WebSphere eXtreme Scale 8.5.0

Tables of Contents

CWXQY1215E
CWXQY1217E
CWXQY1220E
CWXQY1221E
CWXQY1222E
CWXQY1223E
CWXQY1225E
CWXQY1227E
CWXQY1229E
CWXQY1230E
CWXQY1231E
CWXQY1232E
CWXQY1233E
CWXQY1234E
CWXQY1235E
CWXQY1236E
CWXQY1238E
CWXQY1240E
CWXQY1241E
CWXQY1242E
CWXQY1243E
CWXQY1244E
CWXQY1246E
CWXQY1247E
CWXQY1248E
CWXQY1249E
CWXQY1250E
CWXQY1251E
CWXQY1252E
CWXQY1253E
CWXQY1254E
CWXQY1255E
CWXQY1256E
CWXQY1257E
CWXQY1258E
CWXQY1259E
CWXQY1260E
CWXQY1261E
CWXQY1262E
CWXQY1263E
CWXQY1264E
CWXQY1265E
CWXQY1266E
CWXQY1267E
CWXQY1268E
CWXQY1269E
CWXQY1270E
CWXQY1271E
CWXQY1272E
CWXQY1282E
CWXQY1283E
CWXQY1285E
CWXQY1286E
CWXQY1287E
CWXQY1288E
CWXQY1289E
CWXQY1290E
CWXQY1291E
CWXQY1292E
CWXQY1293E
CWXQY1294E
CWXQY1296E
CWXQY1297E
CWXQY1298E
CWXQY1299E
CWXQY1300E
CWXQY1301E
CWXQY1302E

CWXSI0057E
CWXSI0058I
CWXSI0059E
CWXSI0060W
CWXSI0061I
CWXSI0062E
CWXSI0063W
CWXSI0064W
CWXSI0065E
CWXSI0066E
CWXSI0067E
CWXSI0068I
CWXSI0069W
CWXSI0070I
CWXSI0071I
CWXSI0072E
CWXSI0073W
CWXSI0074W
CWXSI0075E
CWXSI0076E
CWXSI0077E
CWXSI0078E
CWXSI0079E
CWXSI0080E
CWXSI0081E
CWXSI0082E
CWXSI0083E
CWXSI0084E
CWXSI0085W
CWXSI0086W

CWXSR: Messages for the log analyzer component

CWXSR0001I
CWXSR0002E
CWXSR0003I
CWXSR0004I
CWXSR0005I
CWXSR0006I
CWXSR0007I
CWXSR0008I
CWXSR0009E
CWXSR0010W
CWXSR0011I
CWXSR0012E
CWXSR0013W
CWXSR0500I

SESN: Messages for the WebSphere eXtreme Scale HTTP Session manager

SESN0006E
SESN0007E
SESN0008E
SESN0012E
SESN0013E
SESN0066E
SESN0116W
SESN0117I
SESN0118W
SESN0119W
SESN0120I
SESN0121E
SESN0122E
SESN0123E
SESN0124W
SESN0169I
SESN0170W
SESN0171W
SESN0172I
SESN0175I
SESN0176I
SESN0194W

SESN0195I
SESN0196W

SSLC: Messages for SSL channel security
SSLC0001W
SSLC0002E
SSLC0003E
SSLC0004W
SSLC0005E
SSLC0006E
SSLC0007W
SSLC0008E
SSLC0009E
SSLC0010E
SSLC0011E
SSLC0012E
SSLC0013E

TCPC: Messages for the TCP channel component
TCPC0001I
TCPC0002I
TCPC0003E
TCPC0004W
TCPC0005W
TCPC0006E
TCPC0007E
TCPC0801E
TCPC0802E
TCPC0803E
TCPC0804E
TCPC0805E
TCPC0806E
TCPC0807E
TCPC0808E
TCPC0809E
TCPC0810E
TCPC0811E
TCPC0812E
TCPC0813W
TCPC0814W
TCPC0815E
TCPC0816E
TCPC0817E
TCPC0818E
TCPC0819E

WSBB: Messages for the XsByteBuffer component
WSBB0861E
WSBB0862W
WSBB0863E
WSBB0864E

XSMI: Messages for WebSphere eXtreme Scale migration
XSMI0000E
XSMI0001I
XSMI0002I
XSMI0003I
XSMI0004I
XSMI0005I
XSMI0006I
XSMI0007I
XSMI0008I
XSMI0009I
XSMI0010I
XSMI0011I
XSMI0012I
XSMI2001I
XSMI2002I
XSMI2003I
XSMI2004I
XSMI2005I
XSMI2006I

Glossary

Site map

IBM WebSphere eXtreme Scale Version V8.5 documentation

Welcome to the IBM WebSphere eXtreme Scale Version V8.5 documentation, where you can find information about how to install, maintain, and use the IBM WebSphere eXtreme Scale Version V8.5.

Getting started

Product overview

What's new

Getting started

Installing

Configuring

System requirements

Common tasks

Administering

Developing applications

Monitoring

Security

Troubleshooting and support

Troubleshooting

Support portal

Fix central

Technotes

More information

WebSphere eXtreme Scale V8.5 Education

WebSphere eXtreme Scale V8.5 tutorials

Articles

Redbooks

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.
- What’s new in Version 8.5
  WebSphere eXtreme Scale includes many new features in Version 8.5. Use this topic to learn about the latest product updates.
- 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.
- Notices
- Privacy policy considerations
- Terms and conditions for information centers
  Permissions for the use of these publications is granted subject to the following terms and conditions.
- 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.

- Directory conventions
  The following directory conventions are used throughout the documentation to 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.
- 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.
- 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.
- 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.
- 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.
- Serialization overview
  Data is always expressed, but not necessarily stored, as Java™ objects in the data grid. WebSphere eXtreme Scale uses multiple Java processes to serialize the data, by converting the Java object instances to bytes and back to objects again, as needed, to move the data between client and server processes.
- 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.
- Availability overview
- Transaction processing overview
  WebSphere eXtreme Scale uses transactions as its mechanism for interaction with data.
- Security overview
  WebSphere eXtreme Scale can secure data access, including allowing for integration with external security providers.
- REST data services overview
  The WebSphere eXtreme Scale REST data service is a Java HTTP service that is compatible with Microsoft WCF Data Services (formally 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.

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.

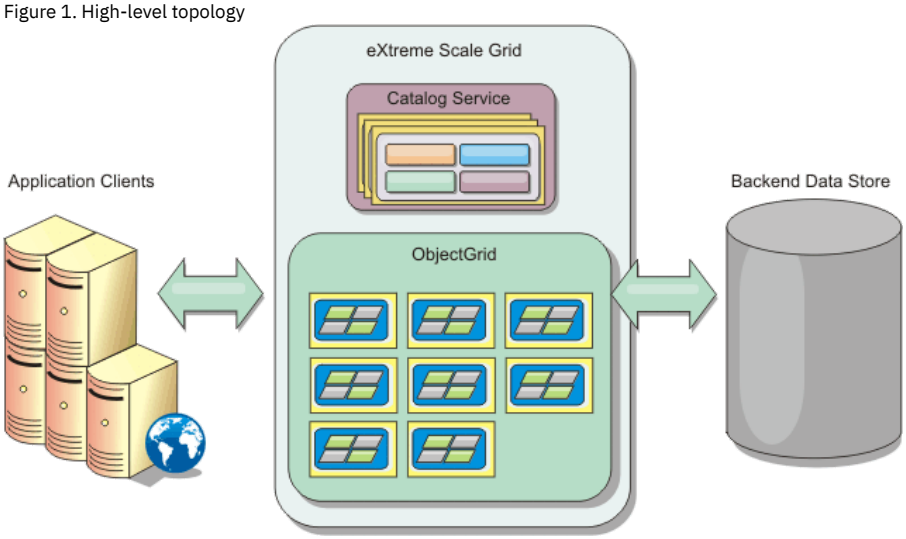
To tackle the problem of this type of failure, eXtreme Scale splits the given data set into partitions, which are equivalent to constrained tree schemas. Constrained tree schemas describe the relationship between entities. When you are using partitions, the entity relationships must model a tree data 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 record resolves the problem of having to maintain many copies of the same data, and it also prevents any version conflicts that might occur. 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 more 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 8.5

WebSphere® eXtreme Scale includes many new features in Version 8.5. Use this topic to learn about the latest product updates.

8.5

Installation Manager

You can now install WebSphere eXtreme Scale using Installation Manager. Learn more...
8.5

All 64-bit Java Runtime Environments supported

You can now install the product and run the monitoring console on all 64-bit platforms. Learn more...
8.5

View cache entries, query and invalidate operations in the web console and xscmd utility

You can view the keys of the cache entries in your data grid in the web console and **xscmd** utility. You can run regular expressions on the keys in your data grid to retrieve sets of data from a map. You can also invalidate sets of data based on the regular expression or partition. Learn more...
8.5

Retrieve environment information with the xscmd utility

You can use the **xscmd** utility with the **-c showinfo** command to view important details about the servers running in your WebSphere eXtreme Scale environment. Learn more...
8.5

Spring cache provider

With Spring cache abstraction, you can transparently add caching to your existing Spring application that uses the WebSphere eXtreme Scale as the cache provider. Learn more...
8.5

Liberty profile support

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 Version 8.5 with WebSphere Application Server Version 8.5. Because the Liberty profile does not include a Java runtime environment (JRE), you must install a JRE that is provided by either Oracle or IBM. Learn more...

8.5

JTA support for connecting applications to eXtreme Scale clients

WebSphere eXtreme Scale provides support for 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. 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. Learn more...

8.5

CatalogDomainManager API for connecting to specific catalog domains using JTA

The CatalogDomainManager API allows WebSphere eXtreme Scale clients and servers that run in WebSphere Application Server to connect to specific catalog service domains that are defined in the administrative console. Learn more...

Related reference:
Deprecated properties and APIs
Removed properties and APIs

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
- Contacting IBM Software Support

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 8.5, 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

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.

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.

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

8.5+ 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.

8.5+

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 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

8.5

- WebSphere Application Server Version 6.1.0.41 or later
- WebSphere Application Server Version 7.0.0.19 or later
- WebSphere Application Server Version 8.0.0.2 or later
- WebSphere Application Server Version 8.5.0.1 or later

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

Java requirements

8.5 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 5 or later.

Directory conventions

The following directory conventions are used throughout the documentation to 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

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.
- 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 predicable 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.
- Evictors
  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.
- 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.

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.

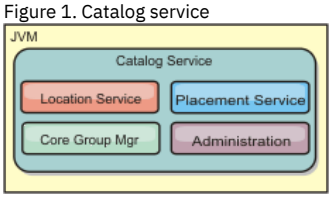
- Catalog service
  The catalog service controls placement of shards and discovers and monitors the health of container servers in the data grid. The catalog service hosts logic that should be idle and has little influence on scalability. It is built to service hundreds of container servers that become available simultaneously, and run services to manage the container servers.
- 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. As a result, each container server hosts a subset of the complete data. A JVM might host one or more shard containers and each shard container can host multiple shards.
- Maps
  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.
- 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.

Related concepts:
Planning the topology

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 1. 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. To provide 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.

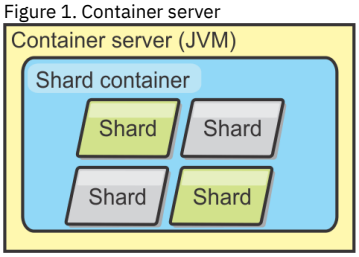
For high availability, configure a catalog service domain. A catalog service domain consists of multiple Java virtual machines, including a master JVM and a number of backup Java virtual machines. For more information, see High availability catalog service.

Related concepts:
High availability catalog service
Related tasks:
Configuring catalog servers and catalog service domains
Configuring the quorum mechanism
Tuning the heartbeat interval setting for failover detection
Configuring WebSphere eXtreme Scale with WebSphere Application Server
Configuring the catalog service in WebSphere Application Server
Creating catalog service domains in WebSphere Application Server
Configuring catalog and container servers
Starting and stopping stand-alone servers
Using the embedded server API to start and stop servers
Configuring WebSphere Application Server applications to automatically start container servers
Configuring container servers in WebSphere Application Server
Controlling placement
Managing data center failures
Managing data center failures when quorum is enabled
Administering with the xscmd utility
Related reference:
Server properties file
startOgServer script
Catalog service domain administrative tasks
ObjectGrid descriptor XML file
Deployment policy descriptor XML file

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. As a result, each container server hosts a subset of the complete data. A JVM might host one or more shard containers and each shard container can host multiple shards.

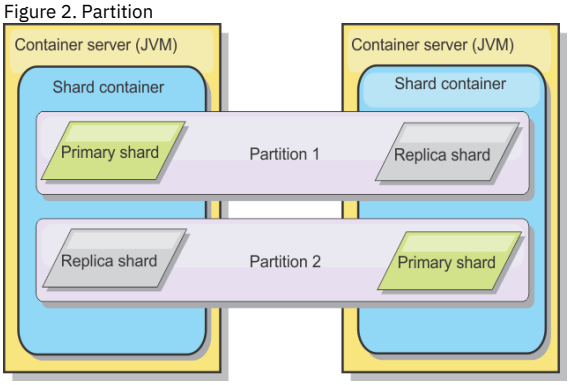
Remember: Plan out the heap size for the container servers, which host all of your data. Configure the heap settings accordingly.
Figure 1. Container server



Partitions host a subset of the data in the grid. WebSphere® eXtreme Scale automatically places partitions in a container. A partition consists of one primary shard and optional replica shards. When more container servers become available, replica shards are created and placed. Existing primary and replica shards are also distributed to new containers to maintain an equals number of shards on each container server.

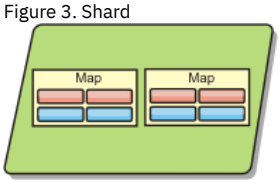
Important: Before final deployment, choose the number of partitions carefully. WebSphere eXtreme Scale uses a hash code to locate partitions in the network and this number cannot be changed dynamically. As a general rule, you can overestimate the number of partitions.

Figure 2. Partition



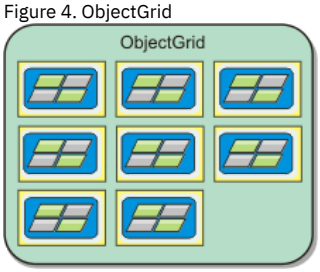
Shards are instances of partitions and have one of two roles: primary or replica. The primary shard and its replicas make up the physical manifestation of the partition. Every partition has several shards that each host all of the data contained in that partition. One shard is the primary, and the others 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 3. 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 4. ObjectGrid

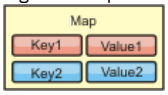


Related tasks:
Configuring catalog and container servers
Starting and stopping stand-alone servers
Using the embedded server API to start and stop servers
Configuring the catalog service in WebSphere Application Server
Configuring WebSphere Application Server applications to automatically start container servers
Configuring container servers in WebSphere Application Server
Controlling placement
Related reference:
Server properties file
ObjectGrid descriptor XML file
Deployment policy descriptor XML file

Maps

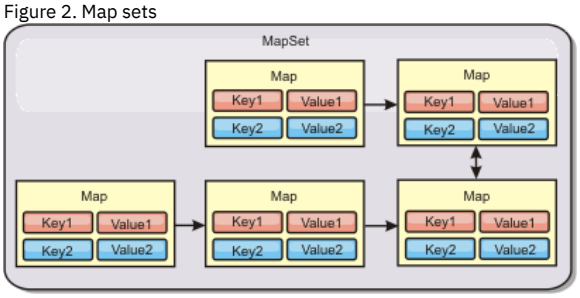
A map is a container for key-value pairs, which allows an application to store a value indexed by a key. Maps support indexes that can be added to index attributes on the key or value. These indexes are automatically used by the query runtime to determine the most efficient way to run a query.

Figure 1. 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 2. 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 in each of the maps using the ObjectMap API for Java 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

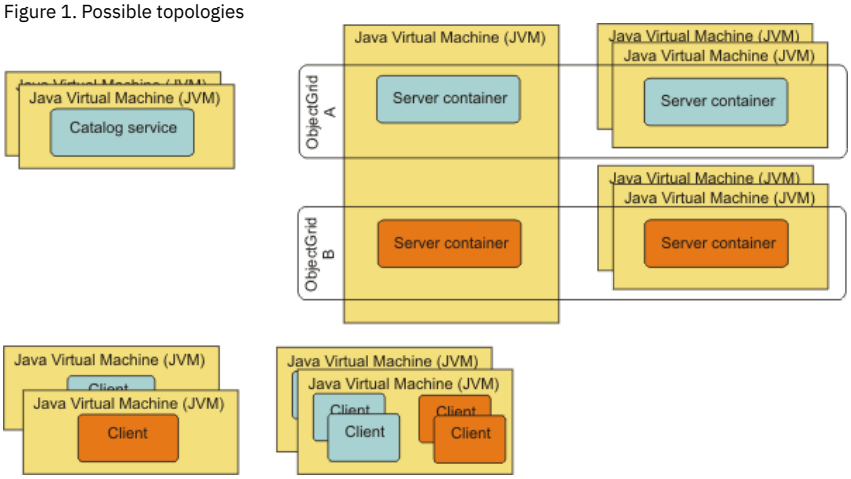
Clients connect to a catalog service, retrieve a description of the server topology, and communicate directly to each server as needed. When the server topology changes because new servers are added or existing servers have failed, the dynamic catalog service routes the client to the appropriate server 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.

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 1. Possible topologies



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.

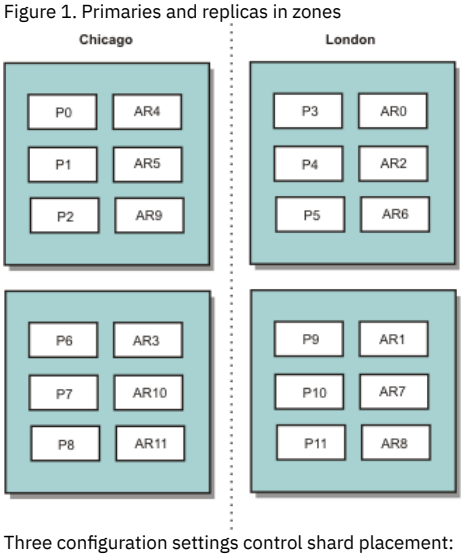
Another use case is in a virtualized environment where many server instances, each with a unique IP address, run on the same physical server.

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 1. Primaries and replicas in zones



Three configuration settings control shard placement:

- Deployment file
- Group containers
- Specify rules

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

Development mode

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

This simple step activates the first 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.

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. Shards are placed so that shards for the same partition are placed in separate zones.

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 deployment policy descriptor XML file. 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 WebSphere® eXtreme Scale

When you are upgrading WebSphere 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, 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 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 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.

Related tasks:
Controlling shard placement with zones
Defining zones for container servers
Viewing zone information with the xscmd utility
Administering with the xscmd utility
Related reference:
Example: Zone definitions in the deployment policy descriptor XML file
Deployment policy descriptor XML file

Evictors

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 can also write your own evictors: For more information, see Custom evictors.

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.

Related concepts:
Tuning evictors
Plug-ins for evicting cache objects
Custom evictors
Related tasks:
Configuring evictors with XML files
Enabling evictors programmatically
Configuring evictors with XML files
Related reference:
ObjectGrid descriptor XML file

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 in 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, enabling sophisticated options to integrate with other services 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 server feature of the eXtreme Scale stand-alone installation. It is required for running eXtreme Scale servers and can also be used for runningeXtreme 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 client feature of the eXtreme Scale stand-alone 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.

Next topic: Installing the Eclipse Equinox OSGi framework with Eclipse Gemini for clients and servers
Related tasks:
Programming to use the OSGi framework
Installing the Eclipse Equinox OSGi framework with Eclipse Gemini for clients and servers
Updating OSGi services for eXtreme Scale plug-ins with xscmd
Managing plug-in life cycles
Installing the Eclipse Equinox OSGi framework with Eclipse Gemini for clients and servers
Related reference:
Server properties file
Related information:
API documentation
Introduction: Starting and configuring the eXtreme Scale server and container to run plug-ins in 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.

8.5+ 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.
- JPA level 2 (L2) cache plug-in
  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.
- 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.
- 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.
  .

Liberty profile

8.5+ 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
- z/OS z/OS workload 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.

z/OS On z/OS systems, the Liberty profile provides an operations environment. You can work natively with this environment using the MVS™ console. For application development, consider using the Eclipse-based developer tools on a separate distributed system, on Mac OS, or in a Linux shell on z/OS.

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.

Related tasks:
Configuring eXtreme Scale servers to use the Liberty profile
Installing the Liberty profile
Related reference:
Liberty profile server properties
Server properties file

Data caching and the Liberty profile

8.5+ 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>
```

REST feature

Use the Representational State Transfer (REST) gateway to access simple data grids that are hosted by a collective 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.rest-1.1</feature>
</featureManager>

<xsRest/>
</server>
```

JPA level 2 (L2) cache plug-in

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 both OpenJPA and Hibernate Version 3.0.

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.

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 or Intra-domain topology.
   - If the data cannot fit into a single JVM heap, use the Embedded, partitioned topology, or Remote topology
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: local read, remote write

- Intra-domain topology: local read, local write
- Embedded, partitioned topology: Partitioned: remote read, remote write
- Remote topology: 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 or Intra-domain topology.

   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.

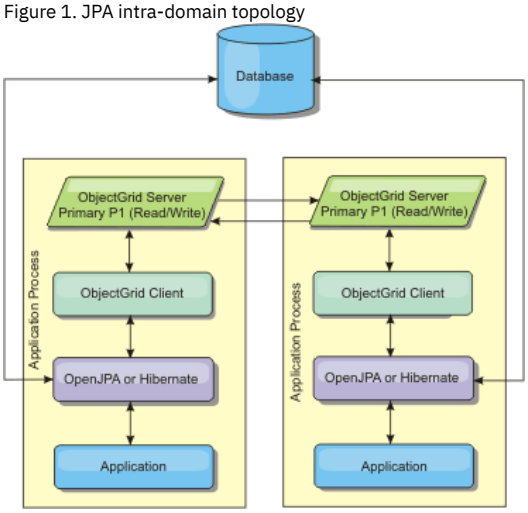
4. **What is the tolerated staleness level of the data?**
   In a system with few JVMs, data replication latency exists for write operations. The goal of the cache is to maintain an ultimate synchronized data view across all JVMs. When you are using the intra-domain topology, a data replication delay 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 1. 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.

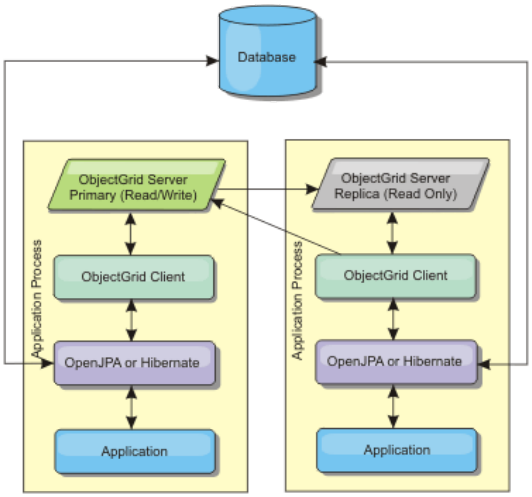
Limitations:

- 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 2. 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.

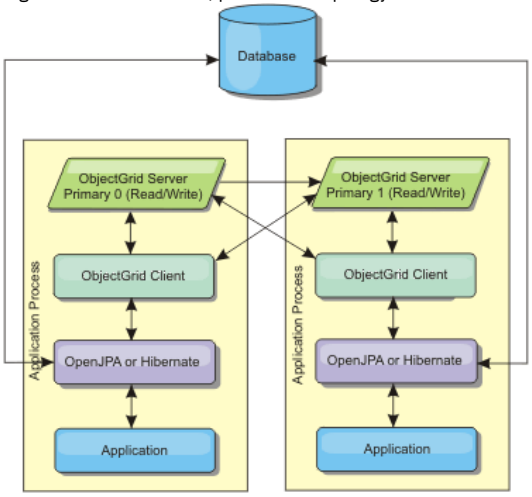
Limitations:

- 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 3. 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.

