.properties` file with the values you obtained in “Taking note of previous configuration settings in WebSphere Application Server administrative console” on page 263.
11. Restart the application server processes.

Note: Change the `splicer.properties` at the Deployment Manager level so that the properties get synchronized to the node agent. If you update the `splicer.properties` at the node level, then the Deployment Manager will overwrite the `splicer.properties` file at the next synchronization.

Note: If you go back to database session management and then return to WebSphere eXtreme Scale session management, the `splicer.properties` file is recreated so any changes you made will be overridden. For a discussion on the file synchronization process from the Deployment Manager to the Notes and what gets changed, see *System Management File Synchronization*.

- If you want to configure an application server so that it is associated with WebSphere eXtreme Scale session management, follow these steps:
 1. Start the WebSphere Application Server administrative console.
 2. In the top menu, click **Servers > Server Types > WebSphere application servers**.
 3. In the **Application servers** area, select **your server name**.
 4. In the **Container Settings** area, click **Session management**
 5. Click **eXtreme Scale session management settings**

Note: If you do not see WebSphere eXtreme Scale, then your WebSphere Application Server profile has not been augmented for WebSphere eXtreme Scale. For more information, see *Creating and augmenting profiles for WebSphere eXtreme Scale*.

6. To configure an application server for WebSphere eXtreme Scale in a stand-alone environment, follow these steps:
 - a. In the **Manage session persistence by** list, select **Remote eXtreme Scale data grid**
 - b. Select the catalog service domain you had created from the list.
 - c. Click **Browse** to select the grid.

7. Click **Apply** or **OK** and save the configuration.
8. A new `splicer.properties` file is created for this application. The location of the `splicer.properties` file is the value of the a new property `com.ibm.websphere.xs.sessionFilterProps`. To locate the custom property, go to **Servers > Server Types > WebSphere application servers**.
9. In the **Application servers** area, select **your server name**.
10. In the **Server Infrastructure** area, select **Custom properties**.
11. Update the `splicer.properties` file with the values you obtained in “Taking note of previous configuration settings in WebSphere Application Server administrative console” on page 263.
12. Restart the application server processes.

Note: Change the `splicer.properties` at the Deployment Manager level so that the properties get synchronized to the node agent. If you update the `splicer.properties` at the node level, then the Deployment Manager will overwrite the `splicer.properties` file at the next synchronization.

Note: If you go back to database session management and then return to WebSphere eXtreme Scale session management, the `splicer.properties` file is recreated so any changes you made will be overridden. For a discussion on the file synchronization process from the Deployment Manager to the Notes and what gets changed, see System Management File Synchronization.

Results

You have now changed your previous configuration settings for either a memory-to-memory or database session management with WebSphere eXtreme Scale session management.

Scenario: Using WebSphere eXtreme Scale as a dynamic cache provider

The WebSphere Application Server provides a Dynamic Cache service available to deployed Java EE applications. This service is used to cache business data, generated HTML, command output, etc. Initially, the only provider for the Dynamic Cache service was the default dynamic cache provider that is built into the WebSphere Application Server. Today customers can also specify WebSphere eXtreme Scale to be the cache provider for any given cache instance. This enables applications that use the Dynamic Cache service, to use the features and performance capabilities of WebSphere eXtreme Scale.

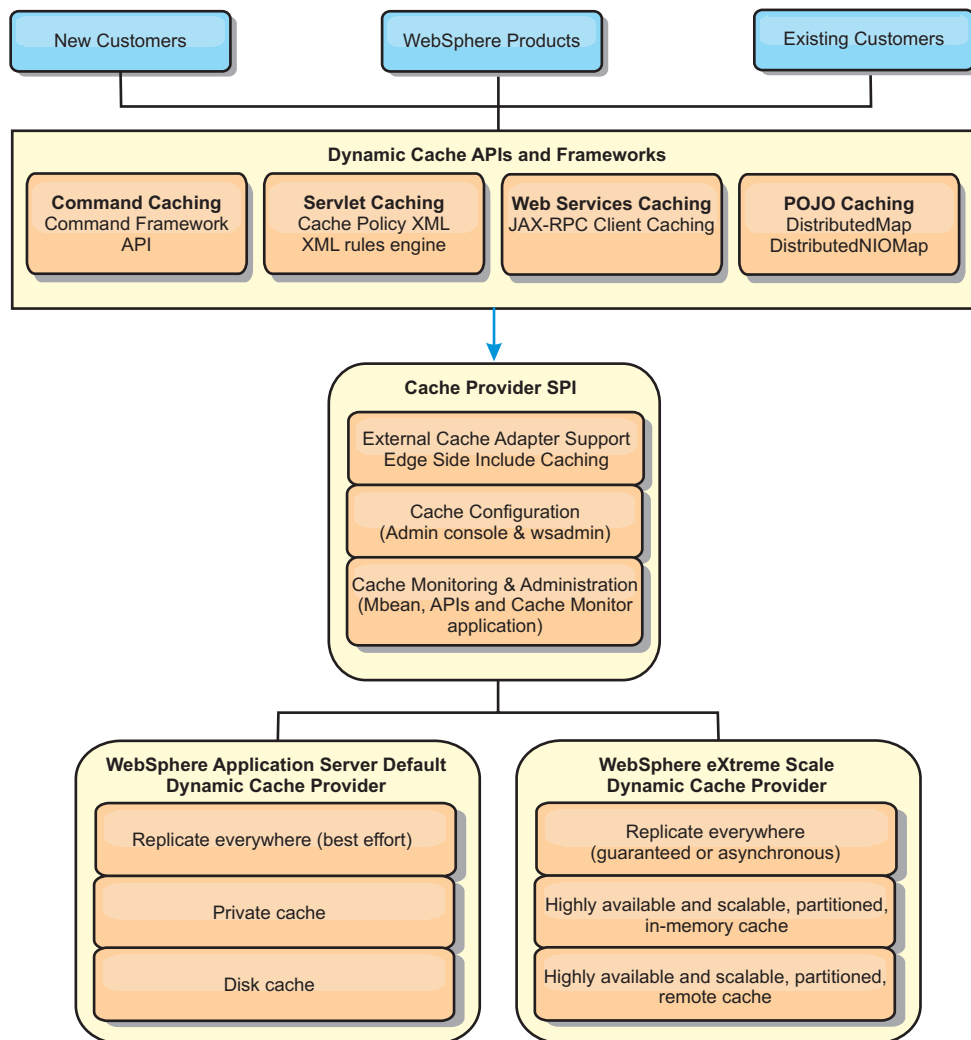
About this task

Dynamic cache provider overview

The WebSphere Application Server provides a dynamic cache service that is available to deployed Java EE applications. This service is used to cache data such as output from servlet, JSP, or commands, and object data programmatically specified within an enterprise application with the DistributedMap APIs. .

Initially, the only service provider for the dynamic cache service was the default dynamic cache engine that is built into WebSphere Application Server. You can also specify WebSphere eXtreme Scale to be the cache provider for any cache instance.

By setting up this capability, you can enable applications that use the dynamic cache service, to use the features and performance capabilities of WebSphere eXtreme Scale.



You can install and configure the dynamic cache provider as described in *Configuring the default dynamic cache instance (baseCache)*.

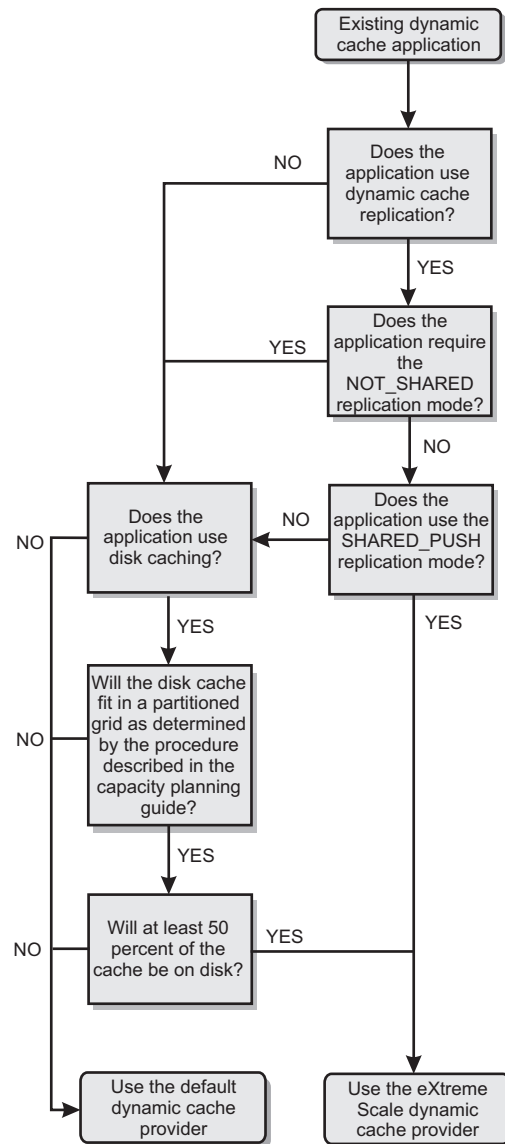
Deciding how to use WebSphere eXtreme Scale

The available features in WebSphere eXtreme Scale significantly increase the distributed capabilities of the dynamic cache service beyond what is offered by the default dynamic cache provider and data replication service. With eXtreme Scale, you can create caches that are truly distributed between multiple servers, rather than just replicated and synchronized between the servers. Also, eXtreme Scale caches are transactional and highly available, ensuring that each server sees the same contents for the dynamic cache service. WebSphere eXtreme Scale offers a higher quality of service for cache replication provided via DRS.