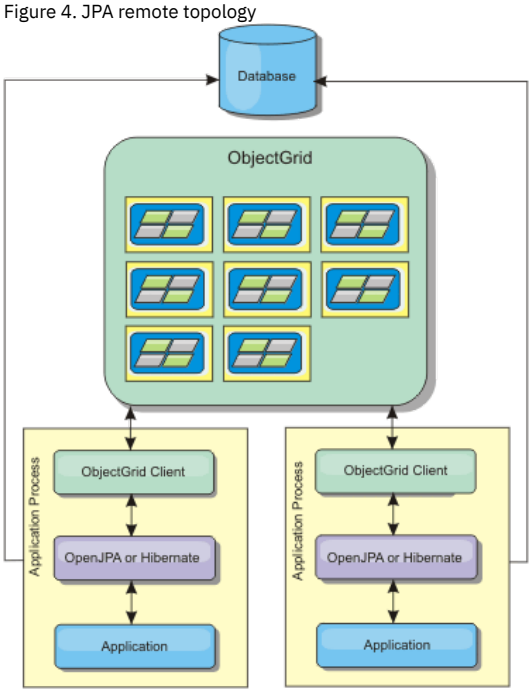
Read about Sizing memory and partition count calculation to maximize the performance of your chosen configuration.

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 4. 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.

- JPA cache configuration properties for both OpenJPA and Hibernate Version 3.0
  WebSphere eXtreme Scale includes level 2 cache plug-ins for both OpenJPA and Hibernate Java Persistence API (JPA) providers. To configure the L2 cache plug-in, you must update properties in the persistence.xml file.
- Configuring the OpenJPA cache plug-in
  You can configure both DataCache and QueryCache implementations for OpenJPA.
- Configuring the Hibernate cache plug-in
  You can enable the cache to use the Hibernate cache plug-in by specifying properties files.

Related tasks:
Configuring the OpenJPA cache plug-in
Troubleshooting multiple data center configurations
Configuring the Hibernate cache plug-in
Related reference:
JPA cache configuration properties for both OpenJPA and Hibernate Version 3.0
Example: OpenJPA ObjectGrid XML files
Example: Hibernate ObjectGrid XML files
Related information:
com.ibm.websphere.objectgrid.openJPA package
com.ibm.websphere.objectgrid.hibernate.cache package

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:

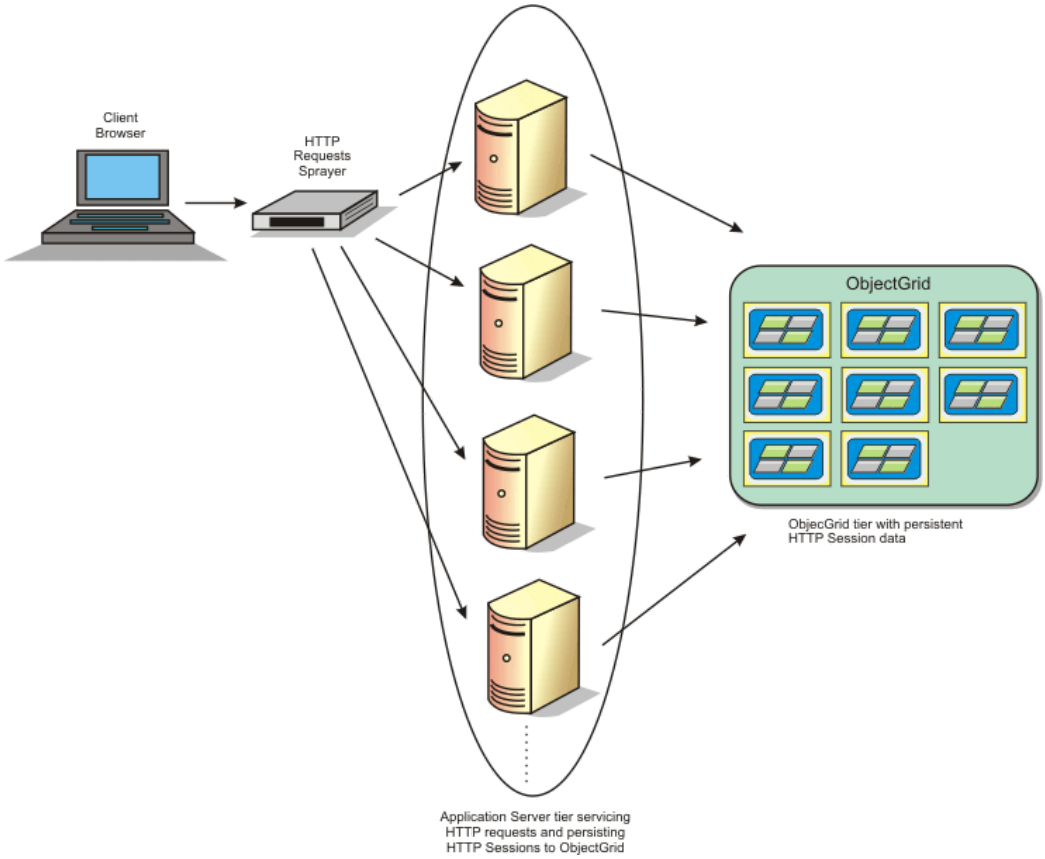
- In environments that use application servers at different versions of WebSphere Application Server, such as in a migration scenario.
- In deployments that use application servers from different vendors. For example, an application that is being developed on open source application servers and that 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 1. 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.

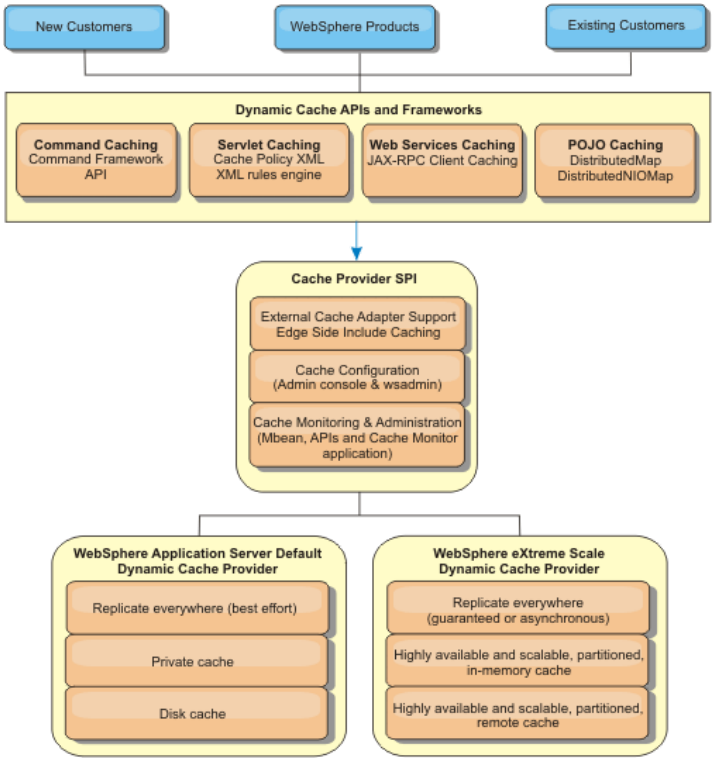
This approach for starting embedded ObjectGrid containers applies to any type of application server. The approaches involving a WebSphere Application Server component or WebSphere Application Server Community Edition GBean are deprecated.

Related tasks:
Configuring HTTP session managers
Configuring the HTTP session manager with WebSphere Application Server
Configuring WebSphere Application Server HTTP session persistence to a data grid
Configuring HTTP session manager with WebSphere Portal
Configuring the HTTP session manager for various application servers
Related reference:
XML files for HTTP session manager configuration
Servlet context initialization parameters
splicer.properties file

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.

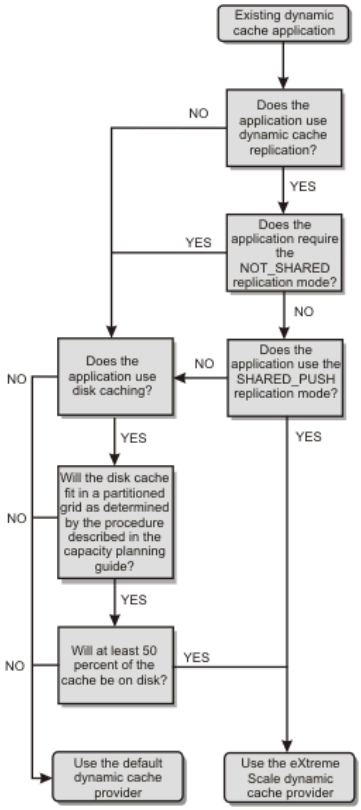


Deciding how to use WebSphere eXtreme Scale

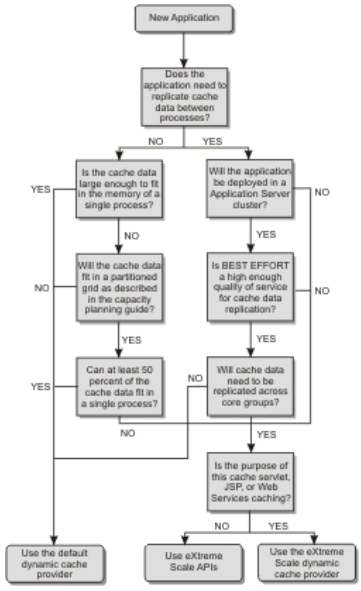
The available features in WebSphere eXtreme Scale significantly increase the distributed capabilities of the dynamic cache service beyond what is offered by the default dynamic cache provider and data replication service. With eXtreme Scale, you can create caches that are truly distributed between multiple 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

Cache features	Default provider	eXtreme Scale provider	eXtreme Scale API
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
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

Table 2. Seamless technology integration

Cache features	Default provider	eXtreme Scale provider	eXtreme Scale API
WebSphere Application Server servlet/JSP results caching	V5.1+	V6.1.0.25+	
WebSphere Application Server Web Services (JAX-RPC) result caching	V5.1+	V6.1.0.25+	
HTTP session caching			x
Cache provider for OpenJPA and Hibernate			x
Database synchronization using OpenJPA and Hibernate			x

Table 3. Programming interfaces

Cache features	Default provider	eXtreme Scale provider	eXtreme Scale API
Command-based API	Command framework API	Command framework API	DataGrid API
Map-based API	DistributedMap API	DistributedMap API	ObjectMap API
EntityManager API			x

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

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

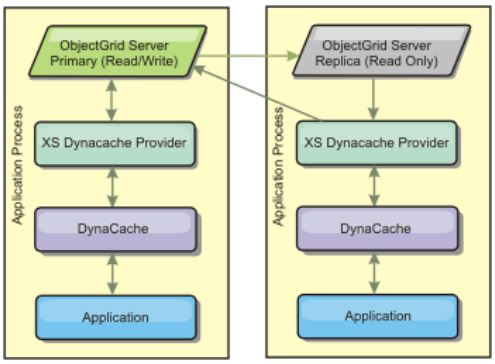
Topology

Deprecated: 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 any of three available topologies. With these topologies you can customize the cache specifically to performance, resource, and administrative needs. These topologies are: embedded, embedded partitioned, and remote.

Embedded topology

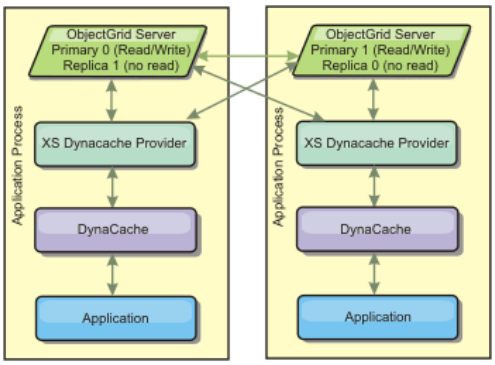
The embedded topology is similar to the default dynamic cache and DRS provider. Distributed cache instances created with the embedded topology keep a full copy of the cache within each eXtreme Scale process that accesses the dynamic cache service, allowing all read operations to occur locally. All write operations go through a single-server process, in which the transactional locks are managed, before being replicated to the rest of the servers. Consequently, this topology is better for workloads where cache-read operations greatly outnumber cache-write operations.

With the embedded topology, new or updated cache entries are not immediately visible on every single server process. A cache entry will not be visible, even to the server that generated it, until it propagates through the asynchronous replication services of WebSphere eXtreme Scale . These services operate as fast as the hardware will allow, but there is still a small delay. The embedded topology is shown in the following image:



Embedded partitioned topology

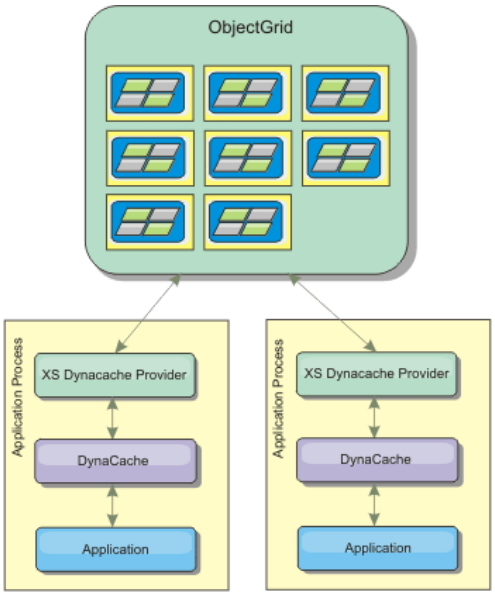
For workloads where cache-writes occur as often as or more frequently than reads, the embedded partitioned or remote topologies are recommended. The embedded partitioned topology keeps all of the cache data within the WebSphere Application Server processes that access the cache. However, each process only stores a portion of the cache data. All reads and writes for the data located on this “partition” go through the process, meaning that most requests to the cache will be fulfilled with a remote procedure call. This results in a higher latency for read operations than the embedded topology, but the capacity of the distributed cache to handle read and write operations will scale linearly with the number of WebSphere Application Server processes accessing the cache. Also, with this topology, the maximum size of the cache is not bound by the size of a single WebSphere process. Because each process only holds a portion of the cache, the maximum cache size becomes the aggregate size of all the processes, minus the overhead of the process. The embedded partitioned topology is shown in the following image:



For example, assume you have a grid of server processes with 256 megabytes of free heap each to host a dynamic cache service. The default dynamic cache provider and the eXtreme Scale provider using the embedded topology would both be limited to an in-memory cache size of 256 megabytes minus overhead. See the Capacity Planning and High Availability section later in this document. The eXtreme Scale provider using the embedded partitioned topology would be limited to a cache size of one gigabyte minus overhead. In this manner, the WebSphere eXtreme Scale provider makes it possible to have an in-memory dynamic cache services that are larger than the size of a single server process. The default dynamic cache provider relies on the use of a disk cache to allow cache instances to grow beyond the size of a single process. In many situations, the WebSphere eXtreme Scale provider can eliminate the need for a disk cache and the expensive disk storage systems needed to make them perform.

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:



Data compression

Another performance feature offered by the WebSphere eXtreme Scale dynamic cache provider that can help users manage cache overhead is compression. The default dynamic cache provider does not allow for compression of cached data in memory. With the eXtreme Scale provider, this becomes possible. Cache compression using the deflate algorithm can be enabled on any of the three distributed topologies. Enabling compression will increase the overhead for read and write operations, but will drastically increase cache density for applications like servlet and JSP caching.

Local in-memory cache

The WebSphere eXtreme Scale dynamic cache provider can also be used to back dynamic cache instances that have **replication disabled**. Like the default dynamic cache provider, these caches can store non-serializable data. They can also offer better performance than the default dynamic cache provider on large

multi-processor enterprise servers because the eXtreme Scale code path is designed to maximize in-memory cache concurrency.

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

Dynamic cache statistics are reported via the CacheMonitor application or the dynamic cache MBean. When using the eXtreme Scale dynamic cache provider, statistics will still be reported through these interfaces, but the context of the statistical values will be different.

If a dynamic cache instance is shared between three servers named A, B, and C, then the dynamic cache statistics object only returns statistics for the copy of the cache on the server where the call was made. If the statistics are retrieved on server A, they only reflect the activity on server A.

With eXtreme Scale , there is only a single distributed cache shared among all the servers, so it is not possible to track most statistics on a server-by-server basis like the default dynamic cache provider does. A list of the statistics reported by the Cache Statistics API and what they represent when you are using the WebSphere eXtreme Scale dynamic cache provider follows. Like the default provider, these statistics are not synchronized and therefore can vary up to 10% for concurrent workloads.

- **Cache Hits** : Cache hits are tracked per server. If traffic on Server A generates 10 cache hits and traffic on Server B generates 20 cache hits, the cache statistics will report 10 cache hits on Server A and 20 cache hits on Server B.
- **Cache Misses**: Cache misses are tracked per server just like cache hits.
- **Memory Cache Entries**: This statistic reports the number of cache entries in the distributed cache. Every server that accesses the cache will report the same value for this statistic, and that value will be the total number of cache entries in memory over all the servers.
- **Memory Cache Size in MB**: This metric is supported only for caches using the remote, embedded, or embedded_partitioned topologies. It reports the number of megabytes of Java heap space consumed by the cache, across the entire grid. This statistic reports heap usage only for the primary partitions; you must take replicas into account. Because the default setting for the remote and embedded_partitioned topologies is one asynchronous replica, double this number to get the true memory consumption of the cache.
- **Cache Removes**: This statistic reports the total number of entries removed from the cache by any method, and is an aggregate value for the whole distributed cache. If traffic on Server A generates 10 invalidations and traffic on Server B generates 20 invalidations, then the value on both servers will be 30.
- **Cache Least Recently Used (LRU) Removes**: This statistic is aggregate, like cache removes. It tracks the number of entries that were removed to keep the cache under its maximum size.
- **Timeout Invalidations**: This is also an aggregate statistic, and it tracks the number of entries that were removed because they timed out.
- **Explicit Invalidations** : Also an aggregate statistic, this tracks the number of entries that were removed with direct invalidation by key, dependency ID or template.
- **Extended Stats** : The eXtreme Scale dynamic cache provider exports the following extended stat key strings.
  - **com.ibm.websphere.xs.dynacache.remote_hits**: The total number of cache hits tracked at the eXtreme Scale container. This is an aggregate statistic, and its value in the extended stats map is a `long`.
  - **com.ibm.websphere.xs.dynacache.remote_misses**: The total number of cache misses tracked at the eXtreme Scale container. An aggregate statistic, its value in the extended stats map is a `long`.

Reporting reset statistics

The dynamic cache provider allows you to reset cache statistics. With the default provider the reset operation only clears the statistics on the affected server. The eXtreme Scale dynamic cache provider tracks most of its statistical data on the remote cache containers. This data is not cleared or changed when the statistics are reset. Instead the default dynamic cache behavior is simulated on the client by reporting the difference between the current value of a given statistic and the value of that statistic the last time reset was called on that server.

For example, if traffic on Server A generates 10 cache removes, the statistics on Server A and on Server B will report 10 removes. Now, if the statistics on Server B are reset and traffic on Server A generates an additional 10 removes, the statistics on Server A will report 20 removes and the stats on Server B will report 10 removes.

Dynamic cache events

The dynamic cache API allows users to register event listeners. When you are using eXtreme Scale as the dynamic cache provider, the event listeners work as expected for local in-memory caches.

For distributed caches, event behavior will depend on the topology being used. For caches using the embedded topology, events will be generated on the server that handles the write operations, also known as the primary shard. This means that only one server will receive event notifications, but it will have all the event notifications normally expected from the dynamic cache provider. Because WebSphere eXtreme Scale chooses the primary shard at runtime, it is not possible to ensure that a particular server process always receives these events.

Embedded partitioned caches generate events on any server that hosts a partition of the cache. For example, if a WebSphere Application Server Network Deployment environment has 11 application servers that host 11 partitions for a cache, then each server receives the dynamic cache events for the cache entries that it hosts. No single server process would see all of the events unless all 11 partitions were hosted in that server process. As with the embedded topology, it is not possible to ensure that a particular server process receives a particular set of events or any events at all.

Caches that use the remote topology do not support dynamic cache events.

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 WebSphere Application Server built-in dynamic cache provider supports multiple cache replication policies. These policies can be configured globally or on each cache entry. See the dynamic cache documentation for a description of these replication policies.

The eXtreme Scale dynamic cache provider does not honor these policies directly. The replication characteristics of a cache are determined by the configured eXtreme Scale distributed topology type and apply to all values placed in that cache, regardless of the replication policy set on the entry by the dynamic cache service. The following is a list of all the replication policies supported by the dynamic cache service and illustrates which eXtreme Scale topology provides similar replication characteristics.

Note that the eXtreme Scale dynamic cache provider ignores DRS replication policy settings on a cache or cache entry. Users must choose the topology that appropriate to their replication needs.

- NOT_SHARED – currently none of the topologies provided by the eXtreme Scale dynamic cache provider can approximate this policy. This means that all data stored into the cache must have keys and values that implement java.io.Serializable.
- SHARED_PUSH – The embedded topology approximates this replication policy. When a cache entry is created it is replicated to all the servers. Servers only look for cache entries locally. If an entry is not found locally, it is assumed to be non-existent and the other servers are not queried for it.
- SHARED_PULL and SHARED_PUSH_PULL – The embedded partitioned and remote topologies approximate this replication policy. The distributed state of the cache is completely consistent between all the servers.

This information is provided mainly so you can make sure that the topology meets your distributed consistency needs. For example, if the embedded topology is a better choice for your deployment and performance needs, but you require the level of cache consistency provided by SHARED_PUSH_PULL, then consider using embedded partitioned, even though the performance may be slightly lower.

Additional information

- Dynamic cache Redbook
- Dynamic cache documentation
  - WebSphere Application Server 7.0
  - WebSphere Application Server 6.1
- DRS documentation
  - WebSphere Application Server 7.0
  - WebSphere Application Server 6.1

Related concepts:
Dynamic cache capacity planning
Related tasks:
Configuring the dynamic cache provider for WebSphere eXtreme Scale

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 1. ObjectGrid as a database buffer

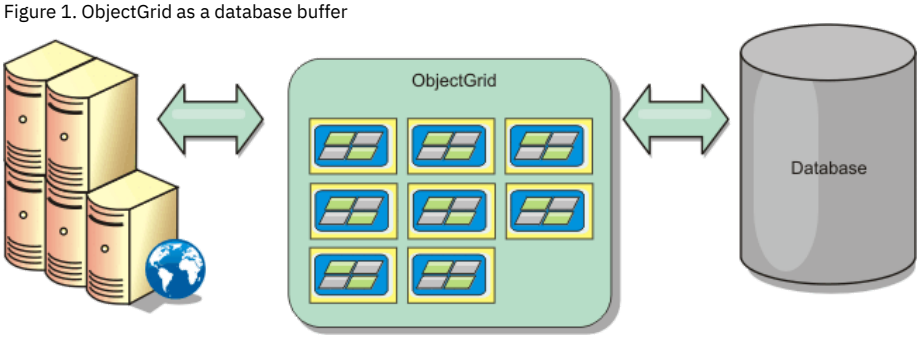
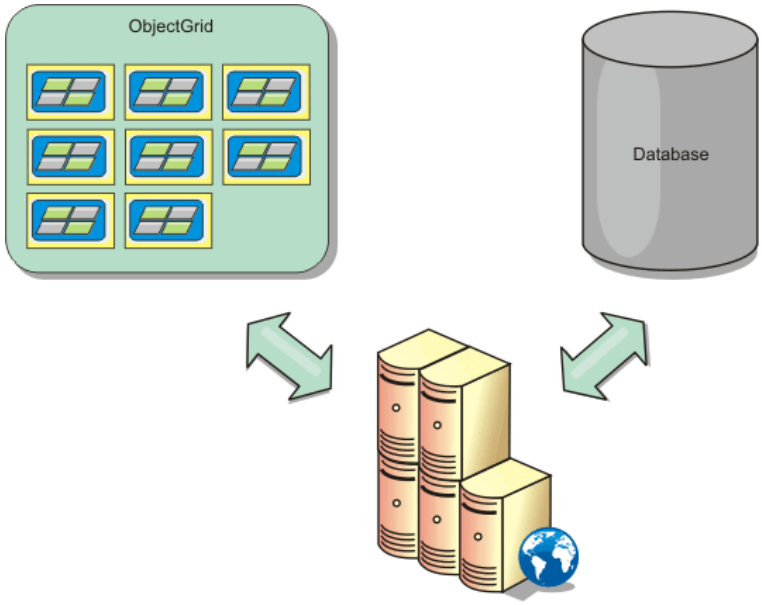


Figure 2. ObjectGrid as a side cache



- 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.
- Side cache
  When WebSphere eXtreme Scale is used as a side cache, the back end is used with the data grid.
- 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.
- Write-behind caching
  You can use write-behind caching to reduce the overhead that occurs when updating a database you are using as a back end.
- Loaders
  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.
- Data preloading and warm-up
  In many scenarios that incorporate the use of a loader, you can prepare your data grid by preloading it with data.
- 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.
- Data invalidation
  To remove stale cache data, you can use invalidation mechanisms.
- Indexing
  Use the MapIndexPlugin plug-in to build an index or several indexes on a BackingMap to support non-key data access.
- JPA Loaders
  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.