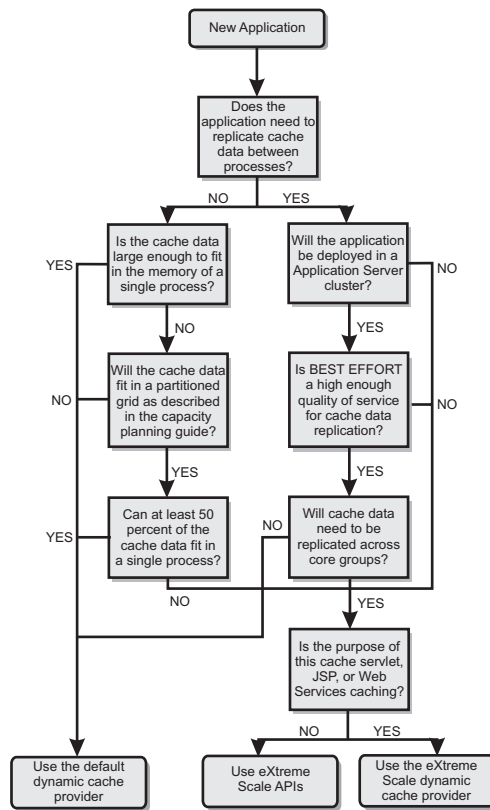
However, these advantages do not mean that the eXtreme Scale dynamic cache provider is the right choice for every application. Use the decision trees and

feature comparison matrix below to determine what technology fits your application best.

Decision tree for migrating existing dynamic cache applications



Decision tree for choosing a cache provider for new applications



Feature comparison

Table 14. Feature comparison

Cache features	Default provider	eXtreme Scale provider	eXtreme Scale API
Local, in-memory caching	Yes	via Near-cache capability	via Near-cache capability
Distributed caching	via DRS	Yes	Yes
Linearly scalable	No	Yes	Yes
Reliable replication (synchronous)	No	Yes	Yes
Disk overflow	Yes	N/A	N/A
Eviction	LRU/TTL/heap-based	LRU/TTL (per partition)	LRU/TTL (per partition)
Invalidation	Yes	Yes	Yes
Relationships	Dependency / template ID relationships	Yes	No (other relationships are possible)
Non-key lookups	No	No	via Query and index
Back-end integration	No	No	via Loaders
Transactional	No	Yes	Yes
Key-based storage	Yes	Yes	Yes

Table 14. Feature comparison (continued)

Cache features	Default provider	eXtreme Scale provider	eXtreme Scale API
Events and listeners	Yes	No	Yes
WebSphere Application Server integration	Single cell only	Multiple cell	Cell independent
Java Standard Edition support	No	Yes	Yes
Monitoring and statistics	Yes	Yes	Yes
Security	Yes	Yes	Yes

For a more detailed description on how eXtreme Scale distributed caches work, see “Planning the topology” on page 147.

Note: An eXtreme Scale distributed cache can only store entries where the key and the value both implement the `java.io.Serializable` interface.

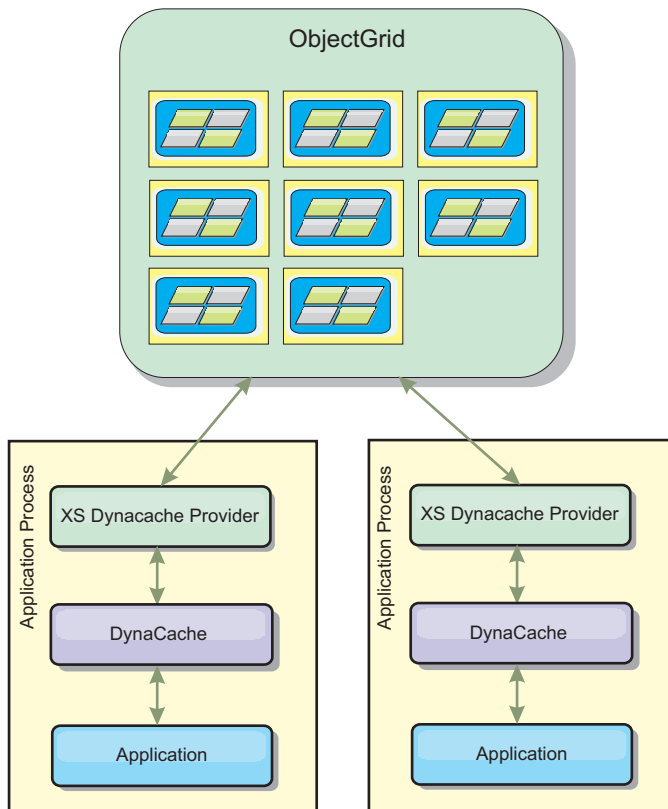
Topology

Deprecated:  **8.6+** The local, embedded, and embedded-partitioned topology types are deprecated.

A dynamic cache service that is created with eXtreme Scale as the provider can be deployed in a remote topology.

Remote topology

The remote topology eliminates the need for a disk cache. All of the cache data is stored outside of WebSphere Application Server processes. WebSphere eXtreme Scale supports standalone container processes for cache data. These container processes have a lower overhead than a WebSphere Application Server process and are also not limited to using a particular Java Virtual Machine (JVM). For example, the data for a dynamic cache service being accessed by a 32-bit WebSphere Application Server process could be located in an eXtreme Scale container process running on a 64-bit JVM. This allows users to use the increased memory capacity of 64-bit processes for caching, without incurring the additional overhead of 64-bit for application server processes. The remote topology is shown in the following image:



Dynamic cache engine and eXtreme Scale functional differences

Users should not notice a functional difference between the two caches except that the WebSphere eXtreme Scale backed caches do not support disk offload or statistics and operations related to the size of the cache in memory.

No appreciable difference exists in the results returned by most dynamic cache API calls, regardless of whether you are using the default dynamic cache provider or the eXtreme Scale cache provider. For some operations, you cannot emulate the behavior of the dynamic cache engine with eXtreme Scale.

Dynamic cache statistics

You can retrieve statistical data for a WebSphere eXtreme Scale dynamic cache with eXtreme Scale monitoring tooling. For more information, see [Monitoring](#).

MBean calls

The WebSphere eXtreme Scale dynamic cache provider does not support disk caching. Any MBean calls relating to disk caching do not work.

Dynamic cache replication policy mapping

The eXtreme Scale dynamic cache provider's remote topology supports a replication policy that most closely matches the SHARED_PULL and SHARED_PUSH_PULL policy (using the terminology used by the default WebSphere Application Server dynamic cache provider). In an eXtreme Scale dynamic cache, the distributed state of the cache is consistent between all the servers.

8.6+ Global index invalidation

You can use a global index to improve invalidation efficiency in large partitioned environments; for example, more than 40 partitions. Without the global index feature, the dynamic cache template and dependency invalidation processing must send remote agent requests to all partitions, which results in slower performance. When you configure a global index, invalidation agents are sent only to applicable partitions that contain cache entries that are related to the Template or Dependency ID. The potential performance improvement is greater in environments with large numbers of partitions configured. You can configure a global index with the Dependency ID and Template ID indexes, which are available in the example dynamic cache objectGrid descriptor XML files. For more information, see “Configuring an Enterprise Data Grid in a stand-alone environment for dynamic caching” on page 274.

Near cache

You can configure a dynamic cache instance to create and maintain a near cache, which resides locally within the application server JVM. The near cache contains a subset of the entries that are contained within the remote dynamic cache instance. You can configure a near cache instance with a `dynacache-nearCache-ObjectGrid.xml` file. For more information, see “Configuring an Enterprise Data Grid in a stand-alone environment for dynamic caching” on page 274. There are also custom properties for tuning the near-cache. For more information, see Dynamic cache custom properties.

Multi-master replication

A dynamic cache instance can be configured to support a multi-master replication topology. Collision arbitration is important for any replication topology. For more information, see “Design considerations for multi-master replication” on page 176. The sample `objectgrid.xml` files that are delivered for the dynamic cache grid configuration are configured with a default collision arbiter: `<bean`

```
id="CollisionArbiter"  
className="com.ibm.ws.objectgrid.dynacache.arbiters.DynacacheCollisionArbiter"/>
```

The arbiter is invoked to resolve collisions during replication. It first resolves collisions that result from remove and invalidation events, applying these actions over any other event. For all other events, the changes from the lexically lowest named catalog service domain will be applied. For more information, see “Planning multiple data center topologies” on page 169.

Note: Dynamic cache grid users of WebSphere Portal Server or WebSphere Commerce Server may have defined multiple cache instances within their WebSphere Application Server configuration. If you decide to enable multi-master replication for the eXtreme Scale servers, then this configuration will only affect those cache instances that are defined to use the eXtreme Scale servers as the dynamic cache provider, and will not affect the cache instances defined to use the default WebSphere Application Server dynamic cache provider.

Additional information

- Dynamic cache Redbook
- Dynamic cache documentation
 - WebSphere Application Server 7.0
- DRS documentation

Planning environment capacity

If you have an initial data set size and a projected data set size, you can plan the capacity that you need to run WebSphere eXtreme Scale. By using these planning exercises, you can deploy WebSphere eXtreme Scale efficiently for future changes and maximize the elasticity of the data grid, which you would not have with a different scenario such as an in-memory database or other type of database.

Configuring an Enterprise Data Grid in a stand-alone environment for dynamic caching

Copy and modify these deployment and objectGrid descriptor files in order to configure an enterprise grid for dynamic caching. These files are used to start an enterprise data grid.

About this task

When WebSphere eXtreme Scale is specified as the provider for a WebSphere Application Server dynamic cache instance, the WebSphere eXtreme Scale servers are started in either a stand-alone environment or within a WebSphere Application Server environment, see [Starting and stopping stand-alone servers](#) for more information. This process requires the use of deployment and objectGrid descriptor files that are used to configure the enterprise data grid. Dynamic caching requires a specific configuration. Therefore, several XML files are delivered with WebSphere eXtreme Scale that are intended to be copied, altered (as needed), and used to start the enterprise data grid. These files can be used as-is, but are subject to change and therefore should be copied to a separate location before they are altered or used.

Note: Depending on how you have installed WebSphere eXtreme Scale, these files are located in either the `was_root/optionalLibraries/ObjectGrid/dynacache/etc` directory for installations with WebSphere Application Server; or for an installation in a stand-alone environment, these files are located in `wxs_install_root/ObjectGrid/dynacache/etc` directory.

Important: It is highly recommended that these files be copied to some other location before they are edited or used.

Dynamic cache descriptor file (dynacache-deployment.xml)

This file is the deployment descriptor file for starting a container server for dynamic caching, see [Deployment policy descriptor XML file](#) for more information. Although this file can be used as-is, the following following elements or attributes are occasionally changed or have significant importance:

- **mapSet name and map ref**

The **name** attribute in `mapSet`, and the defined value for `map ref` do not directly correspond to the dynamic cache instance name configured for WebSphere Application Server and are typically not changed. If, however, these values are changed, then corresponding custom properties must be added to the configuration of the dynamic cache instance. For more information, see [Customizing a dynamic cache instance with custom properties](#).

- **numberOfPartitions**

This attribute may be changed to represent the appropriate number of partitions for your configuration. For more information, see “Planning environment capacity” on page 274.

- **maxAsyncReplicas**

This attribute may be changed. A dynamic cache typically is used in a side-cache model with a database or some other source as the system of record for the data. As a result, setting this to OPTIMISTIC or NONE will trigger near cache processing, when the eXtreme I/O (XIO) transport type is used, and the space and performance trade-offs required to make the data highly available discourage the use of replication. However, in some cases high availability is important.

- **numInitialContainers**

This attribute should be set to the number of containers that will be included in the initial startup of the enterprise data grid. Having this set correctly will aid in the placement and distribution of partitions throughout the data grid.

Dynamic cache ObjectGrid descriptor XML file (dynacache-objectgrid.xml)

This file is the recommended ObjectGrid descriptor file for starting a container server for dynamic caching, see ObjectGrid descriptor XML file for more information. It is configured to run with the eXtreme I/O transport type (XIO) using eXtreme Data Formatting (XDF). In addition, the Dependency ID and Template ID indexes are configured to use a Global Index, which improves invalidation performance. Although this file can be used as-is, the following following elements or attributes are occasionally changed or have significant importance.

- **objectGrid name and backingMap name**

The **name** attributes in the objectGrid and backingMap elements do not directly correspond to the dynamic cache instance name configured for WebSphere Application Server cache instance and typically do not need to be changed. If, however, these attributes are changed, then the corresponding custom properties must be added to the configuration of the dynamic cache instance. For more information, see Customizing a dynamic cache instance with custom properties.

- **copyMode**

Set this attribute to COPY_TO_BYTES. This value enables eXtreme Data Format (XDF) when the eXtreme I/O (XIO) transport type is used. Changing to some other copyMode will disable XDF and will require that you uncomment the ObjectTransformer plugin bean.

- **lockStrategy**

Set this attribute to PESSIMISTIC. Setting this to OPTIMISTIC or NONE will trigger near cache processing and must be accompanied with properties from the dynamic-nearcache-objectgrid.xml.

- **backingMapPluginCollections**

This element is required. The child elements Evictor plug-in and MapIndex plug-in are both required for dynamic caching and must not be removed.

- **GlobalIndexEnabled**

Both the DEPENDENCY_ID_INDEX and TEMPLATE_INDEX contain a GlobalIndexEnabled property set to true. Setting this value to false will disable the global index feature for these indexes. It is recommended to leave these global indexes enabled unless you are running with a small number of total partitions, for example, less than 40.

- **objectTransformer**

Since this objectGrid descriptor file is intended to run in eXtreme Data Format (XDF), it has been commented out. If you want to disable XDF (by changing the copyMode value) then you must uncomment this plug-in.

Dynamic near cache ObjectGrid descriptor file (dynacache-nearcache-objectgrid.xml)

This file is the recommended ObjectGrid descriptor file for starting grid container servers for dynamic caching when a near-cache is desired. It is configured to run with the eXtreme I/O transport type (XIO) using eXtreme Data Formatting (XDF). In addition, the Dependency ID and Template indexes are configured to use a Global Index, which improves invalidation performance. The dynamic caching near cache capability requires the use of the eXtreme I/O (XIO) transport type.

Although this file can be used as-is, the following elements or attributes are occasionally changed or have significant importance:

- **objectGrid name and backingMap name**

These values in this file do not directly correspond to the dynamic cache instance name configured for the WebSphere Application Server's cache instance and typically do not need to be changed. If, however, these values are changed, then corresponding custom properties must be added to the configuration of the dynamic cache instance.

- **lockStrategy**

This property must be set to OPTIMISTIC or NONE to enable a near cache. No other lockingStrategy supports a near cache.

- **nearCacheInvalidationEnabled**

This property must be set to true to enable a dynamic caching near cache. This feature uses pub-sub to flow invalidations from the far cache to the near cache instances, keeping them in-sync.

- **nearCacheLastAccessTTLSyncEnabled**

This property must be set to true to enable a dynamic caching near cache. This feature uses pub-sub to flow TTL evictions from the far cache to the near cache instances, keeping them in -sync.

- **copyMode**

This backingMap property is set to COPY_TO_BYTES. This value enables eXtreme Data Format (XDF) when the eXtreme I/O (XIO) transport type is used. Changing to some other copyMode will disable XDF and will require that the ObjectTransformer plugin bean be uncommented.

- **CollisionArbitor**

This public interface decides what value to store when there is a collision on multiple primaries.

- **backingMapPluginCollections**

The MapIndexPlugins and Evictor are mandatory items for dynamic caching and must not be removed.

- **GlobalIndexEnabled**

Both the DEPENDENCY_ID_INDEX and TEMPLATE_INDEX contain a GlobalIndexEnabled property set to true. Setting this value to false will disable the global indexfeature for these indexes. It is recommended to leave these global indexes enabled unless you are running with a small number of total partitions (< 40).

- **ObjectTransformer**

Since this file is intended to run in eXtreme Data Format (XDF) this plugin is commented out. If XDF is to be disabled (via changing the copyMode) then this plugin must be uncommented.

Dynamic legacy ObjectGrid descriptor file (dynacache-legacy85-objectgrid.xml)

This file is the recommended ObjectGrid descriptor file for starting a container server for dynamic caching when you have chosen a near-cache. Although this file can be used as-is, the following elements or attributes are occasionally changed or have significant importance:

- **objectGrid name and backingMap name**

These values in this file do not directly correspond to the dynamic cache instance name configured for the WebSphere Application Server's cache instance and typically do not need to be changed. If, however, these values are changed, then corresponding custom properties must be added to the configuration of the dynamic cache instance.

- **copyMode**

This backingMap property is set to COPY_ON_READ_AND_COMMIT. This value should not be changed.

- **lockStrategy**

This backingMap property is set to PESSIMISTIC. This value should not be changed.

- **backingMapPluginCollections**

The MapIndexPlugins, Evictor, and Object Transformer are mandatory items for dynamic caching and must not be removed.

Configuring an Enterprise Data Grid for dynamic caching using a Liberty profile

A Liberty profile server can host a data grid that caches data for applications that have dynamic cache enabled.

Before you begin

- Install the Liberty profile. For more information, see [Installing the Liberty profile](#).
- Create an application that uses dynamic cache. For more information, see [Configuring the default dynamic cache instance \(baseCache\)](#).