Related concepts:
Local in-memory cache
Peer-replicated local cache
Embedded cache
Distributed cache
Planning multiple data center topologies
Loader considerations in a multi-master topology
Programming for JPA integration
Plug-ins for communicating with databases

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

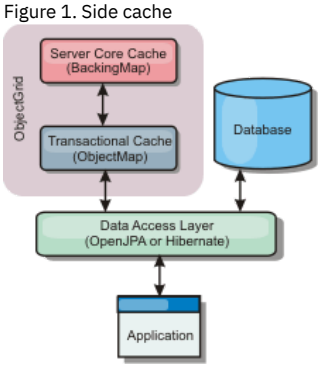
When WebSphere® eXtreme Scale is used as a side cache, the back end is used with the data grid.

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 1. Side cache



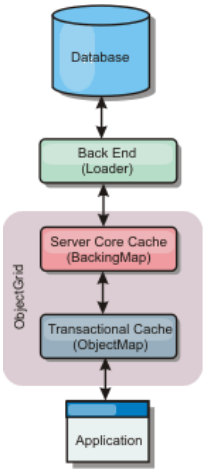
Cache plug-ins for both OpenJPA and Hibernate are included in WebSphere eXtreme Scale, so you can use the product as an automatic side-cache. Using WebSphere eXtreme Scale as a cache provider increases performance when reading and querying data and reduces load to the database. There are advantages that 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 1. In-line cache



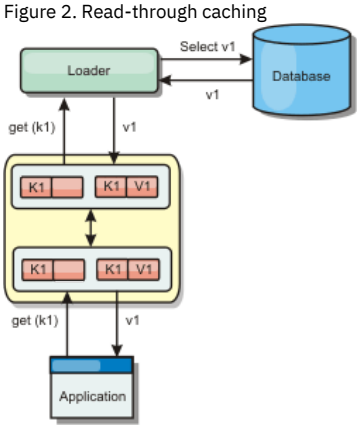
The in-line caching option simplifies data access because it allows applications to access the eXtreme Scale APIs directly. WebSphere eXtreme Scale supports several in-line caching scenarios, as follows.

- 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.

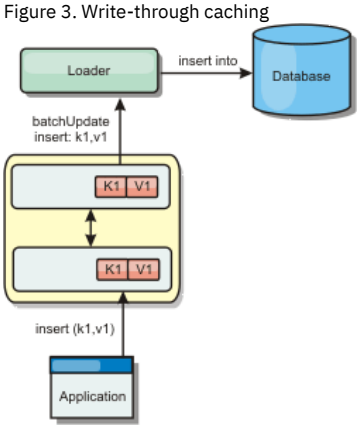
Figure 2. Read-through caching



Write-through caching scenario

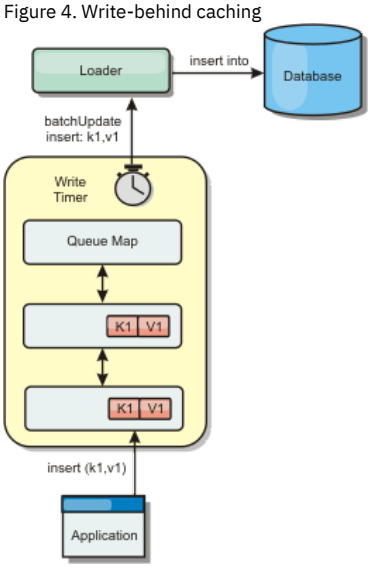
In a write-through cache, every write to the cache synchronously writes to the database using the Loader. This method provides consistency with the back end, but decreases write performance since the database operation is synchronous. Since the cache and database are both updated, subsequent reads for 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 3. 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 4. Write-behind caching



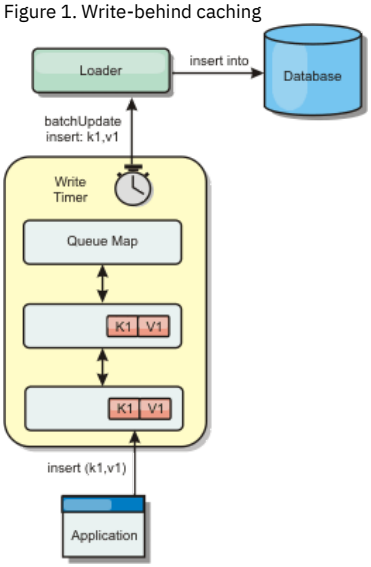
Write-behind caching

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 1. 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.

Related concepts:
Write-behind loader application design considerations
Handling failed write-behind updates
Related reference:
Example: Writing a write-behind dumper class
Example: Writing a write-behind dumper class

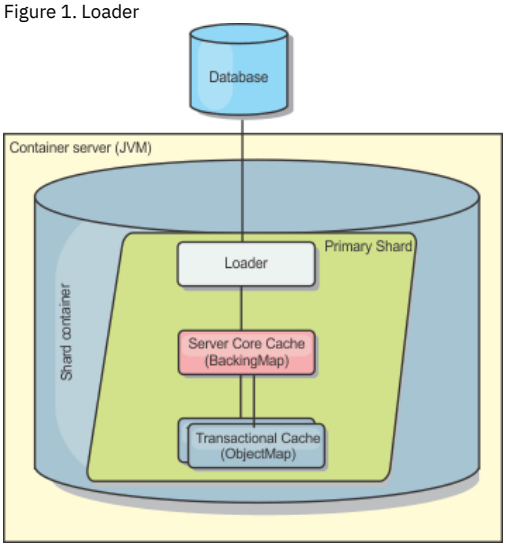
Loaders

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. The Loader logic provides a read-through capability for the cache, which means that data is populated into the cache on demand. Since the entire data set does not need to be loaded upon startup, the cache can be populated lazily. 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 1. Loader



In order to avoid database locking on the row that requires updating, the loader can also perform optimistic transaction locking. In this scenario, no locking is required on a row. The update is overqualified to ensure that only rows that are in the same state as those originally read are changed. By storing a version attribute in the cache value, the loader can also 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 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.

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.

Related concepts:
Plug-ins for communicating with databases
Writing a loader
JPAEntityLoader plug-in
Using a loader with entity maps and tuples
Writing a loader with a replica preload controller
Related tasks:
Monitoring eXtreme Scale information in DB2
Related reference:
JPA loader programming considerations

Data preloading and warm-up

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

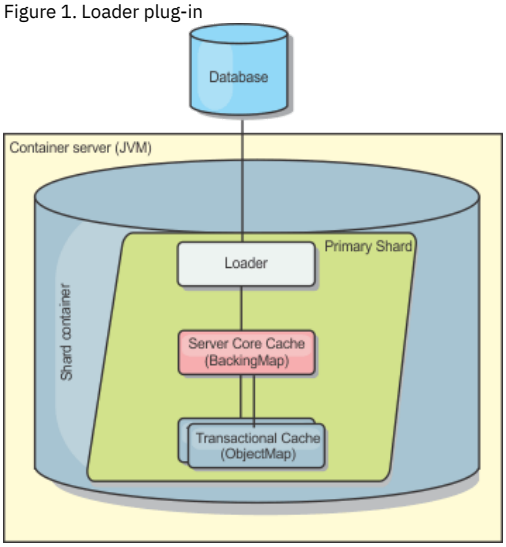
When used as a complete cache, the data grid must hold all of the data and must be loaded before any clients can connect to it. When you are using a sparse cache, you can warm up the cache with data so that clients can have immediate access to data when they connect.

Two approaches exist for preloading data into the data grid: Loader plug-in or client loader.

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 runs 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 completes.

Figure 1. Loader plug-in

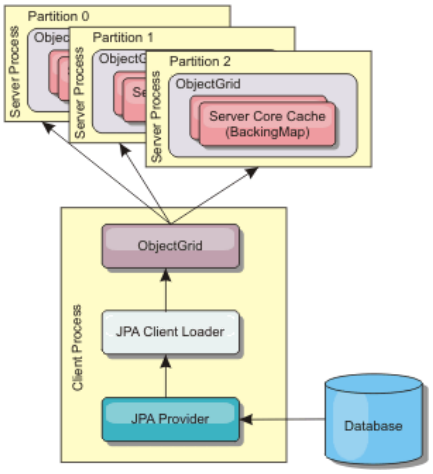


For more information about the Loader plug-in, see Plug-ins for communicating with databases.

Client loader

A client loader is a pattern for using one or more clients to load the data 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 preload mode, so that clients are not able to access the data grid while it is preloading 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.

Figure 2. Client loader



Database synchronization techniques

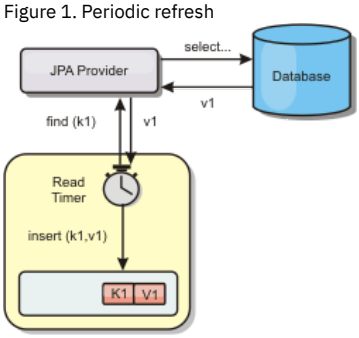
When WebSphere® eXtreme Scale is used as a cache, applications must be written to tolerate stale data if the database can be updated independently from an eXtreme Scale transaction. To serve as a synchronized in-memory database processing space, eXtreme Scale provides several ways of keeping the cache 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 1. 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. See Plug-ins for evicting cache objects for further details.

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.

8.5+

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.

Related concepts:
JMS event listener
Related tasks:
8.5+ Querying and invalidating data
Configuring the dynamic cache provider for WebSphere eXtreme Scale
Related reference:
ObjectGridEventListener plug-in
Introduction to ObjectMap
Related information:
ObjectMap.invalidate method
EntityManager.invalidate method
ObjectGridEventListener interface

Indexing

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.

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, 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.

The HashIndex class is the built-in index plug-in implementation that can support both of the built-in application index interfaces: MapIndex and MapRangeIndex. You also can create your own indexes. You can add HashIndex as either a static or dynamic index into the BackingMap, obtain either MapIndex or MapRangeIndex index proxy object, and use the index proxy object to find cached objects.

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.

Related concepts:
Plug-ins for indexing data
Plug-ins for custom indexing of cache objects
Using a composite index
Tuning query performance
Related tasks:
Configuring the HashIndex plug-in
Accessing data with indexes (Index API)
Related reference:
HashIndex plug-in attributes

JPA Loaders

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.

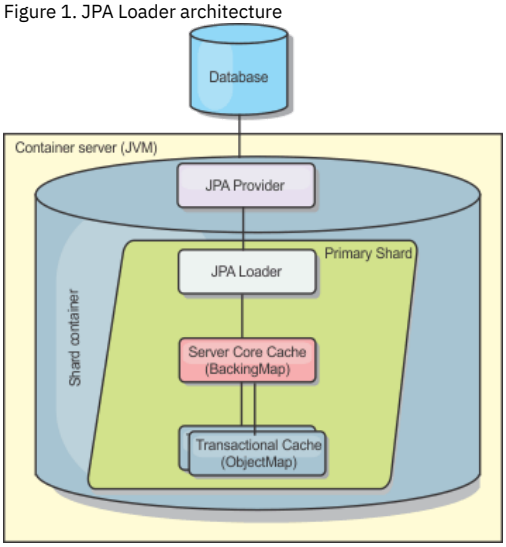
The JPALoader com.ibm.websphere.objectgrid.jpa.JPALoader and the JPAEntityLoader com.ibm.websphere.objectgrid.jpa.JPAEntityLoader plug-ins are two built-in JPA loader plug-ins that are used to synchronize the ObjectGrid maps with a database. You must have a JPA implementation, such as Hibernate or OpenJPA, 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 1. 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 details, see JPAEntityLoader plug-in.For more information, see JPAEntityLoader plug-in.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. For more information about configuring JPA loaders, see Configuring JPA loaders. For more information about programming JPA loaders, see JPA loader programming considerations.
Related tasks:
Developing client-based JPA loaders
Starting the JPA time-based updater
Related reference:
Example: Using the Hibernate plug-in to preload data into the ObjectGrid cache

Serialization overview

Data is always expressed, but not necessarily stored, as Java™ objects in the data grid. WebSphere® eXtreme Scale uses multiple Java processes to serialize the data, by converting the Java object instances to bytes and back to objects again, as needed, to move the data between client and server processes.

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.

Serialization for Java applications

To serialize data, you can use 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 1. Serialization methods that are available when you are running logic that interacts with data objects directly in the data grid shard

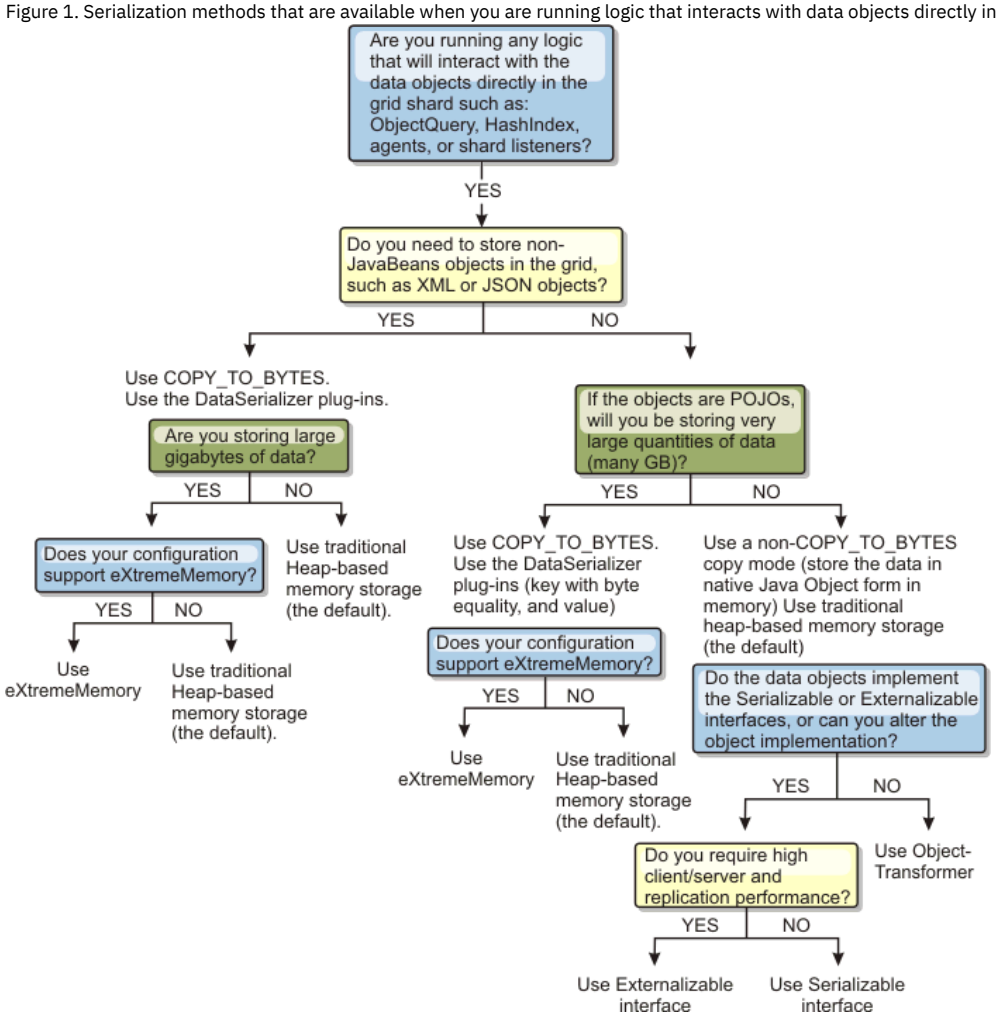
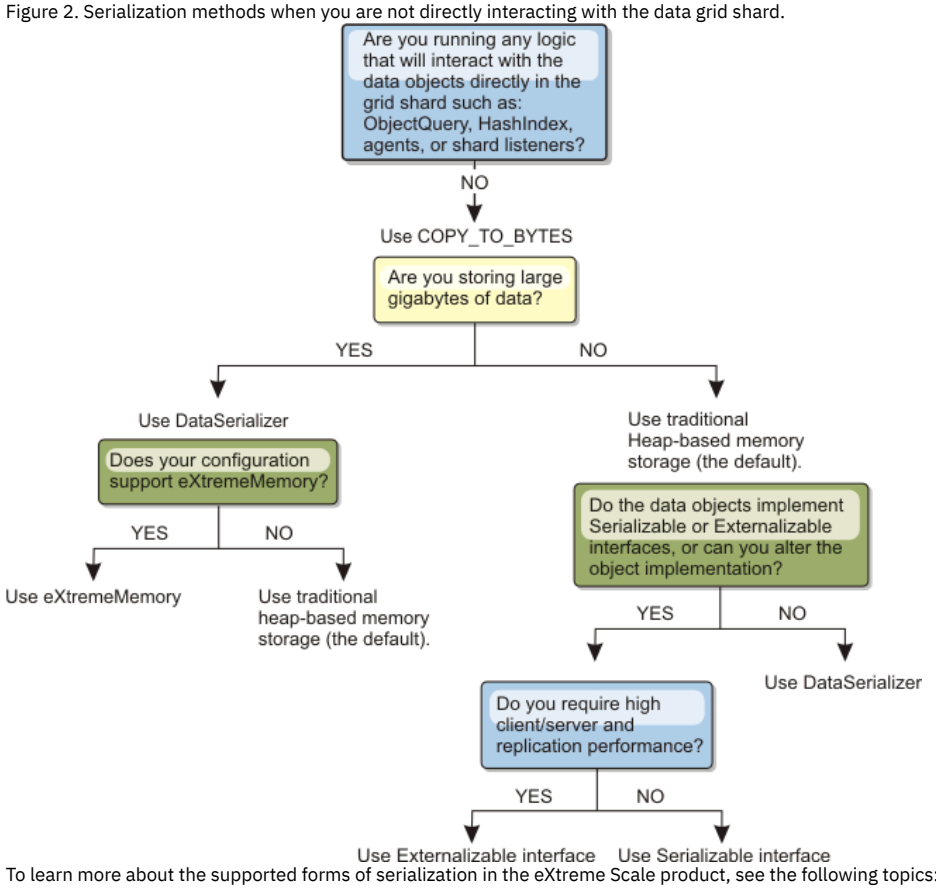


Figure 2. Serialization methods when you are not directly interacting with the data grid shard.



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

Related concepts:
Serialization using Java
Serialization using the DataSerializer plug-ins
ObjectTransformer plug-in
Samples
Java plug-ins overview
Plug-ins for serializing cached objects
Serializer programming overview
IBM eXtremeMemory
Serializer programming overview
Samples
Related tasks:
Avoiding object inflation when updating and retrieving cache data
Planning to use IBM eXtremeMemory
Avoiding object inflation when updating and retrieving cache data
Programming to use the OSGi framework
Related information:
Oracle Java Serialization API
DataSerializer API documentation

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.

Related concepts:
Serialization overview
Serialization using the DataSerializer plug-ins
ObjectTransformer plug-in
Samples
Java plug-ins overview
Plug-ins for serializing cached objects
Serializer programming overview
IBM eXtremeMemory
Related tasks:
Avoiding object inflation when updating and retrieving cache data
Planning to use IBM eXtremeMemory
Related information:
Oracle Java Serialization API

ObjectTransformer plug-in

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="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxsplugobjtrans_myMap">
            <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxsplugobjtrans_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);
```

Related concepts:
Serialization using Java
Serialization overview
Serialization using the DataSerializer plug-ins
Samples
Java plug-ins overview
Plug-ins for serializing cached objects
Serializer programming overview
IBM eXtremeMemory
Tuning serialization performance
Tuning serialization
Using a loader with entity maps and tuples
Tuning copy operations with the ObjectTransformer interface
Related tasks:
Avoiding object inflation when updating and retrieving cache data
Planning to use IBM eXtremeMemory
Related information:
Oracle Java Serialization API

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 API documentation.
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.
Related concepts:
Serialization using Java
Serialization overview
ObjectTransformer plug-in
Samples
Java plug-ins overview
Plug-ins for serializing cached objects
Serializer programming overview
IBM eXtremeMemory
Tuning serialization performance
Tuning serialization
Using a loader with entity maps and tuples
Tuning copy operations with the ObjectTransformer interface
Related tasks:
Avoiding object inflation when updating and retrieving cache data
Planning to use IBM eXtremeMemory
Related information:
Oracle Java Serialization API

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

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 `(Number_Partitions*(1 + 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 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 and handle this overhead based on the number Java virtual machines that 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. Therefore, if you expect to scale to 20 computers, then 20 shards per container Java virtual machines are required (when 240 shards are divided by 12 container JVMs) for the initial deployment.

ObjectMap and partitioning

When you use the fixed partition placement strategy using the default value FIXED_PARTITIONS, 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.

Related concepts:
Replication for availability
Writing a loader with a replica preload controller

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_PARTITIONS` setting. The number of primary shards that are placed across the available containers is equal to the number of partitions that you configured with the numberOfPartitions attribute. If you configured replicas, the minimum total number of shards that are placed is defined by the following formula: `((1 primary shard + minimum synchronous shards) * partitions defined)`. The maximum total number of shards that are placed is defined by the following formula: `((1 primary shard + maximum synchronous shards + maximum asynchronous shards) * partitions)`. Your

WebSphere eXtreme Scale deployment spreads these shards over the available containers. The keys of each map are hashed into assigned partitions that are based on the total partitions you 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 six partitions requires a synchronous replica. If three containers are started, WebSphere eXtreme Scale places four shards per container for MapSet1.

Per-container placement

The per-container placement strategy is enabled when you set the placementStrategy attribute to `PER_CONTAINER` in the map set element in the deployment XML file. With this strategy, the number of primary shards that are placed on each new container is equal to the number of partitions, *P*, that you 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 SessionHandle for routing.

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 placement, 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

Data grids that are configured to use a per-container placement strategy are different. You can specify that the data grid uses the per-container placement strategy by setting the placementStrategy attribute in your deployment XML file to PER_CONTAINER. 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 five primaries (P0 - P4).
    - C0 hosts: P0, P1, P2, P3, P4.
2. Start container C1 hosting five 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 five 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 the per-container placement strategy is used, only replicas.
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 work 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 five containers rather than only one.

A 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 five local partition primaries for the session. The "ID" of the partition that is 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 five again). With each new container started, you have five more partition primaries and five 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 behave. When a container starts, it hosts five 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 five 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 that allows the client to interact with the same partition when necessary. 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 that is bound to the same partition with the SessionHandle.

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

Advantages

The per-container placement strategy is different from a normal fixed partition or hash data grid because the client stores data in a place in the grid. The client 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 placement strategy.

Your deployment does not make a new partition for each Session. So in a per-container deployment, the keys that are used to store data in the partition must be unique within that partition. For example, you can 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 partition.

The previous examples used five partitions, but the numberOfPartitions parameter in the object grid 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 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 to serverproperties or clientproperties.

The per-container strategy for a data grid suits applications that store 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.

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.

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.

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.

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.
- 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.

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 is at the root and core groups hold container servers are at the leaves of the tree. See Caching architecture: Maps, containers, clients, and catalogs for more information.

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.
- **Network partition**: Two catalog servers act as primaries concurrently. Both servers make changes to the catalog server state, which leads to data corruption.

- Core groups
  A core group is a high availability domain of container servers. 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 might be a Java virtual machine (JVM) failure or a newly added JVM that joins the core group.
- 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.
- Catalog server quorums
  The catalog service domain is a fixed set of catalog server Java virtual machines (JVM). For the best performance, do not configure catalog service domains to span data centers. When the quorum mechanism is enabled, all the catalog servers in the quorum must be available and communicating with each other for placement operations to occur in the data grid. The catalog service responds to container server lifecycle events while the catalog service has quorum. These lifecycle events include the placement or removal of shards on a container server when the container server stops or starts. When a brownout scenario or other failure occurs, not all members of the quorum are available. So, you must override quorum because placement operations do not occur if the quorum is not 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.

Related tasks:
Configuring the quorum mechanism
Tuning the heartbeat interval setting for failover detection
Managing data center failures when quorum is not enabled

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 server s 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 data grid. Replication has a programming model impact only when preloading the backing maps.

Map preloading

For a description of preloading methods, including client loaders, see Data preloading and warm-up.

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 a data grid 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 data grid. 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 1. 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 runtime 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 runtime 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 2. 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.

```
CommitedTx = {}, RolledBackTx = {}
```

During synchronous commit processing, use the following sequence:

Table 3. 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.

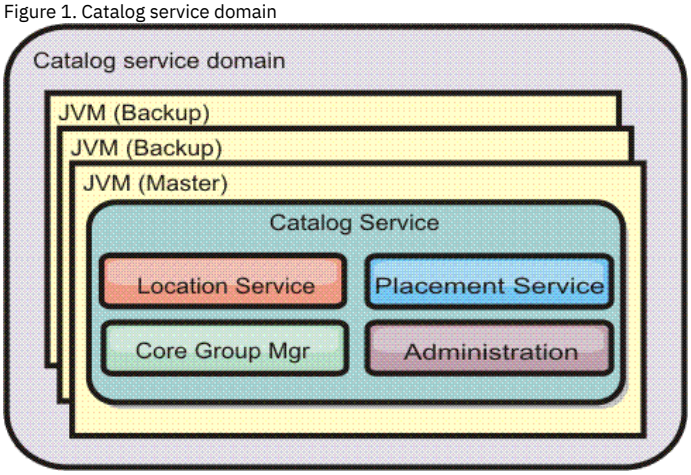
Related concepts:
Partitioning
Writing a loader with a replica preload controller
Related tasks:
Configuring the quorum mechanism
Tuning the heartbeat interval setting for failover detection
Managing data center failures when quorum is not enabled

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.

Figure 1. 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.

When clients contact any one of the catalog servers, the routing table for the catalog service domain is propagated to the clients through the Common Object Request Broker Architecture (CORBA) service context.

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 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.

Related concepts:
Catalog service
Related tasks:
Configuring the quorum mechanism
Tuning the heartbeat interval setting for failover detection
Managing data center failures when quorum is not enabled
Configuring WebSphere eXtreme Scale with WebSphere Application Server
Configuring the catalog service in WebSphere Application Server
Creating catalog service domains in WebSphere Application Server
Related reference:
Catalog service domain administrative tasks
Server properties file

Catalog server quorums

The catalog service domain is a fixed set of catalog server Java virtual machines (JVM). For the best performance, do not configure catalog service domains to span data centers. When the quorum mechanism is enabled, all the catalog servers in the quorum must be available and communicating with each other for placement operations to occur in the data grid. The catalog service responds to container server lifecycle events while the catalog service has quorum. These lifecycle events include the placement or removal of shards on a container server when the container server stops or starts. When a brownout scenario or other failure occurs, not all members of the quorum are available. So, you must override quorum because placement operations do not occur if the quorum is not available.

Failure classification

Single failure: When the failure of one container server or catalog server occurs in the environment, it is considered to be a single failure event. When a single failure event occurs, recovery can occur without data loss.

Double failure: When two failures of any server processes occur simultaneously, data loss can occur on the second failure. Because of the second failure, applications might lose write access to the data that was stored on the failed container server. To prevent double failures, you can isolate components of the data grid from each other. For more information, see Zones.

Quorum loss

If the catalog service loses quorum, it waits for quorum to be reestablished. While the catalog service does not have quorum, it ignores lifecycle events from catalog and container servers.

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

- **A catalog server fails**
  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.
- **Brownout occurs**
  A brownout is when a temporary loss of connectivity occurs. Brownouts are transient and clear within seconds or minutes. Brownouts can be frequent and repeated depending on the cause. Brownouts can be caused by network partitions, long garbage collection pauses, operating system level swapping, or disk I/O problems. Quorum is the mechanism for reacting to brownouts in the catalog server that are long enough to cause heartbeat failures. While the product 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.

WebSphere eXtreme Scale does not lose quorum when a catalog server is stopped with the stop command or any other administrative actions. The system knows that the server instance stopped, which is different from a JVM failure or brownout. The quorum drops to one less server, preserving quorum. The remaining servers still have quorum. Restarting the catalog server sets quorum back to the previous number.

Client behavior during quorum loss

If the client can connect to a catalog server, the client can 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. If no container failures or connectivity issues happen during the quorum loss event, then clients can still fully interact with the container servers.

Recovery after quorum is reestablished

If quorum is lost for any reason, when quorum is reestablished, a recovery protocol is run. When the quorum loss event occurs, all heartbeating for core groups is suspended and failure reports are also ignored. After quorum is back, any container server failures that occurred while quorum was lost are processed. Any shards that were hosted on container servers that were reported as failed are recovered. If primary shards were lost, then surviving replica shards become primary shards. If replica shards were lost, more replica shards are created.

Scenarios for overriding quorum

Quorum loss due to a catalog server failure or a network brownout recovers automatically after the catalog server is restarted or the network brownout ends. When intermittent failures are occurring, such as network instability, you must remove the problematic catalog servers by manually ending the catalog server processes. Then, you can override quorum.

When you override quorum, the catalog service assumes that quorum is achieved with the current membership. Container server lifecycle events are processed. When you run an override quorum command, you are informing the catalog service domain that the failed catalog servers do not have a chance of recovering.

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

- **Brownout:** Brownout scenarios occur and are resolved fairly quickly. 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. You do not need to take any administrative actions.
- **JVM process failure:** The JVM for the C catalog server fails and the catalog service loses quorum. You can override quorum immediately, which restarts the processing of container server lifecycle events. Then, diagnose why the C catalog server failed and resolve any issues. When you are sure that the problem is resolved, 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.
- **Problematic or repeated brownouts:** In this scenario, the A and B catalog servers are on one side of the network partition, while the C catalog server is on the other. You must be careful about when you override quorum in this scenario. You do not want to override quorum just as the brownout temporarily heals, and then have the brownout occur again. If this scenario were to occur, both sides of the network partition could become primary, causing a split brain condition.
- **Multiple failures:** During a failure scenario, catalog server C and one or more container servers are lost. Ensure that the failing servers are stopped. Then, override quorum. The surviving catalog servers use the remaining container servers to run a full recovery by replacing shards that were hosted in the failed container servers. 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. If multiple containers were lost that included primary and all replica shards for particular partitions, data loss for those partitions occurs.

Majority quorum

For added flexibility to the standard quorum support in WebSphere eXtreme Scale, a new quorum type is available called majority quorum. In this quorum type, WebSphere eXtreme Scale does not leave quorum if there are greater than half the catalog servers still running. For example, if there are three catalog servers and one of them cannot communicate with the other two catalog servers, then the other two catalogs stay in quorum. The other two catalogs allow for placement changes to occur. If the other catalog rejoins the group, WebSphere eXtreme Scale tries to let it join dynamically if possible. Otherwise, the catalog is restarted so that it can properly rejoin the catalog cluster. Majority quorum automatically resolves catalog server failures on the majority side when a brownout event affects the catalog servers. Also, this quorum policy greatly reduces the need to recycle the catalogs that were partitioned when they rejoin. Even if the primary catalog server was partitioned, when it rejoins the cluster, the catalog server is merged back and only one primary remains in the cluster. To enable majority quorum, see Configuring the quorum mechanism. However, if you have four catalog servers and two are isolated, then there is no majority and WebSphere eXtreme Scale leaves quorum. Therefore, a majority quorum policy equates to`<number of catalog servers configured in a cluster>/2 +1`.

Note: If a brownout event occurs and affects container servers on the non-majority side, then the container servers need to be recycled when the brownout recovers. Also, if there are concerns of frequent and repeated brownouts within your environment, then standard quorum might prove to be a better option than majority quorum. This way, you can investigate and fix the environmental issue, rather than continually moving data around during repeated container error recovery.

Deciding on a quorum policy for your environment

Use the table to help you decide what type of quorum policy would make sense for your environment, or if you should enable quorum at all.

Quorum state	How does this policy behave during a brownout?	What is the motivation for usage?	Associated risks to consider
Standard	• All configured catalog cluster members need to be able to communicate with each other in order for placement changes to occur as a result of changes in container server lifecycle • Fully removes need to recycle catalogs.	Prevent under any circumstance the need to recycle the grid due to state corruption caused by catalogs that become divergent during periods where communication does not work.	Requires manual intervention to initiate any container failure recovery. This means failure placement will not occur until quorum is overridden. If a container is lost while quorum is violated, then the shards on that container are not available until quorum is overridden. If a client cannot reach a primary shard due to loss of the shard or network issues, it will not be able to update data. In cases where replicaReadEnabled is set to true, the client will not be able to access the data either, until quorum is overridden or reestablished.

Quorum state	How does this policy behave during a brownout?	What is the motivation for usage?	Associated risks to consider
None (No Quorum)	• When communication is broken between multiple catalog servers, each catalog can potentially promote itself as the primary. The result is a conflict on where various partitions can reside. The conflicts can also lead to inconsistent copies of customer data. • The catalog cluster state can also become corrupted in such scenarios, resulting in the need to recycle the catalogs. However, recycle can happen automatically depending on whether the container reconnect settings are enabled or disabled.	You should not manually intervene and let WebSphere eXtreme Scale recycle or shutdown catalogs automatically.	Catastrophic, repeated, insidious communication errors can lead to repeated data movement as a result of constant failure recovery. In worst cases, most of the grid must recycle to achieve consistency when environmental issues are fixed.

Related tasks:
Configuring the quorum mechanism
Tuning the heartbeat interval setting for failover detection
Managing data center failures when quorum is not enabled
Managing data center failures
Managing data center failures when quorum is enabled
Administering with the xscmd utility

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:

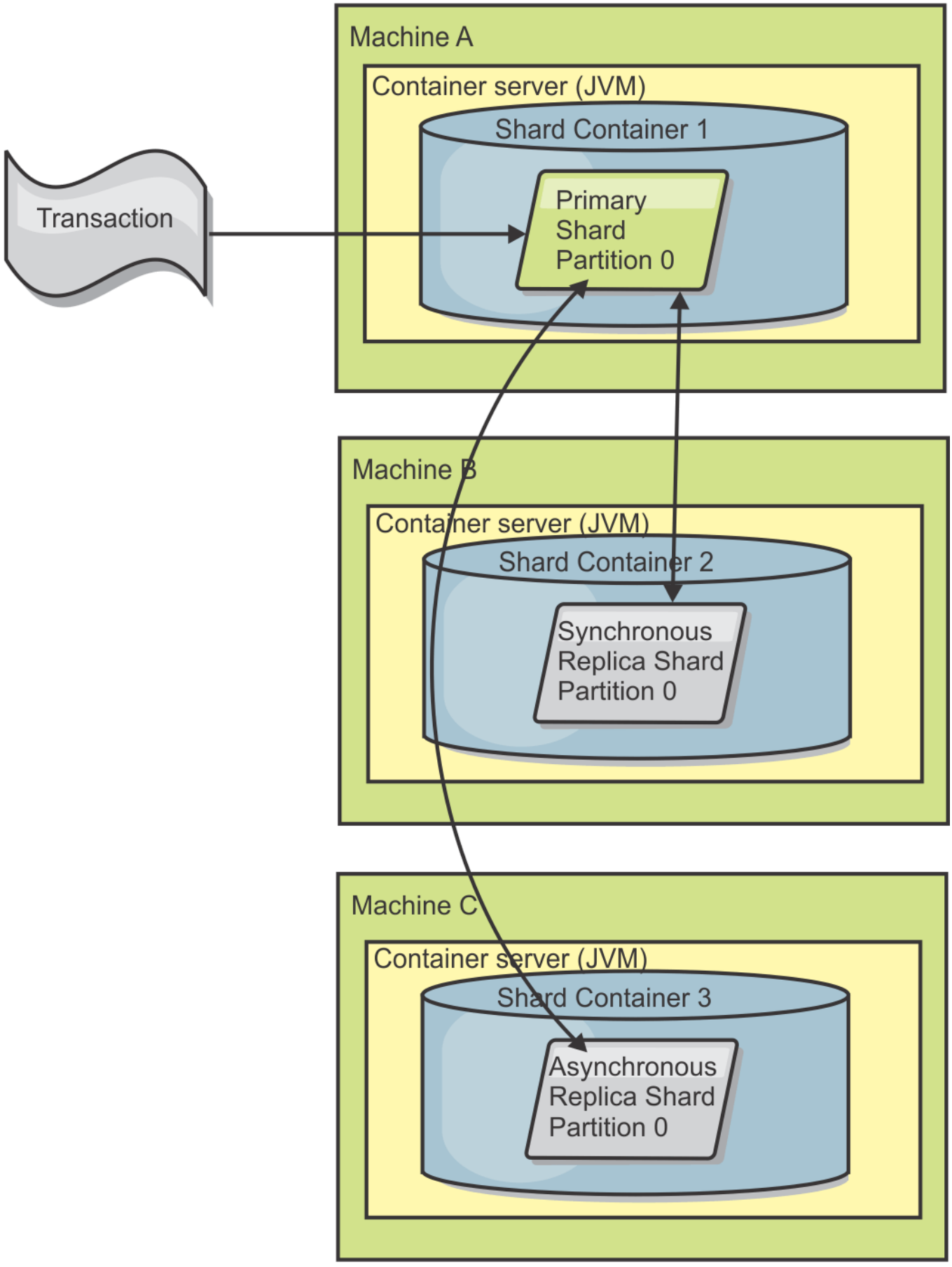
- Primary
- Synchronous replica
- Asynchronous replica

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 1. 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 synch. 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.

Replica behavior during failures

Synchronous replica behavior

The primary shard can accept new transactions during brownout or blackout conditions if the number of replicas online is at least at the minsync 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 resynchronized with the current state of the primary when the link is reestablished.

Attention: 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 deletes the buffered list and when the replica reattaches then it is cleared and resynchronized.

- 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.
- 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.
- 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.
- 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.
- Map sets for replication
  Replication is enabled by associating BackingMaps with a map set.

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.

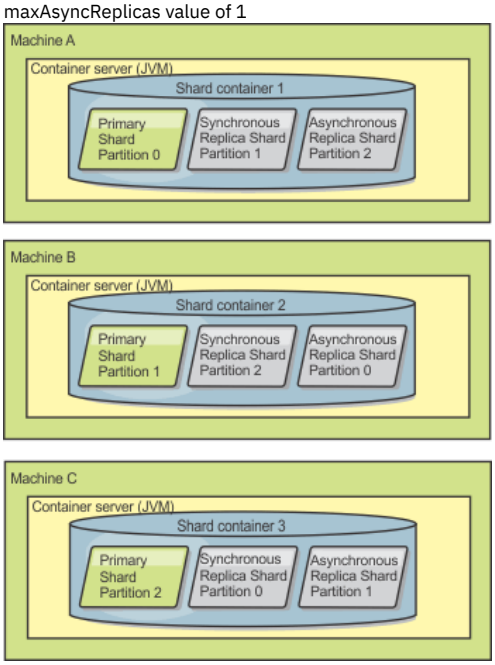
Scaling out

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 1. 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 Deployment policy descriptor XML file.

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 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.

```
CWOBJ1526I: 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 1. 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.

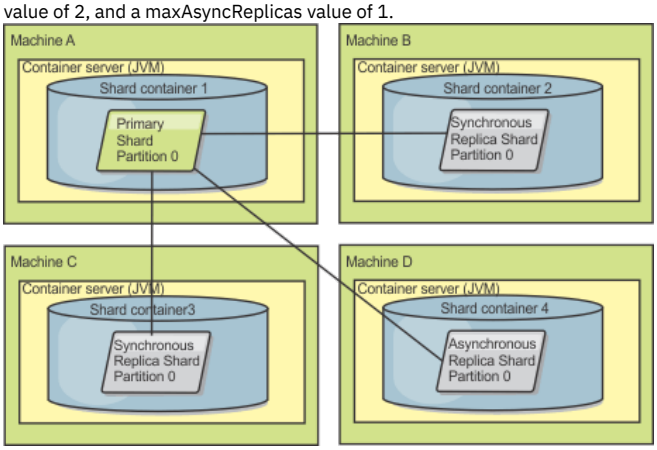


Figure 2. The container for the primary shard fails

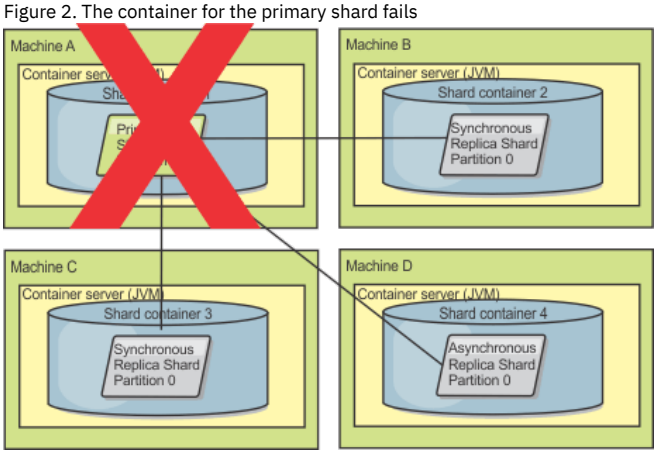


Figure 3. The synchronous replica shard on ObjectGrid container 2 becomes the primary shard

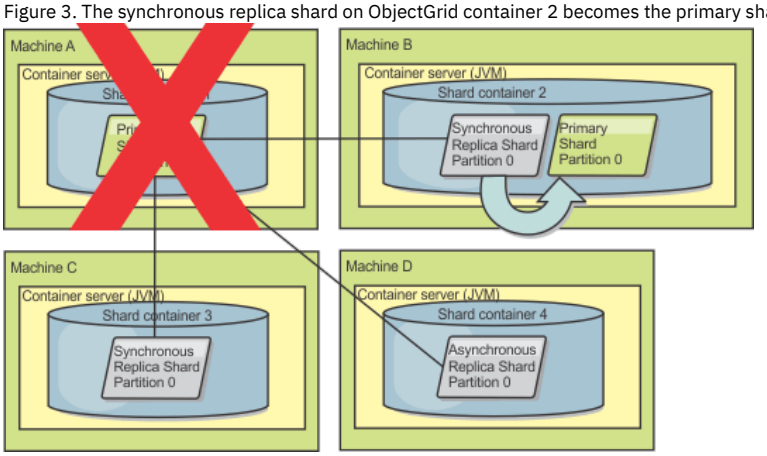
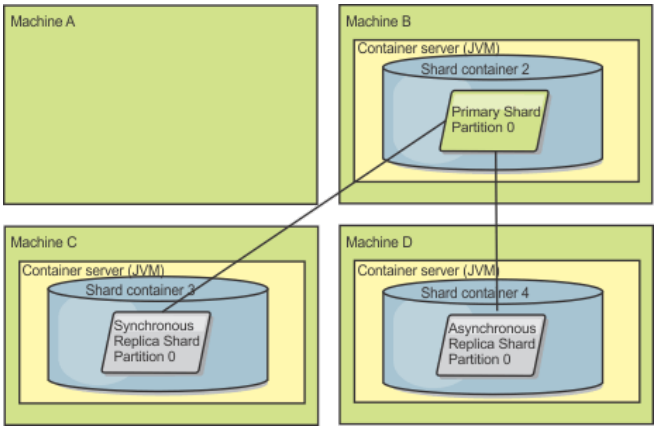


Figure 4. Machine B contains the primary shard. Depending on how automatic repair mode is set and the availability of the containers, a new synchronous replica shard might or might not be placed on a machine.



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.

```
CWOBJ1524I: 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.

```
CWOBJ1537E: 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:

```
CWOBJ1527W 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 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.

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.

- 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.
- 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.
- 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.
- 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.
- Deadlocks
  Deadlocks can occur when two transactions try to update the same cache entry.
- 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.
- 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.
- 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.
- JMS for distributed transaction changes
  Use Java™ Message Service (JMS) for distributed transaction changes between different tiers or in environments on mixed platforms.

Related tasks:
Resolving lock timeout exceptions

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. 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.

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.

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.

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

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.

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 WebSphereTranscationCallback plug-in.
8.5+

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.

Transaction processing in Java EE applications

8.5+ 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.

Next topic: Installing an eXtreme Scale resource adapter
Related information:
Unshareable and shareable connections
Connection handles
Transaction type and connection behavior
Transaction support in WebSphere Application Server
Global transactions
Local transaction containment
Local and global transactions

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.

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.

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"/>
```

Related concepts:
Tuning the copy mode
Improving performance with byte array maps
Related reference:
ObjectGrid descriptor XML file

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:

No locking
: Running without the locking setting is the fastest. If you are using read-only data, then you might not need locking.

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.

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.

Optimistic no versioning locking
: This locking strategy allows you to disable version control. This is important because near cache is only enabled if you are doing Optimistic locking. With the current implementation, you need a plug-in or callback handler to handle version control. However, using the OPTIMISTIC_NO_VERSIONING locking strategy to disable version control on the client and only enable it on the server, is an additional performance savings.

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.

8.5 To avoid a java.lang.IllegalStateException exception, you must call the setNumberOfLockBuckets method before the initialize or getSession methods on the ObjectGrid instance. The setNumberOfLockBuckets method parameter is a Java primitive integer that specifies the number of lock buckets to use. Using a prime number can allow for a uniform distribution of map entries over the lock buckets. A good starting point for best performance is to set the number of lock buckets to about 10 percent of the expected number of BackingMap entries.

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.
- 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.

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.
- 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.

Optimistic no versioning

You can enable OPTIMISTIC_NO_VERSIONING locking either through the client override XML file or programmatically. See the following examples of both approaches:
Client override XML file example

```
<objectGrid name="lockStrategyGrid">
            <backingMap name="opt_with_noversion" lockStrategy="OPTIMISTIC_NO_VERSIONING"/>
            <backingMap name="opt_with_none" lockStrategy="NONE"/>
            <backingMap name="optnoversion_with_opt" lockStrategy="OPTIMISTIC"/>
            <backingMap name="optnoversion_with_none" lockStrategy="NONE"/>
        </objectGrid>
```

Programmatic example

```
ObjectGridConfiguration lsConfig = ObjectGridConfigFactory.createObjectGridConfiguration("lockStrategyGrid");
                BackingMapConfiguration oMapWithOVConfig =
ObjectGridConfigFactory.createBackingMapConfiguration("opt_with_noversion");
                oMapWithOVConfig.setLockStrategy(LockStrategy.OPTIMISTIC_NO_VERSIONING);
                lsConfig.addBackingMapConfiguration(oMapWithOVConfig);
```

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.
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.

Related tasks:
Configuring a locking strategy in the ObjectGrid descriptor XML file
Configuring and implementing locking in Java applications
Configuring the lock timeout value in the ObjectGrid descriptor XML file

JMS for distributed transaction changes

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 Configuring Java Message Service (JMS)-based client synchronization and Configuring peer-to-peer replication with JMS 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 Distributing changes between peer JVMs.

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.

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.

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.

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.

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.