About this task

The Liberty profile hosts the data grid which supports dynamic-cache-enabled applications. This means that the application runs on a traditional installation of WebSphere Application Server. For those applications to be cached by the eXtreme Scale runtime environment, you must configure WebSphere Application Server to use the catalog domain service and server properties that you specify in the Liberty profile.

Procedure

1. Enable the WebSphere eXtreme Scale dynamic cache feature.
 - a. Add the dynamic cache feature to the Liberty profile server.xml file. For example, your server.xml file resembles the following stanza of code:


```

<featureManager>
<feature>eXtremeScale.server-1.1</feature>
<feature>eXtremeScale.dynacacheGrid-1.1</feature>
</featureManager>

```

- Optional: Set properties on the `xsDynacacheGrid` element in the `server.xml` file. You can change any of the following properties; however, it is recommended that you accept the default values.

globalIndexDisabled

Global index invalidation improves invalidation efficiency in a large, partitioned environment; for example, more than 40 partitions. For more information, see “Data invalidation” on page 65. Default value: `false`

objectGridName

A string that specifies the name of the data grid. Default value: `DYNACACHE_REMOTE`. For more information, see ObjectGrid descriptor XML file and Deployment policy descriptor XML file.

objectGridTxTimeout

Specifies the amount of time in seconds that a transaction is allowed for completion. If a transaction does not complete in this amount of time, the transaction is marked for rollback and a `TransactionTimeoutException` exception results. Default value: 30 (in seconds)

backingMapLockStrategy

Specifies if the internal lock manager is used whenever a map entry is accessed by a transaction. Set this attribute to one of three values: `OPTIMISTIC`, `PESSIMISTIC`, or `NONE`. Default value: `PESSIMISTIC`

backingMapCopyMode

Specifies if a get operation of an entry in the `BackingMap` instance returns the actual value, a copy of the value, or a proxy for the value. If you use eXtreme data format (XDF) so that both Java and .NET can access the same data grid, then the default and required copy mode is `COPY_TO_BYTES`. Otherwise, the copy mode, `COPY_ON_READ_AND_COMMIT` is used. Set the `CopyMode` attribute to one of five values:

COPY_ON_READ_AND_COMMIT

The default value is `COPY_ON_READ_AND_COMMIT`. Set the value to `COPY_ON_READ_AND_COMMIT` to ensure that an application never has a reference to the value object that is in the `BackingMap` instance. Instead, the application is always working with a copy of the value that is in the `BackingMap` instance. (Optional)

COPY_ON_READ

Set the value to `COPY_ON_READ` to improve performance over the `COPY_ON_READ_AND_COMMIT` value by eliminating the copy that occurs when a transaction is committed. To preserve the integrity of the `BackingMap` data, the application commits to delete every reference to an entry after the transaction is committed. Setting this value results in an `ObjectMap.get` method returning a copy of the value instead of a reference to the value, which ensures changes that are made by the application to the value does not affect the `BackingMap` element until the transaction is committed.

COPY_ON_WRITE

Set the value to `COPY_ON_WRITE` to improve performance over the `COPY_ON_READ_AND_COMMIT` value by eliminating the copy that occurs when `ObjectMap.get` method is called for the first time by a transaction for a given key. Instead, the `ObjectMap.get` method

returns a proxy to the value instead of a direct reference to the value object. The proxy ensures that a copy of the value is not made unless the application calls a set method on the value interface.

NO_COPY

Set the value to `NO_COPY` to allow an application to never modify a value object that is obtained using an `ObjectMap.get` method in exchange for performance improvements. Set the value to `NO_COPY` for maps associated with EntityManager API entities.

COPY_TO_BYTES

Set the value to `COPY_TO_BYTES` to improve memory footprint for complex Object types and to improve performance when the copying of an Object relies on serialization to make the copy. If an Object is not `Cloneable` or a custom `ObjectTransformer` with an efficient `copyValue` method is not provided, the default copy mechanism is to serialize and inflate the object to make a copy. With the `COPY_TO_BYTES` setting, inflate is only performed during a read and serialize is only performed during commit.

Default value: `COPY_ON_READ_AND_COMMIT`

backingMapNearCacheEnabled

Set the value to `true` to enable the client local cache. To use a near cache, the `lockStrategy` attribute must be set to `NONE` or `OPTIMISTIC`. Default value: `false`

mapSetNumberOfPartitions

Specifies the number of partitions for the `mapSet` element. Default value: 47

mapSetMinSyncReplicas

Specifies the minimum number of synchronous replicas for each partition in the `mapSet`. Shards are not placed until the domain can support the minimum number of synchronous replicas. To support the `minSyncReplicas` value, you need one more container server than the `minSyncReplicas` value. If the number of synchronous replicas falls below the `minSyncReplicas` value, write transactions are no longer allowed for that partition. Default value: 0

mapSetMaxSyncReplicas

Specifies the maximum number of synchronous replicas for each partition in the `mapSet`. No other synchronous replicas are placed for a partition after a domain reaches this number of synchronous replicas for that specific partition. Adding container servers that can support this `ObjectGrid` can result in an increased number of synchronous replicas if your `maxSyncReplicas` value has not already been met. Default value: 0

mapSetNumInitialContainers

Specifies the number of container servers that are required before initial placement occurs for the shards in this `mapSet` element. This attribute can help save process and network bandwidth when bringing a data grid online from a cold startup. Default value: 1

mapSetDevelopmentMode

With this attribute, you can influence where a shard is placed in relation to its peer shards. When the `developmentMode` attribute is set to `false`, no two shards from the same partition are placed on the same computer. When the `developmentMode` attribute is set to `true`, shards from the same

partition can be placed on the same machine. In either case, no two shards from the same partition are ever placed in the same container server.
Default value: false

mapSetReplicaReadEnabled

If this attribute is set to true, read requests are distributed amongst a partition primary and its replicas. If the replicaReadEnabled attribute is false, read requests are routed to the primary only. Default value: false

3. Configure WebSphere Application Server to point to the Liberty profile.
You can connect WebSphere eXtreme Scale containers and dynamic-cache-enabled web applications to a catalog service domain that is running in another WebSphere Application Server cell or as stand-alone processes. Because remotely configured catalog servers do not automatically start in the cell, you must manually start any remotely configured catalog servers.
When you configure a remote catalog service domain, the domain name must match the domain name that you specified when you start the remote catalog servers. The default catalog service domain name for stand-alone catalog servers is DefaultDomain. Specify a catalog service domain name with the **startOgServer** or **startXsServer** command **-domain** parameter, a server properties file, or with the embedded server API. You must start each remote catalog server process in the remote domain with the same domain name. For more information about starting catalog servers, see Starting a stand-alone catalog service that uses the ORB transport.

Configuring dynamic cache instances

The dynamic cache service in WebSphere Application Server supports the creation of both a default cache instance (baseCache) and additional servlet and object cache instances.

About this task

The default cache instance (baseCache) was initially the only dynamic cache instance supported by the WebSphere Application Server and is currently the out-of-box dynamic cache instance used by WebSphere Commerce Suite. The additional servlet and object cache instances were added in later releases of WebSphere Application Server and are configured in a separate **Cache instance** section of the WebSphere Application Server administrative console.

Chapter 4. Samples



Several WebSphere eXtreme Scale tutorials, examples, and samples are available.

Examples

The following topics illustrate key WebSphere eXtreme Scale features.

- DataGrid API example
- Configuring local deployments

Community samples

The following samples from the WebSphere eXtreme Scale Samples Gallery illustrate how to use WebSphere eXtreme Scale in various environments to exhibit different features of the product.

Table 15. Available samples

Sample	Description
Asynchronous Service Framework	The Asynchronous Service framework provides a scalable and fault-tolerant processing fabric for asynchronous processing of messages. For more information, including how to download the sample, see the Samples Gallery: Asynchronous Service Framework sample .
Binary JSON (BSON) serializer	This sample demonstrates how to write an eXtreme Scale serializer and configure it to be used with eXtreme Scale. The serializer included with this sample uses Binary JSON (BSON) to describe and serialize objects. For more information, including how to download the sample, see the Samples Gallery: BSON serializer sample .
Client authentication security	This sample describes how to configure authentication requiring the client to provide valid credentials before the server gives any grid access. For more information, including how to download the sample, see the Samples Gallery: Client authentication security .
Creating dynamic maps	This sample demonstrates how to create maps after your grid has already been initialized. For eXtreme Scale 7.0 and higher, you can use templates to retrieve maps. For more information, including how to download the sample, see the Samples Gallery: Creating dynamic maps after grid initialization .