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 configureeXtreme 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.
Related tasks:
Tutorial: Configuring Java SE security
Related information:
Introduction: Integrate WebSphere eXtreme Scale security with WebSphere Application Server using the WebSphere Application Server Authentication plug-ins
WebSphere Application Server: Securing applications and their environment

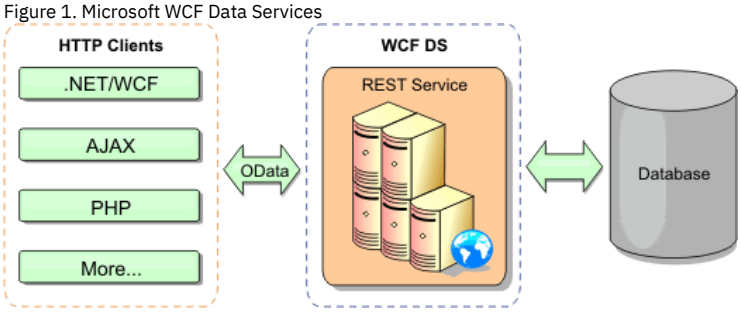
REST data services overview

The WebSphere® eXtreme Scale REST data service is a Java™ HTTP service that is compatible with Microsoft WCF Data Services (formally 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

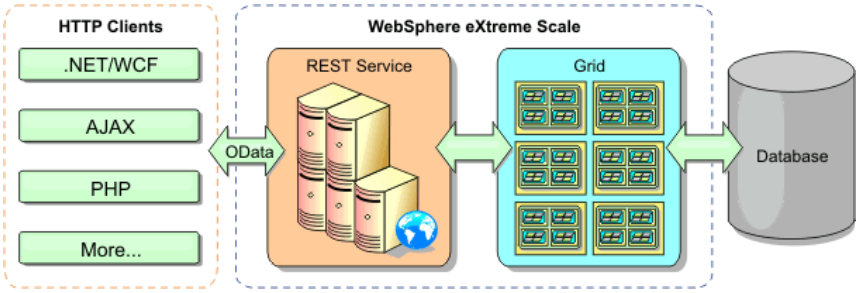
The REST data service allows any HTTP client to access a data grid. The REST data service is compatible with the WCF Data Services support supplied with the Microsoft .NET Framework 3.5 SP1. RESTful applications can be developed with the rich tooling provided by Microsoft Visual Studio 2008 SP1. The figure provides an overview of how WCF Data Services interacts with clients and databases.

Figure 1. 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 2. 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.
- 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 Java EE 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.

Related concepts:
Operations with the REST data service
Related tasks:
Accessing data with the REST data service
Configuring REST data services
Related reference:
Optimistic concurrency in the REST data service
Request protocols for the REST data service
Retrieve requests with the REST data service
Retrieving non-entities with REST data services
Insert requests with REST data services
Update requests with REST data services
Delete requests with REST data services

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: 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.

- 8.5+ Scenario: Using JCA to connect transactional applications to eXtreme Scale clients
  The following scenario is about connecting clients to applications that participate in transactions.
- 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.
- 8.5+ 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.
- Migrating a WebSphere Application Server memory-to-memory replication or database session to use WebSphere eXtreme Scale session management
  You can migrate any previously set memory-to-memory replication session or database session to use WebSphere eXtreme Scale session management.

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 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.

1. 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.
2. Installing the Eclipse Equinox OSGi framework with Eclipse Gemini for clients and servers
   If you want to deploy WebSphere eXtreme Scale in the OSGi framework, then you must set up the Eclipse Equinox Environment.
3. 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.
4. 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.
5. 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.
6. 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.

Related concepts:
Samples
System APIs and plug-ins
Related tasks:
8.5+ Scenario: Using JCA to connect transactional applications to eXtreme Scale clients
Scenario: Configuring HTTP session failover in the Liberty profile
8.5+ Scenario: Running grid servers in the Liberty profile using Eclipse tools
Migrating a WebSphere Application Server memory-to-memory replication or database session to use WebSphere eXtreme Scale session management
Configuring eXtreme Scale plug-ins with OSGi Blueprint
Configuring OSGi-enabled plug-ins using the ObjectGrid descriptor XML file
Building eXtreme Scale dynamic plug-ins
Related information:
Building OSGi applications with the Blueprint Container specification
OSGi Bundle Activator API documentation
Spring namespace schema

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 in 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, enabling sophisticated options to integrate with other services 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 server feature of the eXtreme Scale stand-alone installation. It is required for running eXtreme Scale servers and can also be used for runningeXtreme 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 client feature of the eXtreme Scale stand-alone 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.

Next topic: Installing the Eclipse Equinox OSGi framework with Eclipse Gemini for clients and servers
Related tasks:
Programming to use the OSGi framework
Installing the Eclipse Equinox OSGi framework with Eclipse Gemini for clients and servers
Updating OSGi services for eXtreme Scale plug-ins with xscmd
Managing plug-in life cycles
Installing the Eclipse Equinox OSGi framework with Eclipse Gemini for clients and servers
Related reference:
Server properties file
Related information:
API documentation
Introduction: Starting and configuring the eXtreme Scale server and container to run plug-ins in the OSGi framework

Installing the Eclipse Equinox OSGi framework with Eclipse Gemini for clients and servers

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, \
gemini-blueprint-core-1.0.0.jar@1:start, \
gemini-blueprint-extender-1.0.0.jar@1:start, \
gemini-blueprint-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
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.

Previous topic: OSGi framework overview
Next topic: Running eXtreme Scale containers with non-dynamic plug-ins in an OSGi environment
Related concepts:
OSGi framework overview
Related tasks:
Programming to use the OSGi framework
Updating OSGi services for eXtreme Scale plug-ins with xscmd
Related reference:
Server properties file
Related information:
Introduction: Starting and configuring the eXtreme Scale server and container to run plug-ins in the OSGi framework

Installing eXtreme Scale bundles

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.
Related concepts:
Embedded server API

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.
Previous topic: Installing the Eclipse Equinox OSGi framework with Eclipse Gemini for clients and servers
Next topic: Administering eXtreme Scale servers and applications in an OSGi environment
Related concepts:
Java plug-ins overview
Related tasks:
Starting eXtreme Scale servers using the Eclipse Equinox OSGi framework

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/myOsgiEnv/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.
Previous topic: Running eXtreme Scale containers with non-dynamic plug-ins in an OSGi environment
Next topic: Building and running eXtreme Scale dynamic plug-ins for use in an OSGi environment

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
  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.
- Configuring eXtreme Scale plug-ins with OSGi Blueprint
  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.
- 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.

Previous topic: Administering eXtreme Scale servers and applications in an OSGi environment
Next topic: Running eXtreme Scale containers with dynamic plug-ins in an OSGi environment

Building eXtreme Scale dynamic plug-ins

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 BackingMapLifecyleListener 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 callsed 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 callsed 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.

Related tasks:
Upgrading agents and data models dynamically from OSGi bundles in the Liberty profile

Configuring eXtreme Scale plug-ins with OSGi Blueprint

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="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsblueprintosgi_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="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsblueprintosgi_h
ttp://www.ibm.com/schema/objectgrid"

           xsi:schemaLocation="http://www.ibm.com/schema/objectgrid
                http://www.ibm.com/schema/objectgrid/objectgrid.xsd">

    <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsblueprintosgi_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="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsblueprintosgi_myPluginBeanFac
tory"
    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.

Related concepts:
Samples
System APIs and plug-ins
Related tasks:
Configuring OSGi-enabled plug-ins using the ObjectGrid descriptor XML file
Building eXtreme Scale dynamic plug-ins
Scenario: Using an OSGi environment to develop and run eXtreme Scale plug-ins
Related information:
Building OSGi applications with the Blueprint Container specification
OSGi Bundle Activator API documentation
Spring namespace schema

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

Complete the following tasks before the OSGi-enabled plug-ins are installed and started.

- The eXtreme Scale server or client bundle is installed into the Eclipse Equinox OSGi framework. See Installing eXtreme Scale bundles.
- One or more dynamic BackingMap or ObjectGrid plug-ins are implemented. See Building eXtreme Scale dynamic plug-ins.
- The dynamic plug-ins are packaged as OSGi services in OSGi bundles.
- By default, the JVM continues to run when each eXtreme Scale server in an OSGi framework is stopped in the **xscmd** utility with the **-c teardown** command. If you want eXtreme Scale to exit the JVM after each server is stopped, then set the server property exitJVMOnTeardown to `true`. For more information, see Server properties file.

About this task

Install the bundle with the Eclipse Equinox console. There are several different methods to install the bundle, including a modification of the config.ini configuration file. Products that embed Eclipse Equinox include alternative methods for adding bundles in the config.ini file. For more information, seeEclipse runtime options.

OSGi allows bundles to be started that have duplicate services. WebSphere eXtreme Scale uses the latest service ranking. When multiple OSGi frameworks are started 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, see Updating OSGi services for eXtreme Scale plug-ins with xscmd.

Procedure

Install the plug-in bundle into the Eclipse Equinox OSGi framework with the OSGi console.

1. Start the Eclipse Equinox framework with the console enabled.

   ```
   <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 lists 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 started:

   ```
   osgi> ss
   ```

   When the bundle starts, the bundle lists the ACTIVE state, for example:

   ```
   17      ACTIVE      com.mycompany.plugin.bundle_VRM
   ```

Install the plug-in bundle into the Eclipse Equinox OSGi framework with 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 that 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 started. For example:

   ```
   osgi> ss
   ```

   When the bundle starts, the bundle lists 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 developingeXtreme 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
  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.
- 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.
- 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.
- Configuring servers with OSGi Blueprint
  You can configure WebSphere eXtreme Scale container servers using an OSGi blueprint XML file, allowing simplified packaging and development of self-contained server bundles.

Previous topic: Building and running eXtreme Scale dynamic plug-ins for use in an OSGi environment

Configuring OSGi-enabled plug-ins using the ObjectGrid descriptor XML file

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="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsconfigosgiplugs_myTranCallbac
k" 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="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsconfigosgiplugs_myPluginColle
ctionRef1">
            <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsconfigosgiplugs_MapSerializer
Plugin" osgiService="mySerializerFactory"/>
        </backingMapPluginCollection>
        <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsconfigosgiplugs_myPluginColle
ctionRef2">
            <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsconfigosgiplugs_MapSerializer
Plugin" osgiService="myOtherSerializerFactory"/>
            <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsconfigosgiplugs_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.

Related concepts:
Samples
System APIs and plug-ins
Related tasks:
Configuring eXtreme Scale plug-ins with OSGi Blueprint
Building eXtreme Scale dynamic plug-ins
Scenario: Using an OSGi environment to develop and run eXtreme Scale plug-ins
Related information:
Building OSGi applications with the Blueprint Container specification
OSGi Bundle Activator API documentation
Spring namespace schema

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:

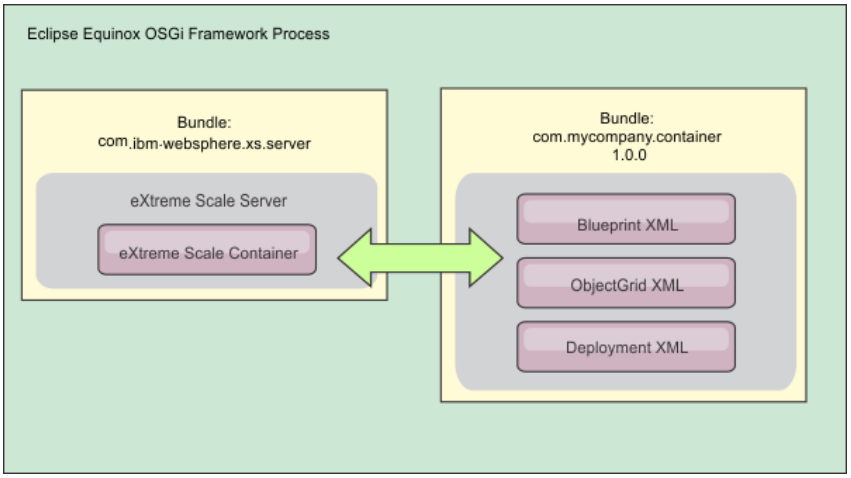
1. The WebSphere eXtreme Scale server bundle must be installed into Eclipse Equinox.
2. Your application must be packaged as an OSGi bundle.
3. Your WebSphere eXtreme Scale plug-ins (if any) must be packaged as an OSGi bundle. They can be bundled in the same bundle as your application or as separate 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:

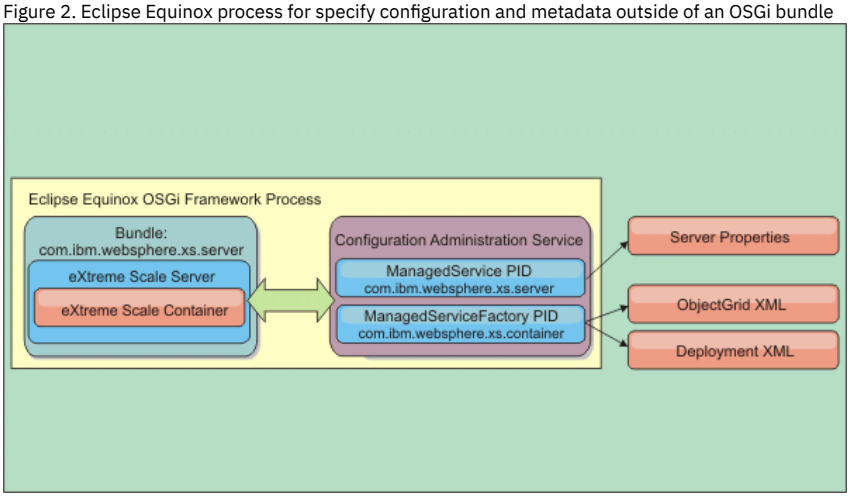
- OSGi Blueprint service
  You can include all configuration and metadata in an OSGi bundle. See the following image to understand the Eclipse Equinox process for this method:

  Figure 1. 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 2. 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.
  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.

Related tasks:
Running eXtreme Scale containers with non-dynamic plug-ins in an OSGi environment

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.

- Updating OSGi services for eXtreme Scale plug-ins with xscmd
  WebSphere eXtreme Scale supports upgrading container server plug-in bundles while the grid is active. This support allows administrators to complete application updates and additions without needing to restart grid processes.

Related tasks:
Updating OSGi services for eXtreme Scale plug-ins with xscmd
Administering with the xscmd utility
Managing ObjectGrid availability
Related reference:
Plug-ins for providing event listeners
Related information:
Eclipse runtime options

Configuring servers with OSGi Blueprint

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="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsconfigblue_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="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsconfigblue_xsServer"
tracespec="ObjectGridOSGi=all=enabled"
tracefile="logs/osgi/wxsserver/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="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsconfigblue_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

8.5+ 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 it 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.

1. 8.5+ 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.
2. 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 or later, or on an application server such as WebSphere Application Server.
3. 8.5+ Configuring eXtreme Scale connection factories
   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.
4. 8.5+ Configuring Eclipse environments to use eXtreme Scale connection factories
   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.
5. 8.5+ 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.
6. 8.5+ Securing J2C client connections
   Use the Java 2 Connector (J2C) architecture to secure connections between WebSphere eXtreme Scale clients and your applications.
7. 8.5+ Developing eXtreme Scale client components to use transactions
   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.
8. 8.5+ Administering J2C client connections
   The WebSphere eXtreme Scale connection factory includes an eXtreme Scale client connection that can be shared between applications and persisted through application restarts.

Related tasks:
Scenario: Using an OSGi environment to develop and run eXtreme Scale plug-ins
Scenario: Configuring HTTP session failover in the Liberty profile
8.5+ Scenario: Running grid servers in the Liberty profile using Eclipse tools
Migrating a WebSphere Application Server memory-to-memory replication or database session to use WebSphere eXtreme Scale session management

Transaction processing in Java EE applications

8.5+ 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.

Next topic: Installing an eXtreme Scale resource adapter
Related information:
Unshareable and shareable connections
Connection handles
Transaction type and connection behavior
Transaction support in WebSphere Application Server
Global transactions
Local transaction containment
Local and global transactions

Installing an eXtreme Scale resource adapter

8.5+ 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 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.
Previous topic: 8.5+ Transaction processing in Java EE applications
Next topic: 8.5+ Configuring eXtreme Scale connection factories

Related information:
Installing a resource adapter archive
Installing resource adapters embedded within applications
Resource adapter collection

Configuring eXtreme Scale connection factories

8.5+ 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 1. 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 for more information.

- Configuring Eclipse environments to use eXtreme Scale connection factories
  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.

Previous topic: Installing an eXtreme Scale resource adapter
Next topic: 8.5+ Configuring Eclipse environments to use eXtreme Scale connection factories
Related tasks:
Configuring catalog servers and catalog service domains
Related reference:
Client properties file
Related information:
Catalog service domain settings
Configuring connection factories for resource adapters within applications
Configuring Java EE Connector connection factories in the administrative console
Configuring new J2C connection factories using wsadmin scripting
J2C connection factories collection
Connection factory JNDI name practices

Configuring Eclipse environments to use eXtreme Scale connection factories

8.5+ 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.

Previous topic: 8.5+ Configuring eXtreme Scale connection factories
Next topic: 8.5+ Configuring applications to connect with eXtreme Scale

Configuring applications to connect with eXtreme Scale

8.5+ 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.

Previous topic: 8.5+ Configuring Eclipse environments to use eXtreme Scale connection factories
Next topic: 8.5+ Securing J2C client connections
Related information:

- Unshareable and shareable connections
- Resource reference benefits
- Creating or changing a resource reference

Securing J2C client connections

8.5+ 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.
Previous topic: 8.5+ Configuring applications to connect with eXtreme Scale
Next topic: 8.5+ Developing eXtreme Scale client components to use transactions

Developing eXtreme Scale client components to use transactions

8.5+ 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.LocalTransction 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.

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 = ((XSConnection)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 = ((XSConnection)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 = ((XSConnection)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 = ((XSConnection)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 = ((XSConnection)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(); }
            }
        }
    /**
         /**
```

Previous topic: 8.5+ Securing J2C client connections
Next topic: 8.5+ Administering J2C client connections
Related information:
Resource reference benefits

Developing components to use transactions

Administering J2C client connections

8.5+ 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 XSConnectionFactory 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.

Previous topic: 8.5+ Developing eXtreme Scale client components to use transactions
Related information:
JCA lifecycle management
Object grid J2C connection MBean API documentation

Scenario: Configuring HTTP session failover in the Liberty profile

8.5+ 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 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.

- 8.5 Enabling the eXtreme Scale web feature in the Liberty profile
  You can enable the web feature to use HTTP session failover in the Liberty profile.
- Configuring a web server plug-in to forward requests to multiple servers in the Liberty profile
  Use this task to configure the web server plug-in to distribute HTTP server requests between multiple servers in the Liberty profile.
- Merging plug-in configuration files for deployment to the application server plug-in
  Generate plug-in configuration files after you configure a unique clone ID in the Liberty server.xml configuration file.
- 8.5+ Enabling the eXtreme Scale web feature in the Liberty profile
  You can enable the web feature to use HTTP session failover in the Liberty profile.
- 8.5+ Configuring a web server plug-in to forward requests to multiple servers in the Liberty profile
  Use this task to configure the web server plug-in to distribute HTTP server requests between multiple servers in the Liberty profile.
- 8.5+ Merging plug-in configuration files for deployment to the application server plug-in
  Generate plug-in configuration files after you configure a unique clone ID in the Liberty server.xml configuration file.

Related tasks:
Scenario: Using an OSGi environment to develop and run eXtreme Scale plug-ins
8.5+ Scenario: Using JCA to connect transactional applications to eXtreme Scale clients
8.5+ Scenario: Running grid servers in the Liberty profile using Eclipse tools
Migrating a WebSphere Application Server memory-to-memory replication or database session to use WebSphere eXtreme Scale session management
8.5+

Enabling the eXtreme Scale web feature in the Liberty profile

8.5+ 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

8.5 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.
Related reference:
Liberty profile web feature properties
Liberty profile xsWebGrid feature properties
Liberty profile webApp feature properties

Configuring a web server plug-in to forward requests to multiple servers in the Liberty profile

8.5+ 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:

- 8.5 Enabling the eXtreme Scale web feature in the Liberty profile

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

8.5+ 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
- Configuring a web server plug-in to forward requests to multiple servers in the Liberty profile

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

8.5+ 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.

- 8.5+ 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.
- 8.5+ 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.

Related tasks:
Scenario: Using an OSGi environment to develop and run eXtreme Scale plug-ins
8.5+ Scenario: Using JCA to connect transactional applications to eXtreme Scale clients
Scenario: Configuring HTTP session failover in the Liberty profile
Migrating a WebSphere Application Server memory-to-memory replication or database session to use WebSphere eXtreme Scale session management

Installing the Liberty profile developer tools for WebSphere eXtreme Scale

8.5+ 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 Installing the Liberty profile developer tools and (optionally) the Liberty profile 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.

Related concepts:
Embedded server API

Setting up your development environment within Eclipse

8.5+ 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.

- 8.5+ 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.
- 8.5+ Creating an OSGi bundle project for eXtreme Scale data 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.

Configuring eXtreme Scale in the Liberty profile using Eclipse tools

8.5+ 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 data grid development

8.5+ 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.

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 IBM Elastic Caching Community Samples illustrate how to use WebSphere eXtreme Scale in various environments to exhibit different features of the product.

Articles with tutorials and examples

Table 1. Available articles by feature

Article	Features

Article	Features
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
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.
- 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.
- 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.

Related concepts:
Serialization using Java
Serialization overview
Serialization using the DataSerializer plug-ins
ObjectTransformer plug-in
Java plug-ins overview
Plug-ins for serializing cached objects
Serializer programming overview
IBM eXtremeMemory
Serializer programming overview
Serialization overview
Related tasks:
Avoiding object inflation when updating and retrieving cache data
Planning to use IBM eXtremeMemory
Configuring eXtreme Scale plug-ins with OSGi Blueprint
Configuring OSGi-enabled plug-ins using the ObjectGrid descriptor XML file
Building eXtreme Scale dynamic plug-ins
Scenario: Using an OSGi environment to develop and run eXtreme Scale plug-ins
Avoiding object inflation when updating and retrieving cache data
Programming to use the OSGi framework
Related information:
Oracle Java Serialization API
Building OSGi applications with the Blueprint Container specification
OSGi Bundle Activator API documentation
Spring namespace schema
DataSerializer API documentation

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

Related reference:
Server properties file
Client properties file

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 ObjectGridA -setstatsspec ALL=enabled
     ```

     Windows

     ```
     xsadmin.bat 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 **startOgServer** 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.

  - Navigate to the WebSphere Application Server bin directory:

    ```
    cd was_root/bin
    ```

  - 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.
- 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.
- 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.

Related reference:
xsadmin utility reference
Related information:
developerWorks: Enhancing xsadmin for WebSphere eXtreme Scale
developerWorks: Enhancing xsadmin for WebSphere eXtreme Scale

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.

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.

Table 1. 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...>`. *foreign_domain_name* is the name of the foreign catalog service domain, and *host1:port1,host2:port2...* 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
-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

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:** -mapsizes, -teardown,-revisions,-getTraceSpec,-setTraceSpec,-getStatsSpec,-setStatsSpec
-fz	n/a	Filters for only this zone. **Use with the following arguments:** -mapsizes, -teardown,-revisions,-getTraceSpec,-setTraceSpec,-getStatsSpec,-setStatsSpec,-routetable
-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 -setstatsspec argument. **Allows the following filters:** `-fst -fc -fz -fs -fh -fp`
-getTraceSpec	n/a	Displays the current trace specification. You can set the trace specification with the -settracespec argument. **Allows the following filters:** `-fst -fc -fz -fs -fh -fp`
-h	n/a	Displays the help for the **xsadmin** 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: `service:jmx:`*`protocol`*`:`*`sap`*. The *protocol* and *sap* variable definitions follow: *protocol* Specifies the transport protocol to be used to connect to the connector server. *sap* Specifies the address at which the connector server is found. For more information about the format of the JMX service URL, see Class JMXServiceURL (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:** `-fm -fst -fc -fz -fs -fh -fp`
-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 **xsadmin** 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:
		
		• Entire configuration:
		`-placementStatus`
		• For a specific data grid:
		`-placementStatus -g my_grid`
		• For a specific map set:
		`-placementStatus -m my_mapset`
		• For a specific data grid and map set:
		`-placementStatus -g my_grid -m my_mapset`

-primaries	n/a	Displays a list of the primary shards.
-profile	n/a	Specifies a fully qualified path to the properties file for the **xsadmin** utility.
-quorumstatus	n/a	Displays the status of quorum for the catalog service.