Table 15. Available samples (continued)

Sample	Description
Getting started with the product	The getting started samples are provided for a quick introduction to WebSphere eXtreme Scale and basic operation in a WebSphere Application Server environment. For more information, including how to download the sample, see the Samples Gallery: WebSphere eXtreme Scale getting started web application sample.
Getting started with Spring	The getting started with Spring sample is provided for a quick introduction to the Spring framework integration. The sample consists of shell and batch scripts that are designed to start a simple grid with little customization needed. For more information, including how to download the sample, see the Samples Gallery: Getting started with Spring sample.
Google protocol buffer serializer	This sample demonstrates how to write an eXtreme Scale serializer and configure it to run with eXtreme Scale. The serializer included with this sample uses the Google protocol buffers to describe and serialize objects. For more information, including how to download the sample, see the Samples Gallery: Google protocol buffer serializer sample.
Liberty profile	The Airport sample is provided as an introduction to eXtreme Scale installed in the WebSphere Application Server Version 8.5 Liberty profile environment. Liberty is a lightweight application server with a small Java virtual machine (JVM) footprint that starts in less than five seconds. The Liberty profile server runs simple create, read, update, and delete functions in the eXtreme Scale grid in milliseconds. The sample shows how large amounts of data (in this case, information about thousands of airports worldwide) can be stored using the WebSphere Application Server Liberty profile with WebSphere eXtreme Scale. For more information, including how to download the sample, see the Samples Gallery: Liberty profile airport sample.
Multimaster replication	The Multi-Master Replication Getting Started sample is provided for a quick introduction to multi-master (AP) replication. For more information, including how to download the sample, see the Samples Gallery: Multimaster Replication sample.

Table 15. Available samples (continued)

Sample	Description
OSGi framework	The OSGi sample is provided to help you install and run a WebSphere eXtreme Scale data grid in the Eclipse Equinox OSGi framework. The sample includes several plug-in bundles that illustrate the best practices for developing dynamic eXtreme Scale plug-in bundles so that the eXtreme Scale servers can be updated without a costly restart. For more information, including how to download the sample, see the Samples Gallery: WebSphere eXtreme Scale OSGi sample.
Queries with Entity Manager API	This sample demonstrates how to use queries in a distributed partitioned map with the EntityManager API. For more information, including how to download the sample, see the Samples Gallery: Running Queries in a partitioned grid using Entity Manager API .
Parallel queries with a ReduceGridAgent implementation	This sample demonstrates how to use the Data Grid API to run a query over every partition in the grid. For more information, including how to download the sample, see the Samples Gallery: Running Queries in Parallel using a ReduceGridAgent .
Resource adapter	Connect to an eXtreme Scale data grid in your application with the WebSphere eXtreme Scale resource adapter using the Java Transaction API (JTA). This sample helps you install the resource adapter, configure a connection factory, and code samples to connect and add or retrieve objects from your data grid. For more information, including how to download the sample, see the Samples Gallery: Using the WebSphere eXtreme Scale resource adapter .

Articles with tutorials and examples

Table 16. Available articles by feature

Article	Features
WebSphere eXtreme Scale Tutorial	Partitioning, replication, shards, zones, programming APIs, performance tuning
Building grid-ready applications	ObjectMap API, EntityManager API, queries, agents, Java SE and EE, statistics, partitioning, administration and operations, Eclipse
Scalable grid-style computing and data processing	EntityManager API, agents
Building a scalable, resilient, high-performance database alternative	ObjectMap API, replication, partitioning, administration and operations, Eclipse
Enhancing xsadmin for WebSphere eXtreme Scale	Administration

Table 16. Available articles by feature (continued)

Article	Features
Redbook: User's Guide	All topics

Free trial

To get started using WebSphere eXtreme Scale, download a free trial version. You can develop innovative, high-performance applications by extending the data caching concept using advanced features.

Trial download

You can download a free trial version of WebSphere eXtreme Scale, from [Download eXtreme Scale trial](#).

After downloading and unzipping the trial version of eXtreme Scale, navigate to the `gettingstarted` directory, and read the `GETTINGSTARTED_README.txt` file. This tutorial gets you started using eXtreme Scale, create a data grid on several servers, and run some simple applications to store and retrieve data in a grid. Before deploying eXtreme Scale in a production environment, there are several options to consider, including the number of servers to use, the amount of storage on each server, and synchronous or asynchronous replication.

Sample properties files

Server properties files contain settings for running your catalog servers and container servers. You can specify a server properties file for either a stand-alone or WebSphere Application Server configuration. Client property files contain settings for your client.

You can use the following sample properties files that are in the `wxs_install_root\properties` directory to create your properties file:

- `sampleServer.properties`
- `sampleClient.properties`

Sample: xsadmin utility

With the `xsadmin` utility, you can format and display textual information about your WebSphere eXtreme Scale topology. The sample utility provides a method for parsing and discovering current deployment data, and can be used as a foundation for writing custom utilities.

Before you begin

- The `xsadmin` utility is provided as a sample of how you can create custom utilities for your deployment. The `xscmd` utility is provided as a supported utility for monitoring and administering your environment. For more information, see [Administering with the xscmd utility](#).
- For the `xsadmin` utility to display results, you must have created your data grid topology. Your catalog servers and container servers must be started. See [Starting and stopping stand-alone servers](#) for more information.
- Verify that the `JAVA_HOME` environment variable is set to use the runtime environment that installed with the product. If you are using the trial version of the product, you must set the `JAVA_HOME` environment variable.

About this task

The **xsadmin** sample utility uses an implementation of Managed Beans (MBeans). This sample monitoring application enables rapidly integrated monitoring capabilities that you can extend by using the interfaces in the `com.ibm.websphere.objectgrid.management` package. You can look at the source code of the **xsadmin** sample application in the `wxs_home/samples/xsadmin.jar` file in a stand-alone installation, or in the `wxs_home/xsadmin.jar` file in a WebSphere Application Server installation.

You can use the **xsadmin** sample utility to view the current layout and specific state of the data grid, such as map content. In this example, the layout of the data grid in this task consists of a single *ObjectGridA* data grid with one *MapA* map that belongs to the *MapSetA* map set. This example demonstrates how you can display all active containers within a data grid and print filtered metrics regarding the map size of the *MapA* map. To see all possible command options, run the **xsadmin** utility without any arguments or with the **-help** option.

Procedure

1. Go to the bin directory.

```
cd wxs_home/bin
```

2. Run the **xsadmin** utility.

- To display the online help, run the following command:

UNIX

```
xsadmin.sh
```

Windows

```
xsadmin.bat
```

You must pass in only one of the listed options for the utility to work. If no **-g** or **-m** option is specified, the **xsadmin** utility prints out information for every grid in the topology.

- To enable statistics for all of the servers, run the following command:

UNIX

```
xsadmin.sh -g ObjectGridA -setstatsspec ALL=enabled
```

Windows

```
xsadmin.bat -g ObjectGridA -setstatsspec ALL=enabled
```

- To display all online containers for a grid, run the following command:

UNIX

```
xsadmin.sh -g ObjectGridA -m MapSetA -containers
```

Windows

```
xsadmin.bat -g ObjectGridA -m MapSetA -containers
```

All container information is displayed. An example of the output follows:

```
Connecting to Catalog service at localhost:1099
```

```
*** Show all online containers for grid - ObjectGridA & mapset - MapSetA
```

```
Host: 192.168.0.186
```

```
Container: server1_C-0, Server:server1, Zone:DefaultZone
```

```
Partition Shard Type
```

0 Primary

```
Num containers matching = 1
Total known containers = 1
Total known hosts = 1
```

Attention: To obtain this information when Transport Layer Security/Secure Sockets Layer (TLS/SSL) is enabled, you must start the catalog and container servers with the JMX service port set. To set the JMX service port, you can either use the **-JMXServicePort** option on the **start0gServer** script or you can call the `setJMXServicePort` method on the `ServerProperties` interface.

- To connect to the catalog service and display information about MapA, run the following command:

UNIX

```
xsadmin.sh -g ObjectGridA -m MapSetA -mapsizes -fm MapA
```

Windows

```
xsadmin.bat -g ObjectGridA -m MapSetA -mapsizes -fm MapA
```

The size of the specified map is displayed. An example of the output follows:
Connecting to Catalog service at localhost:1099

```
*****Displaying Results for Grid - ObjectGridA, MapSet - MapSetA*****
```

```
*** Listing Maps for server1 ***
```

Map Name	Partition	Map Size	Used Bytes (B)	Shard Type
MapA	0	0	0	Primary

- To connect to the catalog service using a specific JMX port and display information about the MapA map, run the following command:

UNIX

```
xsadmin.sh -g ObjectGridA -m MapSetA -mapsizes -fm MapA
-ch CatalogMachine -p 6645
```

Windows

```
xsadmin.bat -g ObjectGridA -m MapSetA -mapsizes -fm MapA
-ch CatalogMachine -p 6645
```

The **xsadmin** sample utility connects to the MBean server that is running on a catalog server. A catalog server can run as a stand-alone process, WebSphere Application Server process, or embedded within a custom application process. Use the **-ch** option to specify the catalog service host name, and the **-p** option to specify the catalog service naming port.

The size of the specified map is displayed. An example of the output follows:
Connecting to Catalog service at CatalogMachine:6645

```
*****Displaying Results for Grid - ObjectGridA, MapSet - MapSetA*****
```

```
*** Listing Maps for server1 ***
```

```
Map Name: MapA Partition #: 0 Map Size: 0 Shard Type: Primary
Server Total: 0
```

- To connect to a catalog service hosted in a WebSphere Application Server process, perform the following steps:

The **-dmgr** option is required when connecting to a catalog service hosted by any WebSphere Application Server process or cluster of processes. Use the **-ch** option to specify the host name if not `localhost`, and the **-p** option to

override the catalog service bootstrap port, which uses the process `BOOTSTRAP_ADDRESS`. The `-p` option is only needed if the `BOOTSTRAP_ADDRESS` is not set to the default of 9809.

Note: The stand-alone version of WebSphere eXtreme Scale cannot be used to connect to a catalog service hosted by a WebSphere Application Server process. Use the `xsadmin` that is script included in the `was_root/bin` directory, which is available when the installing WebSphere eXtreme Scale on WebSphere Application Server or WebSphere Application Server Network Deployment.

- a. Navigate to the WebSphere Application Server bin directory:

```
cd was_root/bin
```

- b. Launch the `xsadmin` utility using the following command:

UNIX

```
xsadmin.sh -g ObjectGridA -m MapSetA -mapsizes -fm MapA -dmgr
```

Windows

```
xsadmin.bat -g ObjectGridA -m MapSetA -mapsizes -fm MapA -dmgr
```

The size of the specified map is displayed.

Connecting to Catalog service at localhost:9809

```
****Displaying Results for Grid - ObjectGridA, MapSet - MapSetA****
```

```
*** Listing Maps for server1 ***
```

```
Map Name: MapA Partition #: 0 Map Size: 0 Shard Type: Primary  
Server Total: 0
```

- To display the configured and runtime placement of your configuration, run one of the following commands:

```
xsadmin -placementStatus  
xsadmin -placementStatus -g myOG -m myMapSet  
xsadmin -placementStatus -m myMapSet  
xsadmin -placementStatus -g myOG
```

You can scope the command to display placement information for the entire configuration, a single data grid, a single map set, or a combination of a data grid and map set. An example of the output follows:

```
*****Printing Placement Status for Grid - Grid, MapSet - mapSet*****
```

```
<objectGrid name="Grid" mapSetName="mapSet">  
  <configuration>  
    <attribute name="placementStrategy" value="FIXED_PARTITIONS"/>  
    <attribute name="numInitialContainers" value="3"/>  
    <attribute name="minSyncReplicas" value="0"/>  
    <attribute name="developmentMode" value="true"/>  
  </configuration>  
  <runtime>  
    <attribute name="numContainers" value="3"/>  
    <attribute name="numMachines" value="1"/>  
    <attribute name="numOutstandingWorkItems" value="0"/>  
  </runtime>  
</objectGrid>
```

Creating a configuration profile for the `xsadmin` utility

You can save your frequently specified parameters for the `xsadmin` utility in a properties file. As a result, the `xsadmin` utility calls are shorter.

Before you begin

Create a basic deployment of WebSphere eXtreme Scale that includes at least one catalog server and at least one container server. For more information, see **startOgServer** script (ORB).

About this task

See “**xsadmin** utility reference” for a list of the properties that you can put in a configuration profile for the **xsadmin** utility. If you specify both a properties file and a corresponding parameter as a command line argument, the command line argument overrides the properties file value.

Procedure

1. Create a configuration profile properties file. This properties file should contain any global properties that you want to use in all your **xsadmin** command invocations.

Save the properties file with any name you choose. For example, you might place the file in the following path: `/opt/ibm/WebSphere/wxs71/ObjectGrid/security/<my.properties>`.

Replace `<my.properties>` the name of your file. For example, you might set the following properties in your file:

- `XSADMIN_TRUST_TYPE=jks`
- `XSADMIN_TRUST_PATH=/opt/ibm/WebSphere/wxs71/ObjectGrid/bin/security/key.jks`
- `XSADMIN_USERNAME=ogadmin`


2. Run the **xsadmin** utility with the properties file that you created. Use the **-profile** parameter to indicate the location of your properties file. You can also use the **-v** parameter to display verbose output.

```
./xsadmin.sh -l -v -password xsadmin -ssl -trustPass ogpass -profile /opt/ibm/WebSphere/wxs71/ObjectGrid/security/<my.properties>
```

xsadmin utility reference

You can pass arguments to the **xsadmin** utility with two different methods: with a command-line argument, or with a properties file.

xsadmin arguments

Note:  The **xsadmin** utility has now been deprecated. Use the **xscmd** utility instead. The **xscmd** utility is provided as a supported utility for monitoring and administering your environment. For more information, see Administering with the **xscmd** utility.

You can define a properties file for the **xsadmin** utility with Version 7.1 Fix 1 or later. By creating a properties file, you can save some of the frequently used arguments, such as the user name. The properties that you can add to a properties file are in the following table. If you specify both a property in a properties file and the equivalent command-line argument, the command-line argument value overrides the properties file value.

For more information about defining a properties file for the **xsadmin** utility, see “Creating a configuration profile for the **xsadmin** utility” on page 287.

Table 17. Arguments for the xsadmin utility

Command Line Argument	Equivalent Property Name in Properties File	Description and valid values
-bp	n/a	Indicates the listener port. Default: 2809
-ch	n/a	Indicates the JMX host name for the catalog server. Default: localhost
-clear	n/a	Clears the specified map. Allows the following filters: -fm
-containers	n/a	For each data grid and map set, displays a list of container servers. Allows the following filters: -fnp
-continuous	n/a	Specify this flag if you want continuous map size results to monitor the data grid. When you run this command with the -mapsizes argument, the map size is displayed every 20 seconds.
-coregroups	n/a	Displays all core groups for the catalog server. This argument is used for advanced diagnostics.
-dismissLink <catalog_service_domain>	n/a	Removes a link between 2 catalog service domains. Provide the name of the foreign catalog service domain to which you previously connected with the -establishLink argument.
-dmgr	n/a	Indicates if you are connecting to a WebSphere Application Server hosted catalog service. Default: false
-empties	n/a	Specify this flag if you want to show empty containers in the output.
-establishLink <foreign_domain_name> <host1:port1,host2:port2...>	n/a	Connects the catalog service domain to a foreign catalog service domain. Use the following format: -establishLink <foreign_domain_name> <host1:port1,host2:port2...>. <i>foreign_domain_name</i> is the name of the foreign catalog service domain, and <i>host1:port1,host2:port2...</i> is a comma-separated list of catalog server host names and Object Request Broker (ORB) ports that are running in this catalog service domain.
-fc	n/a	Filters for only this container. If you are filtering container servers in a WebSphere Application Server Network Deployment environment, use the following format: <cell_name>/<node_name>/<serverName_containerSuffix> Use with the following arguments: -mapsizes, -teardown,-revisions,-getTraceSpec,-setTraceSpec,-getStatsSpec,-setStatsSpec
-fh	n/a	Filters for only this host. Use with the following arguments: -mapsizes, -teardown,-revisions,-getTraceSpec,-setTraceSpec,-getStatsSpec,-setStatsSpec,-routetable
-fm	n/a	Filters only for this map. Use with the following arguments: -clear, -mapsizes
-fnp	n/a	Filters servers that have no primary shards. Use with the following arguments: -containers
-fp	n/a	Filters for only this partition. Use with the following arguments: -mapsizes, -teardown,-revisions,-getTraceSpec,-setTraceSpec,-getStatsSpec,-setStatsSpec,-routetable
-fs	n/a	Filters for only this server. If you are filtering application servers in a WebSphere Application Server Network Deployment environment, use the following format: <cell_name>/<node_name>/<server_name> Use with the following arguments: -mapsizes, -teardown,-revisions,-getTraceSpec,-setTraceSpec,-getStatsSpec,-setStatsSpec

Table 17. Arguments for the `xsadmin` utility (continued)