Command Line Argument	Equivalent Property Name in Properties File	Description and valid values
-releaseShard <*container_server_name*> <*objectgrid_name*> <*map_set_name*> <*partition_name*>	n/a	Used in conjunction with the -reserveShard argument. The -releaseShard argument must be invoked after a shard has been reserved and placed. . The -releaseShard argument invokes the ContainerMBean.release() method.
-reserved	n/a	Used with the -containers argument to display only shards that have been reserved with the -reserveShard argument.
-reserveShard <*container_server_name*> <*objectgrid_name*> <*map_set_name*> <*partition_name*>	n/a	Moves a primary shard to the specified container server. The ContainerMBean.reserve() method is invoked by this argument.
-resumeBalancing <*objectgrid_name*> <*map_set_name*>	n/a	Attempts to balance requests. Enables future rebalancing attempts on the specified ObjectGrid and map set.
-revisions	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 ObjectGridMBean.getKnownRevisions() method. **Allows the following filters:** `-fst -fc -fz -fs -fh -fp`
-routetable	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 `TargetNotAvailable` exceptions. **Required arguments:** In a stand-alone environment, you must specify the -bp and -p 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:** `-fz -fh -fp`
-settracespec <*trace_string*>	n/a	Enables trace on servers during run time. See the following example: **`-setTraceSpec "ObjectGridReplication=all=enabled"`** See Collecting trace and Server trace options for more information about the trace strings that you can specify. **Allows the following filters:** `-fst -fc -fz -fs -fh -fp`
-swapShardWithPrimary <*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.
-setstatsspec <*stats_spec*>	n/a	Enables statistics gathering. This argument invokes the DynamicServerMBean.setStatsSpec and DynamicServerMBean.getStatsSpec methods. For more information, see Enabling statistics. **Allows the following filters:** `-fm -fst -fc -fz -fs -fh -fp`
-suspendBalancing <*objectgrid_name*> <*map_set_name*>	n/a	Prevents future attempts to balance the specified ObjectGrid and map set.
-ssl	n/a	Indicates that Secure Sockets Layer (SSL) is enabled.
-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.

Command Line Argument	Equivalent Property Name in Properties File	Description and valid values
-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.

Related tasks:
Sample: xsadmin utility
Stopping servers gracefully with the xscmd utility
Related information:
developerWorks: Enhancing xsadmin for WebSphere eXtreme Scale
developerWorks: Enhancing xsadmin for WebSphere eXtreme Scale

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
```

Tutorials

You can use tutorials to help you understand product usage scenarios, including entity manager, queries, and security.

- Tutorial: Querying a local in-memory data grid
  You can develop a local in-memory ObjectGrid that can store order information for a website, and use the ObjectQuery API to query the data grid.
- Tutorial: Storing order information in entities
  The tutorial for the entity manager shows you how to use WebSphere® eXtreme Scale to store order information on a Web site. You can create a simple Java™ Platform, Standard Edition 5 application that uses an in-memory, local data grid. The entities use Java SE 5 annotations and generics.
- Tutorial: Configuring Java SE security
  With the following tutorial, you can create a distributed eXtreme Scale environment in a Java Platform, Standard Edition environment.
- 8.5 Tutorial: Run eXtreme Scale clients and servers in the Liberty profile
  You can run WebSphere eXtreme Scale in the Liberty profile.
- Tutorial: Integrate WebSphere eXtreme Scale security with WebSphere Application Server
  This tutorial demonstrates how to secure a WebSphere eXtreme Scale server deployment in a WebSphere Application Server environment.
- Tutorial: Integrate WebSphere eXtreme Scale security in a mixed environment with an external authenticator
  This tutorial demonstrates how to secure WebSphere eXtreme Scale servers that are partially deployed in a WebSphere Application Server environment.
- Tutorial: Running eXtreme Scale bundles in the OSGi framework
  The OSGi sample builds on the Google Protocol Buffers serializer samples. When you complete this set of lessons, you will have run the serializer sample plug-ins in the OSGi framework.

Tutorial: Querying a local in-memory data grid

You can develop a local in-memory ObjectGrid that can store order information for a website, and use the ObjectQuery API to query the data grid.

Before you begin

Be sure to have objectgrid.jar file in the classpath.

About this task

Each step in the tutorial builds on the previous step. Follow each of the steps to build a simple Java™ Platform, Standard Edition Version 5 or later application that uses an in-memory, local data grid.

1. ObjectQuery tutorial - step 1
   With the following steps, you can continue to develop a local, in-memory ObjectGrid that stores order information for an online retail store using the ObjectMap APIs. You define a schema for the map and run a query against the map.
2. ObjectQuery tutorial - step 2
   With the following steps, you can continue to create an ObjectGrid with one map and an index, along with a schema for the map. Then you can insert an object into the cache and later retrieve it using a simple query.
3. ObjectQuery tutorial - step 3
   With the following step, you can create an ObjectGrid with two maps and a schema for the maps with a relationship, then insert objects into the cache and later retrieve them using a simple query.
4. ObjectQuery tutorial - step 4
   The following step shows how to create an ObjectGrid with four maps and a schema for the maps. Some of the maps maintain a one-to-one (unidirectional) and one-to-many (bidirectional) relationship. After creating the maps, you can then run the sample Application.java program to insert objects into the cache and run queries to retrieve these objects.

ObjectQuery tutorial - step 1

With the following steps, you can continue to develop a local, in-memory ObjectGrid that stores order information for an online retail store using the ObjectMap APIs. You define a schema for the map and run a query against the map.

Procedure

1. Create an ObjectGrid with a map schema.
   Create an ObjectGrid with one map schema for the map, then insert an object into the cache and later retrieve it using a simple query.

   OrderBean.java

   ```
   public class OrderBean implements Serializable {
       String orderNumber;
       java.util.Date date;
       String customerName;
       String itemName;
       int quantity;
       double price;
   }
   ```

2. Define the primary key.
   The previous code shows an OrderBean object. This object implements the java.io.Serializable interface because all objects in the cache must (by default) be Serializable.

   The orderNumber attribute is the primary key of the object. The following example program can be run in stand-alone mode. You should follow this tutorial in an Eclipse Java™ project that has the objectgrid.jar file added to the class path.

   Application.java

   ```
   package querytutorial.basic.step1;

   import java.util.Iterator;

   import com.ibm.websphere.objectgrid.ObjectGrid;
   import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
   import com.ibm.websphere.objectgrid.ObjectMap;
   import com.ibm.websphere.objectgrid.Session;
   import com.ibm.websphere.objectgrid.config.QueryConfig;
   import com.ibm.websphere.objectgrid.config.QueryMapping;
   import com.ibm.websphere.objectgrid.query.ObjectQuery;

   public class Application
   {

       static public void main(String [] args) throws Exception
       {
           ObjectGrid og = ObjectGridManagerFactory.getObjectGridManager().createObjectGrid();
           og.defineMap("Order");

           // Define the schema
           QueryConfig queryCfg = new QueryConfig();
           queryCfg.addQueryMapping(new QueryMapping("Order", OrderBean.class.getName(),
                                                     "orderNumber", QueryMapping.FIELD_ACCESS));
           og.setQueryConfig(queryCfg);

           Session s = og.getSession();
           ObjectMap orderMap = s.getMap("Order");

           s.begin();
           OrderBean o = new OrderBean();
           o.customerName = "John Smith";
   ```

```
        o.date = new java.util.Date(System.currentTimeMillis());
        o.itemName = "Widget";
        o.orderNumber = "1";
        o.price = 99.99;
        o.quantity = 1;
        orderMap.put(o.orderNumber, o);
        s.commit();

        s.begin();
        ObjectQuery query = s.createObjectQuery("SELECT o FROM Order o WHERE o.itemName='Widget'");
        Iterator result = query.getResultIterator();
        o = (OrderBean) result.next();
        System.out.println("Found order for customer: " + o.customerName);
        s.commit();
        // Close the session (optional in Version 7.1.1 and later) for improved performance
        s.close();
}
}
```

This eXtreme Scale application first initializes a local ObjectGrid with an automatically generated name. Next, the application creates a BackingMap and a QueryConfig that defines what Java type is associated with the map, the name of the field that is the primary key for the map, and how to access the data in the object. You then obtain a Session to get the ObjectMap instance and insert an OrderBean object into the map in a transaction.

After the data is committed into the cache, you can use ObjectQuery to find the OrderBean using any of the persistent fields in the class. Persistent fields are those that do not have the transient modifier. Because you did not define any indexes on the BackingMap, ObjectQuery must scan each object in the map using Java reflection.

What to do next

ObjectQuery tutorial - step 2 demonstrates how an index can be used to optimize the query.

Next topic: ObjectQuery tutorial - step 2

ObjectQuery tutorial - step 2

With the following steps, you can continue to create an ObjectGrid with one map and an index, along with a schema for the map. Then you can insert an object into the cache and later retrieve it using a simple query.

Before you begin

Be sure that you have completed ObjectQuery tutorial - step 1 before proceeding with this step of the tutorial.

Procedure

Schema and index

`Application.java`

```
// Create an index
    HashIndex idx= new HashIndex();
    idx.setName("theItemName");
    idx.setAttributeName("itemName");
    idx.setRangeIndex(true);
    idx.setFieldAccessAttribute(true);
    orderBMap.addMapIndexPlugin(idx);
}
```

The index must be a com.ibm.websphere.objectgrid.plugins.index.HashIndex instance with the following settings:

- The Name is arbitrary, but must be unique for a given BackingMap.
- The AttributeName is the name of the field or bean property which the indexing engine uses to introspect the class. In this case, it is the name of the field for which you will create an index.
- RangeIndex must always be true.
- FieldAccessAttribute should match the value set in the QueryMapping object when the query schema was created. In this case, the Java™ object is accessed using the fields directly.

When a query runs that filters on the itemName field, the query engine automatically uses the defined index. Using the index allows the query to run much faster and a map scan is not needed. The next step demonstrates how an index can be used to optimize the query.

Next step

Previous topic: ObjectQuery tutorial - step 1
Next topic: ObjectQuery tutorial - step 3

ObjectQuery tutorial - step 3

With the following step, you can create an ObjectGrid with two maps and a schema for the maps with a relationship, then insert objects into the cache and later retrieve them using a simple query.

Before you begin

Be sure you have completed ObjectQuery tutorial - step 2 prior to proceeding with this step.

About this task

In this example, there are two maps, each with a single Java™ type mapped to it. The Order map has OrderBean objects and the Customer map has CustomerBean objects in it.

Procedure

Define maps with a relationship.

`OrderBean.java`

```
public class OrderBean implements Serializable {
    String orderNumber;
    java.util.Date date;
    String customerId;
    String itemName;
    int quantity;
    double price;
}
```

The OrderBean no longer has the customerName in it. Instead, it has the customerId, which is the primary key for the CustomerBean object and the Customer map.

`CustomerBean.java`

```
public class CustomerBean implements Serializable{
    private static final long serialVersionUID = 1L;
    String id;
    String firstName;
    String surname;
    String address;
    String phoneNumber;
}
```

The relationship between the two types or Maps follows:

`Application.java`

```
public class Application
{

    static public void main(String [] args)
        throws Exception
    {
        ObjectGrid og = ObjectGridManagerFactory.getObjectGridManager().createObjectGrid();
        og.defineMap("Order");
        og.defineMap("Customer");

        // Define the schema
        QueryConfig queryCfg = new QueryConfig();
        queryCfg.addQueryMapping(new QueryMapping(
            "Order", OrderBean.class.getName(), "orderNumber", QueryMapping.FIELD_ACCESS));
        queryCfg.addQueryMapping(new QueryMapping(
            "Customer", CustomerBean.class.getName(), "id", QueryMapping.FIELD_ACCESS));
        queryCfg.addQueryRelationship(new QueryRelationship(
             OrderBean.class.getName(), CustomerBean.class.getName(), "customerId", null));
        og.setQueryConfig(queryCfg);

        Session s = og.getSession();
        ObjectMap orderMap = s.getMap("Order");
        ObjectMap custMap = s.getMap("Customer");

        s.begin();
        CustomerBean cust = new CustomerBean();
        cust.address = "Main Street";
        cust.firstName = "John";
        cust.surname = "Smith";
        cust.id = "C001";
        cust.phoneNumber = "5555551212";
        custMap.insert(cust.id, cust);

        OrderBean o = new OrderBean();
        o.customerId = cust.id;
        o.date = new java.util.Date();
        o.itemName = "Widget";
        o.orderNumber = "1";
```

```
        o.price = 99.99;
        o.quantity = 1;
        orderMap.insert(o.orderNumber, o);
        s.commit();

        s.begin();
        ObjectQuery query = s.createObjectQuery(
            "SELECT c FROM Order o JOIN o.customerId as c WHERE o.itemName='Widget'");
        Iterator result = query.getResultIterator();
        cust = (CustomerBean) result.next();
        System.out.println("Found order for customer: " + cust.firstName + " " + cust.surname);
        s.commit();
    // Close the session (optional in Version 7.1.1 and later) for improved performance
        s.close();
    }
}
```

The equivalent XML in the ObjectGrid deployment descriptor follows:

```
<?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="CompanyGrid">
      <backingMap name="Order"/>
      <backingMap name="Customer"/>

      <querySchema>
         <mapSchemas>
           <mapSchema
             mapName="Order"
             valueClass="com.mycompany.OrderBean"
             primaryKeyField="orderNumber"
             accessType="FIELD"/>
           <mapSchema
             mapName="Customer"
             valueClass="com.mycompany.CustomerBean"
             primaryKeyField="id"
             accessType="FIELD"/>
         </mapSchemas>
         <relationships>
           <relationship
             source="com.mycompany.OrderBean"
             target="com.mycompany.CustomerBean"
             relationField="customerId"/>
         </relationships>
      </querySchema>
    </objectGrid>
  </objectGrids>
</objectGridConfig>
```

What to do next

ObjectQuery tutorial - step 4, expands the current step by including field and property access objects and additional relationships.
Previous topic: ObjectQuery tutorial - step 2
Next topic: ObjectQuery tutorial - step 4

ObjectQuery tutorial - step 4

The following step shows how to create an ObjectGrid with four maps and a schema for the maps. Some of the maps maintain a one-to-one (unidirectional) and one-to-many (bidirectional) relationship. After creating the maps, you can then run the sample Application.java program to insert objects into the cache and run queries to retrieve these objects.

Before you begin

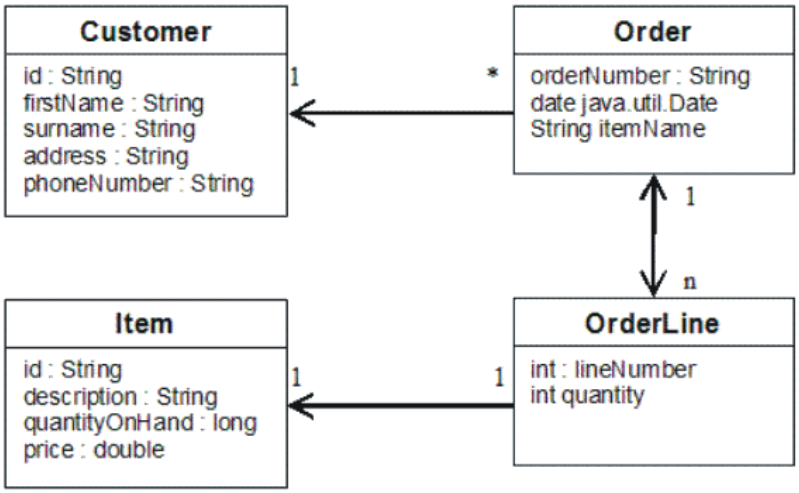
Be sure to have completed ObjectQuery tutorial - step 3 prior to continuing with the current step.

About this task

You are required to create four JAVA classes. These are the maps for the ObjectGrid:

- OrderBean.java
- OrderLineBean.java
- CustomerBean.java
- ItemBean.java

Figure 1. Order Schema. An Order schema has a one-to-one relationship with Customer and a one-to-many relationship with OrderLine. The OrderLine map has a one-to-one relationship with Item and includes the quantity ordered.



After creating these JAVA classes with these relationships, you can then run the sample Application.java program. This program lets you insert objects into the cache and retrieve these using several queries.

Procedure

1. Create the following JAVA classes:

```
OrderBean.java

public class OrderBean implements Serializable {
   String orderNumber;
   java.util.Date date;
   String customerId;
   String itemName;
   List<Integer> orderLines;
}

OrderLineBean.java

public class OrderLineBean implements Serializable {
         int lineNumber;
         int quantity;
         String orderNumber;
         String itemId;
}

CustomerBean.java

public class CustomerBean implements Serializable{
  String id;
  String firstName;
  String surname;
  String address;
  String phoneNumber;
}

ItemBean.java

public class ItemBean implements Serializable {
         String id;
         String description;
         long quantityOnHand;
         double price;
}
```

2. After creating the classes, you can run the sample Application.java:

```
Application.java


                public class Application static public void main(String [] args)throws Exception
        // Configure programatically
                                objectGrid og = ObjectGridManagerFactory.getObjectGridManager().createObjectGrid();
        og.defineMap("Order");
        og.defineMap("Customer");
        og.defineMap("OrderLine");
        og.defineMap("Item");

        // Define the schema
        QueryConfig queryCfg = new QueryConfig();
        queryCfg.addQueryMapping(new QueryMapping("Order", OrderBean.class.getName(), "orderNumber",
QueryMapping.FIELD_ACCESS));
        queryCfg.addQueryMapping(new QueryMapping("Customer", CustomerBean.class.getName(), "id",
```

```
QueryMapping.FIELD_ACCESS));
        queryCfg.addQueryMapping(new QueryMapping("OrderLine", OrderLineBean.class.getName(), "lineNumber",
QueryMapping.FIELD_ACCESS));
        queryCfg.addQueryMapping(new QueryMapping("Item", ItemBean.class.getName(), "id",
QueryMapping.FIELD_ACCESS));
        queryCfg.addQueryRelationship(new QueryRelationship(OrderBean.class.getName(),
CustomerBean.class.getName(), "customerId", null));
        queryCfg.addQueryRelationship(new QueryRelationship(OrderBean.class.getName(),
OrderLineBean.class.getName(),
                                "orderLines", "lineNumber"));
        queryCfg.addQueryRelationship(new QueryRelationship(OrderLineBean.class.getName(),
ItemBean.class.getName(), "itemId", null));
        og.setQueryConfig(queryCfg);

        // Get session and maps;
        Session s = og.getSession();
        ObjectMap orderMap = s.getMap("Order");
        ObjectMap custMap = s.getMap("Customer");
        ObjectMap itemMap = s.getMap("Item");
        ObjectMap orderLineMap = s.getMap("OrderLine");

        // Add data
        s.begin();
        CustomerBean aCustomer = new CustomerBean();
        aCustomer.address = "Main Street";
        aCustomer.firstName = "John";
        aCustomer.surname = "Smith";
        aCustomer.id = "C001";
        aCustomer.phoneNumber = "5555551212";
        custMap.insert(aCustomer.id, aCustomer);

        // Insert an order with a reference to the customer, but without any OrderLines yet.
        // Because we are using CopyMode.COPY_ON_READ_AND_COMMIT, the
        // insert won't be copied into the backing map until commit time, so
        // the reference is still good.

                                OrderBean anOrder = new OrderBean();
                                anOrder.customerId = aCustomer.id;
                                anOrder.date = new java.util.Date();
                                anOrder.itemName = "Widget";
                                anOrder.orderNumber = "1";
                                anOrder.orderLines = new ArrayList();
                                orderMap.insert(anOrder.orderNumber, anOrder);

        ItemBean anItem = new ItemBean();
        anItem.id = "AC0001";
        anItem.description = "Description of widget";
        anItem.quantityOnHand = 100;
        anItem.price = 1000.0;
        itemMap.insert(anItem.id, anItem);

                                // Create the OrderLines and add the reference to the Order
        OrderLineBean anOrderLine = new OrderLineBean();
        anOrderLine.lineNumber = 99;
        anOrderLine.itemId = anItem.id;
        anOrderLine.orderNumber = anOrder.orderNumber;
        anOrderLine.quantity = 500;
        orderLineMap.insert(anOrderLine.lineNumber, anOrderLine);
        anOrder.orderLines.add(Integer.valueOf(anOrderLine.lineNumber));

                                anOrderLine = new OrderLineBean();
        anOrderLine.lineNumber = 100;
        anOrderLine.itemId = anItem.id;
        anOrderLine.orderNumber = anOrder.orderNumber;
        anOrderLine.quantity = 501;
        orderLineMap.insert(anOrderLine.lineNumber, anOrderLine);
        anOrder.orderLines.add(Integer.valueOf(anOrderLine.lineNumber));
        s.commit();

                                s.begin();
        // Find all customers who have ordered a specific item.
        ObjectQuery query = s.createObjectQuery("SELECT c FROM Order o JOIN o.customerId as c WHERE
o.itemName='Widget'");
        Iterator result = query.getResultIterator();
        aCustomer = (CustomerBean) result.next();
        System.out.println("Found order for customer: " + aCustomer.firstName + " " + aCustomer.surname);
        s.commit();

                                s.begin();
        // Find all OrderLines for customer C001.
        // The query joins are expressed on the foreign keys.
        query = s.createObjectQuery("SELECT ol FROM Order o JOIN o.customerId as c JOIN o.orderLines as ol WHERE
c.id='C001'");
        result = query.getResultIterator();
        System.out.println("Found OrderLines:");
        while(result.hasNext()) {
            anOrderLine = (OrderLineBean) result.next();
            System.out.println(anOrderLine.lineNumber + ", qty=" + anOrderLine.quantity);
        }
        // Close the session (optional in Version 7.1.1 and later) for improved performance
        s.close();
```

```
                }
}
```

3. Using the XML configuration below (in the ObjectGrid deployment descriptor) is equivalent to the programmatic approach above.

```
<?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="CompanyGrid">
  <backingMap name="Order"/>
  <backingMap name="Customer"/>
  <backingMap name="OrderLine"/>
  <backingMap name="Item"/>

<querySchema>
 <mapSchemas>
  <mapSchema
               mapName="Order"
               valueClass="com.mycompany.OrderBean"
               primaryKeyField="orderNumber"
               accessType="FIELD"/>
  <mapSchema
               mapName="Customer"
               valueClass="com.mycompany.CustomerBean"
               primaryKeyField="id"
               accessType="FIELD"/>
  <mapSchema
               mapName="OrderLine"
               valueClass="com.mycompany.OrderLineBean"
               primaryKeyField="
               lineNumber"
               accessType="FIELD"/>
  <mapSchema
               mapName="Item"
               valueClass="com.mycompany.ItemBean"
               primaryKeyField="id"
               accessType="FIELD"/>
  </mapSchemas>

<relationships>
       <relationship
               source="com.mycompany.OrderBean"
               target="com.mycompany.CustomerBean"
               relationField="customerId"/>
       <relationship
               source="com.mycompany.OrderBean"
               target="com.mycompany.OrderLineBean"
               relationField="orderLines"
               invRelationField="lineNumber"/>
       <relationship
               source="com.mycompany.OrderLineBean"
               target="com.mycompany.ItemBean"
               relationField="itemId"/>
    </relationships>
   </querySchema>
  </objectGrid>
 </objectGrids>
</objectGridConfig>
```

Previous topic: ObjectQuery tutorial - step 3

Tutorial: Storing order information in entities

The tutorial for the entity manager shows you how to use WebSphere® eXtreme Scale to store order information on a Web site. You can create a simple Java™ Platform, Standard Edition 5 application that uses an in-memory, local data grid. The entities use Java SE 5 annotations and generics.

Before you begin

Ensure that you have met the following requirements before you begin the tutorial:

- You must have Java SE 5.
- You must have the objectgrid.jar file in your classpath.

1. Entity manager tutorial: Creating an entity class
   Create a local ObjectGrid with one entity by creating an Entity class, registering the entity type, and storing an entity instance into the cache.
2. Entity manager tutorial: Forming entity relationships
   Create a simple relationship between entities by creating two entity classes with a relationship, registering the entities with the ObjectGrid, and storing the entity instances into the cache.
3. Entity manager tutorial: Order Entity Schema
   Create four entity classes by using both single and bidirectional relationships, ordered lists, and foreign key relationships. The EntityManager APIs are used to persist and find the entities. Building on the Order and Customer entities that are in the previous parts of the tutorial, this tutorial step adds two more entities: the Item and OrderLine entities.

4. Entity manager tutorial: Updating entries
   If you want to change an entity, you can find the instance, update the instance and any referenced entities, and commit the transaction.
5. Entity manager tutorial: Updating and removing entries with an index
   You can use an index to find, update, and remove entities.
6. Entity manager tutorial: Updating and removing entries by using a query
   You can update and remove entities by using a query.

Related concepts:
Caching objects with no relationships involved (ObjectMap API)
Tuning EntityManager interface performance
Caching objects and their relationships (EntityManager API)
Entity manager in a distributed environment
Interacting with EntityManager
EntityManager fetch plan support
Entity query queues
Related reference:
Entity performance instrumentation agent
Defining an entity schema
Entity listeners and callback methods
Entity listener examples
EntityTransaction interface
Related information:
API documentation
Getting started tutorial lesson 2.1: Creating a client application

Entity manager tutorial: Creating an entity class

Create a local ObjectGrid with one entity by creating an Entity class, registering the entity type, and storing an entity instance into the cache.

Procedure

1. Create the Order object. To identify the object as an ObjectGrid entity, add the @Entity annotation. When you add this annotation, all serializable attributes in the object are automatically persisted in eXtreme Scale, unless you use annotations on the attributes to override the attributes. The orderNumber attribute is annotated with `@Id` to indicate that this attribute is the primary key. An example of an Order object follows:

   Order.java

   ```
   @Entity
   public class Order
   {
       @Id String orderNumber;
       Date date;
       String customerName;
       String itemName;
       int quantity;
       double price;
   }
   ```

2. Run the eXtreme Scale Hello World application to demonstrate the entity operations. The following example program can be issued in stand-alone mode to demonstrate the entity operations. Use this program in an Eclipse Java™ project that has the objectgrid.jar file added to the class path. An example of a simple Hello world application that uses eXtreme Scale follows:

   Application.java

   ```
   package emtutorial.basic.step1;

   import com.ibm.websphere.objectgrid.ObjectGrid;
   import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
   import com.ibm.websphere.objectgrid.Session;
   import com.ibm.websphere.objectgrid.em.EntityManager;

   public class Application
   {

       static public void main(String [] args)
           throws Exception
       {
           ObjectGrid og =
                                   ObjectGridManagerFactory.getObjectGridManager().createObjectGrid();
           og.registerEntities(new Class[] {Order.class});

           Session s = og.getSession();
           EntityManager em = s.getEntityManager();

           em.getTransaction().begin();

           Order o = new Order();
           o.customerName = "John Smith";
           o.date = new java.util.Date(System.currentTimeMillis());
           o.itemName = "Widget";
   ```

```
        o.orderNumber = "1";
        o.price = 99.99;
        o.quantity = 1;

        em.persist(o);
        em.getTransaction().commit();

        em.getTransaction().begin();
        o = (Order)em.find(Order.class, "1");
        System.out.println("Found order for customer: " + o.customerName);
        em.getTransaction().commit();
    }
}
```

This example application performs the following operations:

a. Initializes a local eXtreme Scale with an automatically generated name.
b. Registers the entity classes with the application by using the registerEntities API, although using the registerEntities API is not always necessary.
c. Retrieves a Session and a reference to the entity manager for the Session.
d. Associates each eXtreme Scale Session with a single EntityManager and EntityTransaction. The EntityManager is now used.
e. The registerEntities method creates a BackingMap object that is called Order, and associates the metadata for the Order object with the BackingMap object. This metadata includes the key and non-key attributes, along with the attribute types and names.
f. A transaction starts and creates an Order instance. The transaction is populated with some values. The transaction is then persisted by using the EntityManager.persist method, which identifies the entity as waiting to be included in the associated map.
g. The transaction is then committed, and the entity is included in the ObjectMap instance.
h. Another transaction is made, and the Order object is retrieved by using the key 1. The type cast on the EntityManager.find method is necessary. The Java SE 5 capability is not used to ensure that the objectgrid.jar file works on a Java SE Version 5 and later Java virtual machine.

Next topic: Entity manager tutorial: Forming entity relationships

Entity manager tutorial: Forming entity relationships

Create a simple relationship between entities by creating two entity classes with a relationship, registering the entities with the ObjectGrid, and storing the entity instances into the cache.

Procedure

1. Create the customer entity, which is used to store customer details independently from the Order object. An example of the customer entity follows:

```
Customer.java
@Entity
public class Customer
{
    @Id String id;
    String firstName;
    String surname;
    String address;
    String phoneNumber;
}
```

This class includes information about the customer such as name, address, and phone number.

2. Create the Order object, which is similar to the Order object in the Entity manager tutorial: Creating an entity class topic. An example of the order object follows:

```
Order.java

@Entity
public class Order
{
    @Id String orderNumber;
    Date date;
    @ManyToOne(cascade=CascadeType.PERSIST) Customer customer;
    String itemName;
    int quantity;
    double price;
}
```

In this example, a reference to a Customer object replaces the customerName attribute. The reference has an annotation that indicates a many-to-one relationship. A many-to-one relationship indicates that each order has one customer, but multiple orders might reference the same customer. The cascade annotation modifier indicates that if the entity manager persists the Order object, it must also persist the Customer object. If you choose to not set the cascade persist option, which is the default option, you must manually persist the Customer object with the Order object.

3. Using the entities, define the maps for the ObjectGrid instance. Each map is defined for a specific entity, and one entity is named Order and the other is named Customer. The following example application illustrates how to store and retrieve a customer order:

```
Application.java

public class Application
{
    static public void main(String [] args)
        throws Exception
    {
        ObjectGrid og =
```

```
                                        ObjectGridManagerFactory.getObjectGridManager().createObjectGrid();
        og.registerEntities(new Class[] {Order.class});

        Session s = og.getSession();
        EntityManager em = s.getEntityManager();

        em.getTransaction().begin();

        Customer cust = new Customer();
        cust.address = "Main Street";
        cust.firstName = "John";
        cust.surname = "Smith";
        cust.id = "C001";
        cust.phoneNumber = "5555551212";

        Order o = new Order();
        o.customer = cust;
        o.date = new java.util.Date();
        o.itemName = "Widget";
        o.orderNumber = "1";
        o.price = 99.99;
        o.quantity = 1;

        em.persist(o);
        em.getTransaction().commit();

        em.getTransaction().begin();
        o = (Order)em.find(Order.class, "1");
        System.out.println("Found order for customer: "
                                        + o.customer.firstName + " " + o.customer.surname);
        em.getTransaction().commit();
        // Close the session (optional in Version 7.1.1 and later) for improved performance
        s.close();
    }
}
```

This application is similar to the example application that is in the previous step. In the preceding example, only a single class Order is registered. WebSphere® eXtreme Scale detects and automatically includes the reference to the Customer entity, and a Customer instance for `John Smith` is created and referenced from the new Order object. As a result, the new customer is automatically persisted, because the relationship between two orders includes the cascade modifier, which requires that each object be persisted. When the Order object is found, the entity manager automatically finds the associated Customer object and inserts a reference to the object.

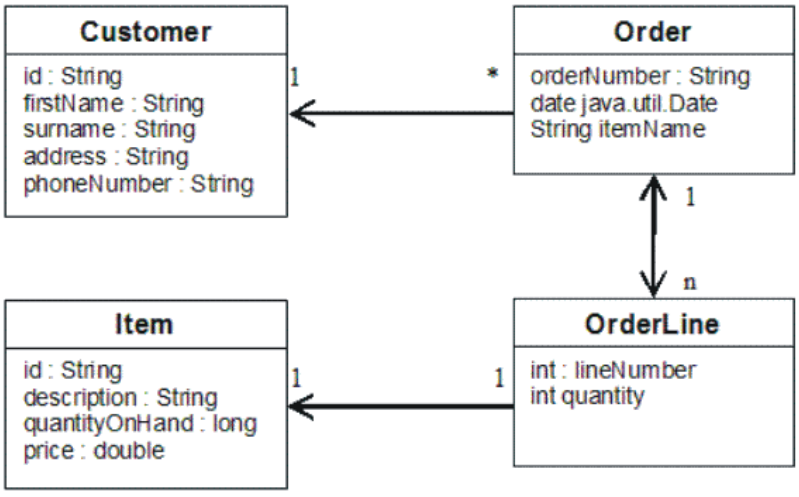
Previous topic: Entity manager tutorial: Creating an entity class
Next topic: Entity manager tutorial: Order Entity Schema

Entity manager tutorial: Order Entity Schema

Create four entity classes by using both single and bidirectional relationships, ordered lists, and foreign key relationships. The EntityManager APIs are used to persist and find the entities. Building on the Order and Customer entities that are in the previous parts of the tutorial, this tutorial step adds two more entities: the Item and OrderLine entities.

About this task

Figure 1. Order Entity Schema. An Order entity has a reference to one customer and zero or more OrderLines. Each OrderLine entity has a reference to a single item and includes the quantity ordered.



Procedure

1. Create the customer entity, which is similar to the previous examples.

```
Customer.java
@Entity
public class Customer
{
    @Id String id;
    String firstName;
    String surname;
    String address;
    String phoneNumber;
}
```

2. Create the Item entity, which holds information about a product that is included in the store's inventory, such as the product description, quantity, and price.

```
Item.java
@Entity
public class Item
{
    @Id String id;
    String description;
    long quantityOnHand;
    double price;
}
```

3. Create the OrderLine entity. Each Order has zero or more OrderLines, which identify the quantity of each item in the order. The key for the OrderLine is a compound key that consists of the Order that owns the OrderLine and an integer that assigns the order line a number. Add the cascade persist modifier to every relationship on your entities.

```
OrderLine.java
@Entity
public class OrderLine
{
    @Id @ManyToOne(cascade=CascadeType.PERSIST) Order order;
    @Id int lineNumber;
    @OneToOne(cascade=CascadeType.PERSIST) Item item;
    int quantity;
    double price;
}
```

4. Create the final Order Object, which has a reference to the Customer for the order and a collection of OrderLine objects.

```
Order.java
@Entity
public class Order
{
    @Id String orderNumber;
    java.util.Date date;
    @ManyToOne(cascade=CascadeType.PERSIST) Customer customer;
    @OneToMany(cascade=CascadeType.ALL, mappedBy="order")
                        @OrderBy("lineNumber") List<OrderLine> lines;
}
```

   The cascade ALL is used as the modifier for lines. This modifier signals the EntityManager to cascade both the PERSIST operation and the REMOVE operation. For example, if the Order entity is persisted or removed, then all OrderLine entities are also persisted or removed.

   If an OrderLine entity is removed from the lines list in the Order object, the reference is then broken. However, the OrderLine entity is not removed from the cache. You must use the EntityManager remove API to remove entities from the cache. The REMOVE operation is not used on the customer entity or the item entity from OrderLine. As a result, the customer entity remains even though the order or item is removed when the OrderLine is removed.

   The mappedBy modifier indicates an inverse relationship with the target entity. The modifier identifies which attribute in the target entity references the source entity, and the owning side of a one-to-one or many-to-many relationship. Typically, you can omit the modifier. However, an error is displayed to indicate that it must be specified if WebSphere® eXtreme Scale cannot discover it automatically. An OrderLine entity that contains two of type Order attributes in a many-to-one relationship typically causes the error.

   The @OrderBy annotation specifies the order in which each OrderLine entity should be in the lines list. If the annotation is not specified, then the lines display in an arbitrary order. Although the lines are added to the Order entity by issuing ArrayList, which preserves the order, the EntityManager does not necessarily recognize the order. When you issue the find method to retrieve the Order object from the cache, the list object is not an ArrayList object.

5. Create the application. The following example illustrates the final Order object, which has a reference to the Customer for the order and a collection of OrderLine objects.
   a. Find the Items to order, which then become Managed entities.
   b. Create the OrderLine and attach it to each Item.
   c. Create the Order and associate it with each OrderLine and the customer.
   d. Persist the order, which automatically persists each OrderLine.
   e. Commit the transaction, which detaches each entity and synchronizes the state of the entities with the cache.
   f. Print the order information. The OrderLine entities are automatically sorted by the OrderLine ID.

```
Application.java

static public void main(String [] args)
        throws Exception
    {
        ...

        // Add some items to our inventory.
```

```
        em.getTransaction().begin();
        createItems(em);
        em.getTransaction().commit();

        // Create a new customer with the items in his cart.
        em.getTransaction().begin();
        Customer cust = createCustomer();
        em.persist(cust);

        // Create a new order and add an order line for each item.
        // Each line item is automatically persisted since the
                                // Cascade=ALL option is set.
        Order order = createOrderFromItems(em, cust, "ORDER_1",
                                        new String[]{"1", "2"}, new int[]{1,3});
        em.persist(order);
        em.getTransaction().commit();

        // Print the order summary
        em.getTransaction().begin();
        order = (Order)em.find(Order.class, "ORDER_1");
        System.out.println(printOrderSummary(order));
        em.getTransaction().commit();
    }

    public static Customer createCustomer() {
        Customer cust = new Customer();
        cust.address = "Main Street";
        cust.firstName = "John";
        cust.surname = "Smith";
        cust.id = "C001";
        cust.phoneNumber = "5555551212";
        return cust;
    }

    public static void createItems(EntityManager em) {
        Item item1 = new Item();
        item1.id = "1";
        item1.price = 9.99;
        item1.description = "Widget 1";
        item1.quantityOnHand = 4000;
        em.persist(item1);

        Item item2 = new Item();
        item2.id = "2";
        item2.price = 15.99;
        item2.description = "Widget 2";
        item2.quantityOnHand = 225;
        em.persist(item2);

    }

    public static Order createOrderFromItems(EntityManager em,
                        Customer cust, String orderId, String[] itemIds, int[] qty) {

        Item[] items = getItems(em, itemIds);

        Order order = new Order();
        order.customer = cust;
        order.date = new java.util.Date();
        order.orderNumber = orderId;
        order.lines = new ArrayList<OrderLine>(items.length);
                        for(int i=0;i<items.length;i++){
                                                OrderLine line = new OrderLine();
            line.lineNumber = i+1;
            line.item = items[i];
            line.price = line.item.price;
            line.quantity = qty[i];
            line.order = order;
            order.lines.add(line);
        }
        return order;
    }

    public static Item[] getItems(EntityManager em, String[] itemIds) {
        Item[] items = new Item[itemIds.length];
        for(int i=0;i<items.length;i++){
                        items[i] = (Item) em.find(Item.class, itemIds[i]);
        }
        return items;
    }
```

The next step is to delete an entity. The EntityManager interface has a remove method that marks an object as deleted. The application should remove the entity from any relationship collections before calling the remove method. Edit the references and issue the remove method, or em.remove(object), as a final step.

Previous topic: Entity manager tutorial: Forming entity relationships
Next topic: Entity manager tutorial: Updating entries

Entity manager tutorial: Updating entries

If you want to change an entity, you can find the instance, update the instance and any referenced entities, and commit the transaction.

Before you begin

Procedure

Update entries. The following example demonstrates how to find the Order instance, change it and any referenced entities, and commit the transaction.

```
public static void updateCustomerOrder(EntityManager em) {
    em.getTransaction().begin();
    Order order = (Order) em.find(Order.class, "ORDER_1");
    processDiscount(order, 10);
    Customer cust = order.customer;
    cust.phoneNumber = "5075551234";
    em.getTransaction().commit();
}

public static void processDiscount(Order order, double discountPct) {
    for(OrderLine line : order.lines) {
        line.price = line.price * ((100-discountPct)/100);
    }
}
```

Flushing the transaction synchronizes all managed entities with the cache. When a transaction is committed, a flush automatically occurs. In this case, the Order becomes a managed entity. Any entities that are referenced from the Order, Customer, and OrderLine also become managed entities. When the transaction is flushed, each of the entities are checked to determine if they have been modified. Those that are modified are updated in the cache. After the transaction completes, by either being committed or rolled back, the entities become detached and any changes that are made in the entities are not reflected in the cache.

Previous topic: Entity manager tutorial: Order Entity Schema
Next topic: Entity manager tutorial: Updating and removing entries with an index

Entity manager tutorial: Updating and removing entries with an index

You can use an index to find, update, and remove entities.

Procedure

Update and remove entities by using an index. Use an index to find, update, and remove entities. In the following examples, the Order entity class is updated to use the @Index annotation. The @Index annotation signalsWebSphere® eXtreme Scale to create a range index for an attribute. The name of the index is the same name as the name of the attribute and is always a MapRangeIndex index type.

```
Order.java
@Entity
public class Order
{
    @Id String orderNumber;
    @Index java.util.Date date;
    @OneToOne(cascade=CascadeType.PERSIST) Customer customer;
    @OneToMany(cascade=CascadeType.ALL, mappedBy="order")
                                @OrderBy("lineNumber") List<OrderLine> lines;  }
```

The following example demonstrates how to cancel all orders that are submitted within the last minute. Find the order by using an index, add the items in the order back into the inventory, and remove the order and the associated line items from the system.

```
public static void cancelOrdersUsingIndex(Session s)
                throws ObjectGridException {
    // Cancel all orders that were submitted 1 minute ago
    java.util.Date cancelTime = new
                        java.util.Date(System.currentTimeMillis() - 60000);
    EntityManager em = s.getEntityManager();
    em.getTransaction().begin();
    MapRangeIndex dateIndex = (MapRangeIndex)
                        s.getMap("Order").getIndex("date");
    Iterator<Tuple> orderKeys = dateIndex.findGreaterEqual(cancelTime);
                while(orderKeys.hasNext()) {
                        Tuple orderKey = orderKeys.next();
                        // Find the Order so we can remove it.
                        Order curOrder = (Order) em.find(Order.class, orderKey);
                        // Verify that the order was not updated by someone else.
                        if(curOrder != null && curOrder.date.getTime() >= cancelTime.getTime()) {
                                for(OrderLine line : curOrder.lines) {
                                        // Add the item back to the inventory.
                                        line.item.quantityOnHand += line.quantity;
                                        line.quantity = 0;
                                }
                        em.remove(curOrder);
```

```
                        }
                    }
            em.getTransaction().commit();
}
```

Previous topic: Entity manager tutorial: Updating entries
Next topic: Entity manager tutorial: Updating and removing entries by using a query

Entity manager tutorial: Updating and removing entries by using a query

You can update and remove entities by using a query.

Procedure

Update and remove entities by using a query.

```
Order.java
@Entity
public class Order
{
    @Id String orderNumber;
    @Index java.util.Date date;
    @OneToOne(cascade=CascadeType.PERSIST) Customer customer;
    @OneToMany(cascade=CascadeType.ALL, mappedBy="order")
                        @OrderBy("lineNumber") List<OrderLine> lines;
}
```

The order entity class is the same as it is in the previous example. The class still provides the @Index annotation, because the query string uses the date to find the entity. The query engine uses indices when they can be used.

```
public static void cancelOrdersUsingQuery(Session s) {
        // Cancel all orders that were submitted 1 minute ago
        java.util.Date cancelTime =
                                        new java.util.Date(System.currentTimeMillis() - 60000);
        EntityManager em = s.getEntityManager();
        em.getTransaction().begin();

        // Create a query that will find the order based on date.  Since
        // we have an index defined on the order date, the query
                                // will automatically use it.
        Query query = em.createQuery("SELECT order FROM Order order
                                        WHERE order.date >= ?1");
        query.setParameter(1, cancelTime);
        Iterator<Order> orderIterator = query.getResultIterator();
                         while(orderIterator.hasNext()) {
                                        Order order = orderIterator.next();
                                        // Verify that the order wasn't updated by someone else.
                                        // Since the query used an index, there was no lock on the row.
                                        if(order != null && order.date.getTime() >= cancelTime.getTime()) {
                                                for(OrderLine line : order.lines) {
                                                        // Add the item back to the inventory.
                                                        line.item.quantityOnHand += line.quantity;
                                                        line.quantity = 0;
                                                }
                                                em.remove(order);
                                        }
                                }
                    em.getTransaction().commit();
}
```

Like the previous example, the cancelOrdersUsingQuery method intends to cancel all orders that were submitted in the past minute. To cancel the order, you find the order using a query, add the items in the order back into the inventory, and remove the order and associated line items from the system.
Previous topic: Entity manager tutorial: Updating and removing entries with an index

Tutorial: Configuring Java SE security

With the following tutorial, you can create a distributed eXtreme Scale environment in a Java™ Platform, Standard Edition environment.

Before you begin

Ensure that you are familiar with the basics of a distributed eXtreme Scale configuration.

About this task

Use this tutorial when you have installed eXtreme Scale in a stand-alone environment. Each step in the tutorial builds on the previous one. Follow each of the steps to secure a distributed eXtreme Scale and develop a simple Java SE application to access the secured eXtreme Scale.
Begin tutorial

1. Java SE security tutorial - Step 1
   In order to work with the rest of the tutorial, you need to create and package a simple Java program and two XML files. These set of files defines a simple ObjectGrid configuration with one ObjectGrid instance named `accounting` and a `customer` map. The SimpleDP.xml file features a deployment policy of one map set configured with one partition and zero minimum required replicas.
2. Java SE security tutorial - Step 2
   Before you can verify that the SimpleApp.java sample runs, you need to start a catalog server and a container server. After starting these services successfully, you can then launch the client and run the sample. Additional security features are added incrementally in the steps of the tutorial to increase the amount of integrated security that is available.
3. Java SE security tutorial - Step 3
   The rest of the tutorial demonstrates how to enable client authentication before connecting to an eXtreme Scale server. To prepare for the next step of this tutorial, you need to package the SecureSimpleApp.java program into a JAR and create a set of configuration files, which include a security.xml file, and two JAAS configuration files. The security.xml file lets you write authentication into the environment, and the JAAS configuration files provide the authentication mechanism when connecting to the server.
4. Java SE security tutorial - Step 4
   Building on the previous step, the following topic shows how to implement client authentication in a distributed eXtreme Scale environment.
5. Java SE security tutorial - Step 5
   After authenticating a client, as in the previous step, you can give security privileges through eXtreme Scale authorization mechanisms.
6. Java SE security tutorial - Step 6
   The following step explains how you can enable a security layer for communication between your environment's endpoints.

Java SE security tutorial - Step 1

In order to work with the rest of the tutorial, you need to create and package a simple Java program and two XML files. These set of files defines a simple ObjectGrid configuration with one ObjectGrid instance named `accounting` and a `customer` map. The SimpleDP.xml file features a deployment policy of one map set configured with one partition and zero minimum required replicas.

Procedure

1. In a command line window, go to the *wxs_home* directory.
2. Create a directory called applib.
3. Ensure your development environment contains the ogclient.jar file in the classpath. For more information, see the *Programming Guide*.
4. Create and compile the following SimpleApp.java class:

```
SimpleApp.java
// 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-J34 (C) COPYRIGHT International Business Machines Corp. 2007-2009
package com.ibm.websphere.objectgrid.security.sample.guide;

import com.ibm.websphere.objectgrid.ClientClusterContext;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectMap;
import com.ibm.websphere.objectgrid.Session;

public class SimpleApp {

    public static void main(String[] args) throws Exception {

        SimpleApp app = new SimpleApp();
        app.run(args);
    }

    /**
     * read and write the map
     * @throws Exception
     */
    protected void run(String[] args) throws Exception {
        ObjectGrid og = getObjectGrid(args);

        Session session = og.getSession();

        ObjectMap customerMap = session.getMap("customer");

        String customer = (String) customerMap.get("0001");

        if (customer == null) {
            customerMap.insert("0001", "fName lName");
        } else {
            customerMap.update("0001", "fName lName");
        }
        customer = (String) customerMap.get("0001");
        // Close the session (optional in Version 7.1.1 and later) for improved performance
        session.close();
```

```
        System.out.println("The customer name for ID 0001 is " + customer);
    }

    /**
     * Get the ObjectGrid
     * @return an ObjectGrid instance
     * @throws Exception
     */
    protected ObjectGrid getObjectGrid(String[] args) throws Exception {
        ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();

        // Create an ObjectGrid
        ClientClusterContext ccContext = ogManager.connect("localhost:2809", null, null);
        ObjectGrid og = ogManager.getObjectGrid(ccContext, "accounting");

        return og;

    }

}
```

5. Compile the package with this file and name the JAR sec_sample.jar. Put this JAR file in the /applib directory.
6. Go to the *wxs_home* directory, and create a directory called xml
7. In the*wxs_home*/xml directory, create the following configuration files:

```
SimpleApp.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="accounting">
            <backingMap name="customer" readOnly="false" copyKey="true"/>
        </objectGrid>
    </objectGrids>
</objectGridConfig>
```

The following XML file configures the deployment environment.