Command Line Argument	Equivalent Property Name in Properties File	Description and valid values
-fst	n/a	Filters for only this shard type. Specify P for primary shards only, A for asynchronous replica shards only, and S for synchronous replica shards only. Use with the following arguments: <code>-mapsizes</code> , <code>-teardown</code> , <code>-revisions</code> , <code>-getTraceSpec</code> , <code>-setTraceSpec</code> , <code>-getStatsSpec</code> , <code>-setStatsSpec</code>
-fz	n/a	Filters for only this zone. Use with the following arguments: <code>-mapsizes</code> , <code>-teardown</code> , <code>-revisions</code> , <code>-getTraceSpec</code> , <code>-setTraceSpec</code> , <code>-getStatsSpec</code> , <code>-setStatsSpec</code> , <code>-routetable</code>
-force	n/a	Forces the action that is in the command, disabling any preemptive prompts. This argument is useful for running batched commands.
-g	n/a	Specifies the ObjectGrid name.
-getstatsspec	n/a	Displays the current statistics specification. You can set the statistics specification with the <code>-setstatsspec</code> argument. Allows the following filters: <code>-fst -fc -fz -fs -fh -fp</code>
-getTraceSpec	n/a	Displays the current trace specification. You can set the trace specification with the <code>-settracespec</code> argument. Allows the following filters: <code>-fst -fc -fz -fs -fh -fp</code>
-h	n/a	Displays the help for the <code>xsadmin</code> utility, which includes a list of arguments.
-hosts	n/a	Displays all of the hosts in the configuration.
-jmxUrl	XSADMIN_JMX_URL	Specifies the address of a JMX API connector server in the following format: <code>service:jmx:protocol:sap</code> . The <code>protocol</code> and <code>sap</code> variable definitions follow: <i>protocol</i> Specifies the transport protocol to be used to connect to the connector server. <i>sap</i> Specifies the address at which the connector server is found. For more information about the format of the JMX service URL, see Class <code>JMXServiceURL</code> (Java 2 Platform SE 5.0).
-l	n/a	Displays all known data grids and map sets.
-m	n/a	Specifies the name of the map set.
-mapsizes	n/a	Displays the size of each map on the catalog server to verify that key distribution is uniform over the shards. Allows the following filters: <code>-fm -fst -fc -fz -fs -fh -fp</code>
-mbeanservers	n/a	Displays a list of all MBean server end points.
-overridequorum	n/a	Overrides the quorum setting so that container server events are not ignored during a data center failure scenario.
-password	XSADMIN_PASSWORD	Specifies the password to log in to the <code>xsadmin</code> utility. Do not specify the password in your properties file if you want your password to remain secure.
-p	n/a	Indicates the JMX port for the catalog server host. Default: 1099 or 9809 for a WebSphere Application Server host, 1099 for stand-alone configurations.
-placementStatus	n/a	Displays the configured placement and runtime placement of your configuration. You can scope the output to a combination of data grids and map sets, or for the entire configuration: <ul style="list-style-type: none">• Entire configuration: <code>-placementStatus</code>• For a specific data grid: <code>-placementStatus -g my_grid</code>• For a specific map set: <code>-placementStatus -m my_mapset</code>• For a specific data grid and map set: <code>-placementStatus -g my_grid -m my_mapset</code>
-primaries	n/a	Displays a list of the primary shards.

Table 17. Arguments for the `xsadmin` utility (continued)

Command Line Argument	Equivalent Property Name in Properties File	Description and valid values
<code>-profile</code>	n/a	Specifies a fully qualified path to the properties file for the <code>xsadmin</code> utility.
<code>-quorumstatus</code>	n/a	Displays the status of quorum for the catalog service.
<code>-releaseShard</code> <container_server_name> <objectgrid_name> <map_set_name> <partition_name>	n/a	Used in conjunction with the <code>-reserveShard</code> argument. The <code>-releaseShard</code> argument must be invoked after a shard has been reserved and placed. . The <code>-releaseShard</code> argument invokes the <code>ContainerMBean.release()</code> method.
<code>-reserved</code>	n/a	Used with the <code>-containers</code> argument to display only shards that have been reserved with the <code>-reserveShard</code> argument.
<code>-reserveShard</code> <container_server_name> <objectgrid_name> <map_set_name> <partition_name>	n/a	Moves a primary shard to the specified container server. The <code>ContainerMBean.reserve()</code> method is invoked by this argument.
<code>-resumeBalancing</code> <objectgrid_name> <map_set_name>	n/a	Attempts to balance requests. Enables future rebalancing attempts on the specified ObjectGrid and map set.
<code>-revisions</code>	n/a	Displays revision identifiers for a catalog service domain including: each data grid, partition number, partition type (primary or replica), catalog service domain, lifetime ID, and number of data revisions for each specific shard. You can use this argument to determine if an asynchronous replica or linked domain is caught up. This argument invokes the <code>ObjectGridMBean.getKnownRevisions()</code> method. Allows the following filters: <code>-fst -fc -fz -fs -fh -fp</code>
<code>-routetable</code>	n/a	Displays the current state of the data grid from a client server perspective. The route table is the information that an ObjectGrid client server uses to communicate with the data grid. Use the route table as a diagnostic aid when you are trying to identify connection problems or <code>TargetNotAvailable</code> exceptions. Required arguments: In a stand-alone environment, you must specify the <code>-bp</code> and <code>-p</code> parameters with this argument if you are not using the default values for the bootstrap listener port and JMX port for the catalog server host. Allows the following filters: <code>-fz -fh -fp</code>
<code>-settracespec <trace_string></code>	n/a	Enables trace on servers during run time. See the following example: <code>-setTraceSpec "ObjectGridReplication=all=enabled"</code> See Collecting trace and Server trace options for more information about the trace strings that you can specify. Allows the following filters: <code>-fst -fc -fz -fs -fh -fp</code>
<code>-swapShardWithPrimary</code> <container_server_name> <objectgrid_name> <map_set_name> <partition_name>	n/a	Swaps the specified replica shard from the specified container server with the primary shard. By running this command, you can manually balance primary shards when needed.
<code>-setstatsspec <stats_spec></code>	n/a	Enables statistics gathering. This argument invokes the <code>DynamicServerMBean.setStatsSpec</code> and <code>DynamicServerMBean.getStatsSpec</code> methods. For more information, see Enabling statistics. Allows the following filters: <code>-fm -fst -fc -fz -fs -fh -fp</code>
<code>-suspendBalancing</code> <objectgrid_name> <map_set_name>	n/a	Prevents future attempts to balance the specified ObjectGrid and map set.
<code>-ssl</code>	n/a	Indicates that Secure Sockets Layer (SSL) is enabled.

Table 17. Arguments for the **xsadmin** utility (continued)

Command Line Argument	Equivalent Property Name in Properties File	Description and valid values
-teardown	n/a	Stops a list or group of catalog and container servers. Allows the following filters: -fst -fc -fz -fs -fh -fp Format to provide a list of servers: server_name_1,server_name_2 ... To stop all servers in a zone, include the -fz argument: -fz <zone_name> To stop all servers on a host, include the -fh argument: -fh <host_name>
-triggerPlacement	n/a	Forces shard placement to run, ignoring the configured numInitialContainers value in the deployment XML file. You can use this argument when you are performing maintenance on your servers to allow shard placement to continue running, even though the numInitialContainers value is lower than the configured value.
-trustPass	XSADMIN_TRUST_PASS	Specifies the password for the specified truststore.
-trustPath	XSADMIN_TRUST_PATH	Specifies a path to the truststore file. Example: etc/test/security/server.public
-trustType	XSADMIN_TRUST_TYPE	Specifies the type of truststore. Valid values: JKS, JCEK, PKCS12, and so on.
-unassigned	n/a	Displays a list of shards that cannot be placed on the data grid. Shards cannot be placed when the placement service has a constraint that is preventing placement.
-username	XSADMIN_USERNAME	Specifies the user name to log in to the xsadmin utility.
-v	n/a	Enables the verbose command-line action. Use this flag if you are using environment variables, a properties file, or both to specify certain command-line arguments, and want to view their values. See "Verbose option for the xsadmin utility" for more information.
-xml	n/a	Prints the unfiltered output from the PlacementServiceMBean.listObjectGridPlacement() method. The other xsadmin arguments filter the output of this method and organize the data into a more consumable format.

Verbose option for the **xsadmin** utility

You can use the **xsadmin** verbose option to troubleshoot problems. Run the **xsadmin -v** command to list all configured parameters. The verbose option displays all values in all scopes, including command line arguments, properties file arguments, and environment-specified arguments. The Effective arguments section includes the settings that are being used in the environment if you have specified the same property using multiple scopes.

Verbose option example

xsadmin command arguments:

The following text is an example of output when using the verbose option from the command line after you run the following command with a properties value specified:

```
./xsadmin -l -v -username xsadmin -password xsadmin -ssl -trustPass ogpass
-profile /opt/ibm/WebSphere/wxs71/ObjectGrid/security/my.properties
```

Properties file arguments:

The contents of the /opt/ibm/WebSphere/wxs71/ObjectGrid/security/my.properties properties file follow:

```

XSADMIN_TRUST_PASS=ogpass
XSADMIN_TRUST_TYPE=jks
XSADMIN_TRUST_PATH=/opt/ibm/WebSphere/wxs71/ObjectGrid/bin/security/key.jks
XSADMIN_USERNAME=ogadmin
XSADMIN_PASSWORD=ogpass

```

Command results:

In the following output from the preceding **xsadmin** command, the text that is in *bold italics* indicates properties and values that are specified both on the command line and in the properties file. In the Effective command line arguments section, you can see that the command line specified arguments override the values in the properties file.

```

Command line specified arguments
*****
XSADMIN_USERNAME=xsadmin
XSADMIN_PASSWORD=xsadmin
XSADMIN_TRUST_PATH=<unspecified>
XSADMIN_TRUST_TYPE=<unspecified>
XSADMIN_TRUST_PASS=ogpass
XSADMIN_PROFILE=/opt/ibm/WebSphere/wxs71/ObjectGrid/security/my.properties
XSADMIN_JMX_URL=<unspecified>
*****
Properties file specified arguments
*****
XSADMIN_USERNAME=ogadmin
XSADMIN_PASSWORD=ogpass
XSADMIN_TRUST_PATH=/opt/ibm/WebSphere/wxs71/ObjectGrid/bin/security/key.jks
XSADMIN_TRUST_TYPE=jks
XSADMIN_TRUST_PASS=ogproppass
XSADMIN_JMX_URL=<unspecified>
*****
Environment-specified arguments
*****
XSADMIN_USERNAME=<unspecified>
XSADMIN_PASSWORD=<unspecified>
XSADMIN_TRUST_PATH=<unspecified>
XSADMIN_TRUST_TYPE=<unspecified>
XSADMIN_TRUST_PASS=<unspecified>
XSADMIN_JMX_URL=<unspecified>
*****
Effective arguments
*****
XSADMIN_USERNAME=xsadmin
XSADMIN_PASSWORD=xsadmin
XSADMIN_TRUST_PATH=/opt/ibm/WebSphere/wxs71/ObjectGrid/bin/security/key.jks
XSADMIN_TRUST_TYPE=jks
XSADMIN_TRUST_PASS=ogpass
XSADMIN_PROFILE=/opt/ibm/WebSphere/wxs71/ObjectGrid/security/my.properties
XSADMIN_JMX_URL=<unspecified>
SSL authentication enabled: true
*****
Connecting to Catalog service at localhost:1099
*** Show all 'objectGrid:mapset' names
Grid Name  MapSet Name
accounting defaultMapSet

```

Attention: The XSADMIN_PROFILE property, although it displays in the verbose output, is not a valid key that you can specify in a properties file. The value of this property in the verbose output indicates the property value that is being used, as indicated in the **-profile** command line argument.

Output without the verbose option

An example of the same command output without the verbose option enabled follows:

```
./xsadmin -l -username xsadmin -password xsadmin -ssl -trustPass ogpass  
-profile /opt/ibm/WebSphere/wxs71/ObjectGrid/security/my.properties  
Connecting to Catalog service at localhost:1099  
*** Show all 'objectGrid:mapset' names  
Grid Name  MapSet Name  
accounting defaultMapSet
```

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, New York 10594 USA

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Mail Station P300
522 South Road
Poughkeepsie, NY 12601-5400
USA
Attention: Information Requests

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at “Copyright and trademark information” at www.ibm.com/legal/copytrade.shtml.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linux is a trademark of Linus Torvalds in the U.S., other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft 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.

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

Index

A

- AP 169
- APIs
 - DataSerializer 78
- architecture
 - clients 21
 - container servers 18
 - maps 20
 - overview 16
 - partitions 18
 - shards 18
 - topologies 147
- availability
 - connectivity 93
 - failure 93
 - map set replication 115
 - overview 93
 - replication 96
- availability partition (AP) 169

B

- benefits
 - write-behind caching 59, 159

C

- cache 284
 - distributed 152
 - embedded 151
 - local 148
 - overview 16
 - technical overview 15
- cache integration
 - overview 34
- capacity planning 274
- catalog service
 - overview 17
- catalog service domains 101
- ClassAlias 189
- client connections
 - administering
 - using JCA 254
- coherent cache 54, 154
- complete cache 55, 156
- connection factories
 - configuring 245
 - configuring Eclipse environments 246
 - creating resource references 247
- container servers
 - high availability 101
 - overview 18
 - per-container placement 83

D

- data access
 - overview 131

- data access (*continued*)
 - partitions 131
 - queries 131
 - stored data 131
 - transactions 131
- data grids 79
- data types 191
- database
 - data preloading 62, 162
 - data preparation 62, 162
 - database synchronization
 - techniques 64, 164
 - read-through cache 56, 157
 - side cache 56, 156
 - sparse and complete cache 55, 156
 - synchronization 64, 164
 - write-behind cache 59, 159
 - write-through cache 56, 157
- deadlocks
 - scenarios for 126
- directory conventions 13
- distributed cache 152
- distributing changes
 - using Java message service 140
- dynamic cache
 - configuration files
 - modify 274
 - configuring 267, 280
 - overview 47, 267
- dynamic cache provider
 - introduction 47, 267

E

- Eclipse Equinox
 - environment setup 219
- embedded cache 151
- enableXm property 184
- enterprise data grid 22, 183
- event-based validation 65, 165
- evictors
 - overview 30
- examples 281
- exclusive lock 126
- eXtreme data format
 - configuring 186
- eXtreme IO 184
- eXtreme memory 184
- eXtreme Scale overview 1
 - free trial 284
- Extreme Transaction Processing 1
 - free trial 284
- eXtremeIO
 - configuration 184
- eXtremeMemory
 - configuration 184

F

- FieldAlias 189
- free trial 284

H

- HTTP session failover
 - Liberty profile 254, 255
- HTTP session manager
 - overview 45

I

- in-line cache 56, 156
- indexes
 - data quality 67, 167
 - performance 67, 167
- integrating with other servers 181
- isolation
 - for transactions 133
 - pessimistic locking 133
 - repeatable read 133

J

- Java Persistence API (JPA)
 - cache plug-in
 - introduction 39
 - cache topology
 - embedded 39
 - embedded partitioned 39
 - remote 39
- JCA
 - administering
 - client connections 254

L

- legal
 - terms and conditions 11
- Liberty profile
 - configuring HTTP session failover 254
 - configuring unique clone IDs 257
 - enabling HTTP session failover 255
 - generating plug-in configuration files 258
 - merging plug-in configuration files 258
- Liberty runtime environment
 - overview 34
- Linux shell
 - overview 34
- load balancing
 - map sets 115
 - replicas 110
 - replication 96
- loaders
 - database 61, 161

- loaders (*continued*)
 - Java Persistence API (JPA)
 - overview 69
- local cache
 - peer replication 149
- locking
 - optimistic 122
 - pessimistic 122
 - strategies for 122
- locks
 - compatibility 126
 - life cycle 126
 - time out 126

M

- map preloading
 - load balancing 110
 - map sets 115
 - replication 96
- marshalling
 - overview 71
- maxXmSize property 184
- multi-master data grid replication
 - planning 169
- multi-master replication
 - configuration planning 173
 - design planning 176
 - planning 169
 - planning for loaders 174
- multimaster replication
 - topologies 169
- MVS console 34

N

- new features 4

O

- OSGi
 - administering applications 223
 - administering servers 223
 - administering services 236
 - building dynamic plug-ins 225
 - building plug-ins 224
 - configuring plug-ins 232
 - configuring servers 239
 - developing plug-ins 217
 - Eclipse Equinox environment 219
 - installing bundles 220
 - installing plug-ins 230
 - overview 33, 217
 - running containers 222
 - with non-dynamic plug-ins 232
 - running plug-ins 217
 - starting servers 234
- OSGi applications
 - overview 34
- OSGi container
 - Apache Aries Blueprint
 - configuration 228
- overview
 - product overview 1
 - technical overview 15

P

- partitions
 - fixed placement 83
 - introduction 81
 - overview 79
 - transactions 86, 134
 - with entities 81
- performance
 - load balancing 110
 - map set replication 115
 - replication 96
- placement
 - overview 79
 - strategies 83
- planning 147
- plug-ins
 - DataSerializer 78
 - ObjectTransformer 74
- properties
 - samples 284

Q

- quorums
 - overview 102

R

- release notes 7
- replicas
 - reading data 109
- replication
 - loaders 106
 - memory cost 106
 - shard types 106
- requirements
 - hardware 12
 - software 12
- resource adapters
 - installing 243
- REST data service
 - overview 145
 - planning 145

S

- SAF registry
 - overview 34
- sample code 281
- samples 281
- scalability
 - overview 79
 - with units or pods 92
- scenarios 183
- security
 - authentication 143
 - authorization 143
 - J2C client connections 248
 - secure transport 143
- serialization 22, 71, 183
 - Java 73
 - overview 78
- server properties
 - enableXm 184
 - maxXmSize 184

- server properties (*continued*)
 - xIOContainerTCPNonSecurePort 184
- session manager interoperability
 - with WebSphere products 181
- sessions 45
- shards
 - allocation 108
 - failure 110
 - lifecycle 110
 - placement 79
 - primary 108
 - recovery 110
 - replica 108
- shared lock 126
- side cache
 - database integration 56, 156
- sparse cache 55, 156
- stand-alone servers
 - starting 192
- starting
 - servers 192
- support 7

T

- topologies
 - clients 21
 - container servers 18
 - maps 20
 - overview 16
 - plan 147
- transactions
 - connecting applications 240
 - copyMode 120
 - cross-grid 86, 134
 - data access 131
 - developing client components 249
 - overview 117
 - processing 119, 241
 - processing overview 116
 - single-partition 86, 134
- transport 184
- transports
 - eXtremeIO 184
- troubleshooting
 - release notes 7
- tutorials 281
- two-phase commit
 - error recovery
 - overview 141

U

- upgradeable lock 126

W

- write-behind
 - database integration 59, 159

X

- XDF 186
- xIOContainerTCPNonSecurePort
 - property 184

- xsadmin utility
 - commands 288
 - configuration profile 288
 - monitoring 284
 - verbose output 292

Z

- zones
 - overview 26



Printed in USA