```
SimpleDP.xml

<?xml version="1.0" encoding="UTF-8"?>
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy ../deploymentPolicy.xsd"
        xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">

        <objectgridDeployment objectgridName="accounting">
                <mapSet name="mapSet1" numberOfPartitions="1" minSyncReplicas="0" maxSyncReplicas="2"
                        maxAsyncReplicas="1">
                        <map ref="customer"/>
                </mapSet>
        </objectgridDeployment>
</deploymentPolicy>
```

Results

These files create a simple ObjectGrid configuration with one ObjectGrid an `accounting` instance and a `customer` map.

Next topic: Java SE security tutorial - Step 2

Java SE security tutorial - Step 2

Before you can verify that the SimpleApp.java sample runs, you need to start a catalog server and a container server. After starting these services successfully, you can then launch the client and run the sample. Additional security features are added incrementally in the steps of the tutorial to increase the amount of integrated security that is available.

Before you begin

To successfully complete this step of the tutorial, you should have access to the following files:

- Have access to the compiled sec_sample.jar package. This package contains the SimpleApp.java program.
- Have access to the necessary configuration files SimpleApp.xml and SimpleDP.xml.

You should have created these files in Java SE security tutorial - Step 1 of this tutorial.
You should also know how to:

- Start and stop a catalog servers and container servers. For more information, see Starting and stopping stand-alone servers.
- Run the **xscmd** utility in order verify the map size inserted into the data grid.

Procedure

1. In a command line window, go to the *wxs_home*/bin directory and start the catalog service.
   - UNIX Linux `./startOgServer.sh catalogServer`
   - Windows `startOgServer.bat catalogServer`
2. Start a container service named `c0`:
   - UNIX Linux `./startOgServer.sh c0 -objectGridFile ../xml/SimpleApp.xml -deploymentPolicyFile ../xml/SimpleDP.xml -catalogServiceEndPoints localhost:2809`
   - Windows `startOgServer.bat c0 -objectGridFile ..\xml\SimpleApp.xml - deploymentPolicyFile ..\xml\SimpleDP.xml -catalogServiceEndPoints localhost:2809`
3. After the catalog server and container server have been started, run the sec_sample.jar sample as follows:
   UNIX

   ```
   java -classpath ../lib/objectgrid.jar:../applib/sec_sample.jar
   com.ibm.websphere.objectgrid.security.sample.guide.SimpleApp
   ```

   Windows

   ```
   java -classpath ..\lib\objectgrid.jar;..\applib\sec_sample.jar
   com.ibm.websphere.objectgrid.security.sample.guide.SimpleApp
   ```

   The output of the sample is: `The customer name for ID 0001 is fName lName` The getObjectGrid method in this class obtains an ObjectGrid, and the run method reads a record from the customer map and updates the value in the accounting grid.
4. Verify the size of the "customer" map inserted into the "accounting" grid, by issuing the **xscmd** command utility as follows:
   - UNIX Linux `./xscmd.sh -c showMapSizes -g accounting -ms mapSet1`
   - Windows `xscmd.bat -c showMapSizes -g accounting -ms mapSet1`
5. Stop a container server named `c0` with one of the following scripts:
   - UNIX Linux `./stopOgServer.sh c0 -catalogServiceEndPoints localhost:2809`
   - Windows `stopOgServer.bat c0 -catalogServiceEndPoints localhost:2809`

   If the server stopped successfully, then you will see the following message:
   `CWOBJ2512I: ObjectGrid server c0 stopped.`
6. Stop the catalog server with one of the following scripts:
   - UNIX Linux `./stopOgServer.sh catalogServer -catalogServiceEndPoints localhost:2809`
   - Windows `stopOgServer.bat catalogServer -catalogServiceEndPoints localhost:2809`

   If the server stopped successfully, then you will see the following message:
   `CWOBJ2512I: ObjectGrid server catalogServer stopped.`

Previous topic: Java SE security tutorial - Step 1
Next topic: Java SE security tutorial - Step 3

Java SE security tutorial - Step 3

The rest of the tutorial demonstrates how to enable client authentication before connecting to an eXtreme Scale server. To prepare for the next step of this tutorial, you need to package the SecureSimpleApp.java program into a JAR and create a set of configuration files, which include a security.xml file, and two JAAS configuration files. The security.xml file lets you write authentication into the environment, and the JAAS configuration files provide the authentication mechanism when connecting to the server.

About this task

Procedure

1. In a command line window, go to the *wxs_home*/applib directory you created in Java SE security tutorial - Step 1.
2. Create and compile the following SecureSimpleApp.java class:
   Attention: In the following example, some lines of code are continued on the next line for publication purposes.

   ```
   SecureSimpleApp.java
   package com.ibm.websphere.objectgrid.security.sample.guide;

   import com.ibm.websphere.objectgrid.ClientClusterContext;
   import com.ibm.websphere.objectgrid.ObjectGrid;
   import com.ibm.websphere.objectgrid.ObjectGridManager;
   import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
   import com.ibm.websphere.objectgrid.security.config.ClientSecurityConfiguration;
   import com.ibm.websphere.objectgrid.security.config.ClientSecurityConfigurationFactory;
   import com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator;
   import com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredentialGenerator;

   public class SecureSimpleApp extends SimpleApp {

       public static void main(String[] args) throws Exception {

           SecureSimpleApp app = new SecureSimpleApp();
           app.run(args);
   ```

```
    }

    /**
     * Get the ObjectGrid
     * @return an ObjectGrid instance
     * @throws Exception
     */
    protected ObjectGrid getObjectGrid(String[] args) throws Exception {
        ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
        ogManager.setTraceFileName("logs/client.log");
        ogManager.setTraceSpecification("ObjectGrid*=all=enabled:ORBRas=all=enabled");

        // Creates a ClientSecurityConfiguration object using the specified file
        ClientSecurityConfiguration clientSC = ClientSecurityConfigurationFactory
                .getClientSecurityConfiguration(args[0]);

        // Creates a CredentialGenerator using the passed-in user and password.
        CredentialGenerator credGen = new UserPasswordCredentialGenerator(args[1], args[2]);
        clientSC.setCredentialGenerator(credGen);

        // Create an ObjectGrid by connecting to the catalog server
        ClientClusterContext ccContext = ogManager.connect("localhost:2809", clientSC, null);
        ObjectGrid og = ogManager.getObjectGrid(ccContext, "accounting");

        return og;

    }

}
```

3. Ensure your development environment contains the ogclient.jar file in the classpath. For more information, see the *Programming Guide*.
4. Compile the package with these files and name the JAR sec_sample.jar.
5. Change to the *wxs_home* directory.
6. Create a directory called security.
7. Create a configuration file called security.xml. Server security properties are specified in this file. These properties are common for both catalog servers and container servers.

```
security.xml
<?xml version="1.0" encoding="UTF-8"?>
<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" loginSessionExpirationTime="300" >

        <authenticator className
="com.ibm.websphere.objectgrid.security.plugins.builtins.KeyStoreLoginAuthenticator">
        </authenticator>
    </security>

</securityConfig>
```

Previous topic: Java SE security tutorial - Step 2
Next topic: Java SE security tutorial - Step 4

Java SE security tutorial - Step 4

Building on the previous step, the following topic shows how to implement client authentication in a distributed eXtreme Scale environment.

Before you begin

Be sure that you have completed Java SE security tutorial - Step 3. You need to have created and complied the SecureSimpleApp.java sample into a sec_sample.jar file, and created a configuration file called security.xml.

About this task

With client authentication enabled, a client is authenticated before connecting to the eXtreme Scale server. This section demonstrates how client authentication can be done in an eXtreme Scale server environment, using the sample SecureSimpleApp.java.
Client credential

The SecureSimpleApp.java sample uses the following two plug-in implementations to obtain client credentials:

`com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredential`

`com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredentialGenerator`

For more information about these plug-ins, see Client authentication programming.

Server authenticator

The example uses an eXtreme Scale built-in implementation: `KeyStoreLoginAuthenticator`, which is for testing and sample purposes (a keystore is a simple user registry and should not be used for production). For more information, see the topic on authenticator plug-in under Client authentication programming.

Procedure

1. In a command line window, go to the *wxs_home* directory.
2. Change to the *wxs_home*/security directory you had created in Java SE security tutorial - Step 3.
3. Create a JAAS configuration file that enforces a method of authentication to the server, og_jaas.config. The `KeyStoreLoginAuthenticator` referenced in the security.xml file uses a keystore by using the JAAS login module "KeyStoreLogin". The keystore can be configured as an option to the KeyStoreLoginModule class.

```
og_jaas.config
KeyStoreLogin{
com.ibm.websphere.objectgrid.security.plugins.builtins.KeyStoreLoginModule required
     keyStoreFile="../security/sampleKS.jks" debug = true;
};
```

Windows Important: If you are using Windows, the directory path does not support backslashes. If you have used backslashes, you must escape any backslash (\ ) characters in the path. For example, if you want to use the path C:\opt\ibm, enter C:\\opt\\ibm in the properties file. Windows directories with spaces are not supported.

4. Change to the *java_home*/bin directory and run the keytool.
5. Change to the *wxs_home* /security directory, and create two users, "manager" and "cashier" with their own passwords.
   a. Use the keytool to create a user "manager" with password "manager1" in the keystore sampleKS.jks.
      - UNIX Linux

```
keytool -genkey -v -keystore sampleKS.jks -storepass sampleKS1 \
-alias manager -keypass manager1 \
-dname CN=manager,O=acme,OU=OGSample -validity 10000
```

      - Windows

```
keytool -genkey -v -keystore sampleKS.jks -storepass sampleKS1 ^
-alias manager -keypass manager1 ^
-dname CN=manager,O=acme,OU=OGSample -validity 10000
```

   b. Use the keytool to create a user "cashier" with password "cashier1" in the keystore sampleKS.jks.
      - UNIX Linux

```
keytool -genkey -v -keystore sampleKS.jks -storepass sampleKS1 \
-alias cashier -keypass cashier1 \
-dname CN=cashier,O=acme,OU=OGSample -validity 10000
```

      - Windows

```
keytool -genkey -v -keystore sampleKS.jks -storepass sampleKS1 ^
-alias cashier -keypass cashier1 ^
-dname CN=cashier,O=acme,OU=OGSample -validity 10000
```

6. Make a copy of the sampleClient.properties file located in wxs_home/properties directory to wxs_home/security/client.properties
   - UNIX Linux

```
cp ../properties/sampleClient.properties client.properties
```

   - Windows

```
copy ..\properties\sampleClient.properties client.properties
```

7. In the *wxs_home*/security directory, save it as client.properties
   Make the following changes to the client.properties file:
   a. **securityEnabled:** Set securityEnabled to true (default value) enables the client security, which includes authentication.
   b. **credentialAuthentication:** Set credentialAuthentication to Supported (default value), which means the client supports credential authentication.
   c. **transportType:** Set transportType to TCP/IP, which means no SSL will be used.
8. Copy the sampleServer.properties file into the *wxs_home*/security directory and save it as server.properties.
   - UNIX Linux

```
cp ../properties/sampleServer.properties server.properties
```

   - Windows

```
copy ..\properties\sampleServer.properties server.properties
```

   Make the following changes in the server.properties file:
   a. **securityEnabled:** Set the securityEnabled attribute to true.
   b. **transportType:** Set transportType attribute to TCP/IP, which means no SSL is used.
   c. **secureTokenManagerType:** Set secureTokenManagerType attribute to none to not configure the secure token manager.
9. Go to the *wxs_home*/bin directory and depending on your platform, issue one of the following commands to start a catalog server. You need to issue the **-clusterSecurityFile** and **-serverProps** command line options to pass in security properties:
   - UNIX Linux

```
./startOgServer.sh catalogServer -clusterSecurityFile ../security/security.xml
-serverProps ../security/server.properties -jvmArgs
-Djava.security.auth.login.config="../security/og_jaas.config"
```

   - Windows

```
startOgServer.bat catalogServer -clusterSecurityFile ..\security\security.xml
-serverProps ..\security\server.properties -jvmArgs
-Djava.security.auth.login.config="..\security\og_jaas.config"
```

10. Start a container server named `c0` with one of the following scripts. The server property file is passed by issuing **-serverProps**.
    a.
    - UNIX Linux

      ```
      ./startOgServer.sh c0 -objectgridFile ../xml/SimpleApp.xml
      -deploymentPolicyFile ../xml/SimpleDP.xml
      -catalogServiceEndPoints localhost:2809
      -serverProps ../security/server.properties
      -jvmArgs -Djava.security.auth.login.config="../security/og_jaas.config"
      ```

    - Windows

      ```
      startOgServer.bat c0 -objectgridFile ..\xml\SimpleApp.xml
      -deploymentPolicyFile ..\xml\SimpleDP.xml
      -catalogServiceEndPoints localhost:2809
      -serverProps ..\security\server.properties
      -jvmArgs -Djava.security.auth.login.config="..\security\og_jaas.config"
      ```

11. After the catalog server and container server have been started, run the sec_sample.jar sample as follows:
    - UNIX Linux

      ```
      java -classpath ../lib/objectgrid.jar:../applib/sec_sample.jar
                      com.ibm.websphere.objectgrid.security.sample.guide.SecureSimpleApp
                      ../security/client.properties manager manager1
      ```

    - Windows

      ```
      java -classpath ..\lib\objectgrid.jar;..\applib\sec_sample.jar
                      com.ibm.websphere.objectgrid.security.sample.guide.SecureSimpleApp
                      ..\security\client.properties manager manager1
      ```

    Linux Use a colon (:) for the classpath separator instead of a semicolon (;) as in the previous example.

    After you issue the class, the following output results:

    The customer name for `ID 0001 is fName lName`.

12. Verify the size of the "customer" map inserted into the "accounting" grid, by issuing the **xscmd** command utility as follows:
    - UNIX Linux `./xscmd.sh -c showMapSizes -g accounting -m customer -username manager -password manager1`
    - Windows `xscmd.bat -c showMapSizes -g accounting -m customer -username manager -password manager1`
13. Optional: To stop the container or catalog servers, you can use the **stopOgServer** or **stopXsServer** command. However you need to provide a security configuration file. The sample client property file defines the following two properties to generate a userID/password credential (manager/manager1).

    ```
    credentialGeneratorClass=com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredentialGenerator
    ```

    ```
    credentialGeneratorProps=manager manager1
    ```

    Stop the container `c0` with the following command.

    - UNIX Linux `./stopOgServer.sh c0 -catalogServiceEndPoints localhost:2809 -clientSecurityFile ../security/client.properties`
    - Windows `stopOgServer.bat c0 -catalogServiceEndPoints localhost:2809 -clientSecurityFile ..\security\client.properties`

    If you do not provide the -clientSecurityFile option, you will see an exception with the following message.

    ```
    >> SERVER (id=39132c79, host=9.10.86.47) TRACE START:

    >> org.omg.CORBA.NO_PERMISSION: Server requires credential authentication but there is no security context from the
    client. This usually happens when the client does not pass a credential the server.

    vmcid: 0x0

    minor code: 0

    completed: No
    ```

    You can also shut down the catalog server using the following command. However, if you want to continue trying the next step tutorial, you can let the catalog server stay running.

    - UNIX Linux `./stopOgServer.sh catalogServer -catalogServiceEndPoints localhost:2809 -clientSecurityFile ../security/client.properties`
    - Windows `stopOgServer.bat catalogServer -catalogServiceEndPoints localhost:2809 -clientSecurityFile ..\security\client.properties`

    If you do shutdown the catalog server, you will see the following output.

    ```
    CWOBJ2512I: ObjectGrid server catalogServer stopped
    ```

    Now, you have successfully made your system partially secure by enabling authentication. You configured the server to plug in the user registry, configured the client to provide client credentials, and changed the client property file and cluster XML file to enable authentication.

    If you provide an invalidate password, you see an exception stating that the user name or password is not correct.

    For more details about client authentication, see Authenticating application clients.

Next step of tutorial

Previous topic: Java SE security tutorial - Step 3
Next topic: Java SE security tutorial - Step 5
Related tasks:
Configuring security profiles for the xscmd utility

Java SE security tutorial - Step 5

After authenticating a client, as in the previous step, you can give security privileges through eXtreme Scale authorization mechanisms.

Before you begin

Be sure to have completed Java SE security tutorial - Step 4 prior to proceeding with this task.

About this task

The previous step of this tutorial demonstrated how to enable authentication in an eXtreme Scale grid. As a result, no unauthenticated client can connect to your server and submit requests to your system. However, every authenticated client has the same permission or privileges to the server, such as reading, writing, or deleting data that is stored in the ObjectGrid maps. Clients can also issue any type of query. This section demonstrates how to use eXtreme Scale authorization to give various authenticated users varying privileges.
Similar to many other systems, eXtreme Scale adopts a permission-based authorization mechanism. WebSphere® eXtreme Scale has different permission categories that are represented by different permission classes. This topic features MapPermission. For complete category of permissions, see Client authorization programming.

In WebSphere eXtreme Scale, the com.ibm.websphere.objectgrid.security.MapPermission class represents permissions to the eXtreme Scale resources, specifically the methods of ObjectMap or JavaMap interfaces. WebSphere eXtreme Scale defines the following permission strings to access the methods of ObjectMap and JavaMap:

- read: Grants permission to read the data from the map.
- write: Grants permission to update the data in the map.
- insert: Grants permission to insert the data into the map.
- remove: Grants permission to remove the data from the map.
- invalidate: Grants permission to invalidate the data from the map.
- all: Grants all permissions to read, write, insert, remote, and invalidate.

The authorization occurs when a client calls a method of ObjectMap or JavaMap. The eXtreme Scale runtime environment checks different map permissions for different methods. If the required permissions are not granted to the client, an AccessControlException results.

This tutorial demonstrates how to use Java Authentication and Authorization Service (JAAS) authorization to grant authorization map accesses for different users.

Procedure

1. **Enable eXtreme Scale authorization**. To enable authorization on the ObjectGrid, you need to set the securityEnabled attribute to `true` for that particular ObjectGrid in the XML file. Enabling security on the ObjectGrid means that you are enabling authorization. Use the following commands to create a new ObjectGrid XML file with security enabled.
   a. Navigate to the xml directory.

   ```
   cd objectgridRoot/xml
   ```

   b. Copy the SimpleApp.xml file to the SecureSimpleApp.xml file.
   - UNIX Linux

     ```
     cp SimpleApp.xml SecureSimpleApp.xml
     ```

   - Windows

     ```
     copy SimpleApp.xml SecureSimpleApp.xml
     ```

   c. Open the SecureSimpleApp.xml file and add `securityEnabled="true"` on the ObjectGrid level as the following XML shows:

   ```
   <?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="accounting" securityEnabled="true">
               <backingMap name="customer" readOnly="false" copyKey="true"/>
           </objectGrid>
       </objectGrids>
   </objectGridConfig>
   ```

2. **Define the authorization policy.** In the previous client authentication topic, you created the users, cashier and manager, in the keystore. In this example, the user "cashier" only has read permissions to all the maps, and the user "manager" has all permissions. JAAS authorization is used in this example. You

must create a JAAS authorization policy file to grant permissions to principals. Create the following og_auth.policy file in the objectgridRoot/security directory:

```
og_auth.policy
grant codebase "http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction"
    principal javax.security.auth.x500.X500Principal "CN=cashier,O=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,O=acme,OU=OGSample" {
    permission com.ibm.websphere.objectgrid.security.MapPermission "accounting.*", "all";
};
```

Note:

- The `codebase "http://www.ibm.com/com/ibm/ws/objectgridRoot/security/PrivilegedAction"` is a specially-reserved URL for ObjectGrid. All ObjectGrid permissions granted to principals should use this special code base.
- The first grant statement grants "read" map permission to principal `"CN=cashier,O=acme,OU=OGSample"`, so the cashier has only map read permission to all the maps in the ObjectGrid accounting.
- The second grant statement grants "all" map permission to principal `"CN=manager,O=acme,OU=OGSample"`, so the manager has all permissions to maps in the ObjectGrid accounting.

Now you can launch a server with an authorization policy. The JAAS authorization policy file can be set using the standard -D property: -Djava.security.policy=../security/og_auth.policy

3. **Run the application.**

After you create the above files, you can run the application.

Use the following commands to start the catalog server. For more information about starting the catalog service, see Starting a stand-alone catalog service.

a. Navigate to the bin directory: `cd objectgridRoot/bin`

b. Start the catalog server.

- UNIX Linux

```
./startOgServer.sh catalogServer
-clusterSecurityFile ../security/security.xml
-serverProps ../security/server.properties
-jvmArgs -Djava.security.auth.login.config="../security/og_jaas.config"
```

- Windows

```
startOgServer.bat catalogServer
-clusterSecurityFile ..\security\security.xml
-serverProps ..\security\server.properties
-jvmArgs -Djava.security.auth.login.config="..\security\og_jaas.config"
```

The security.xml and server.properties files were created in the previous step of this tutorial.

c. You can then start a secure container server using the following script. Run the following script from the bin directory:

- UNIX Linux

```
./startOgServer.sh c0 -objectGridFile ../xml/SecureSimpleApp.xml
-deploymentPolicyFile ../xml/SimpleDP.xml
-catalogServiceEndPoints localhost:2809
-serverProps ../security/server.properties
-jvmArgs -Djava.security.auth.login.config="../security/og_jaas.config"
-Djava.security.policy="../security/og_auth.policy"
```

- Windows

```
startOgServer.bat c0 -objectGridFile ..\xml\SecureSimpleApp.xml
-deploymentPolicyFile ..\xml\SimpleDP.xml
-catalogServiceEndPoints localhost:2809
-serverProps ..\security\server.properties
-jvmArgs -Djava.security.auth.login.config="..\security\og_jaas.config"
-Djava.security.policy="..\security\og_auth.policy"
```

Notice the following differences from the previous container server start command:

- The SecureSimpleApp.xml file is used instead of SimpleApp.xml file, which is the result of your running the sample sec_sample.jar file to set client authentication.
- Another -Djava.security.policy argument was added to set the JAAS authorization policy file to the container server process.

Use the same command as in the previous step of the tutorial:

a. Navigate to the bin directory.

- UNIX Linux

```
java -classpath ../lib/objectgrid.jar:../applib/sec_sample.jar
com.ibm.websphere.objectgrid.security.sample.guide.SecureSimpleApp
../security/client.properties manager manager1
```

- Windows

```
java -classpath ..\lib\objectgrid.jar;..\applib\sec_sample.jar
com.ibm.websphere.objectgrid.security.sample.guide.SecureSimpleApp
 ..\security\client.properties manager manager1
```

b. Because user "manager" has all permissions to maps in the accounting ObjectGrid, the application runs properly.

Now, instead of using user "manager", use user "cashier" to launch the client application.

c. Navigate to the bin directory.

- UNIX Linux

```
java -classpath ../lib/objectgrid.jar:../applib/sec_sample.jar
com.ibm.ws.objectgrid.security.sample.guide.SecureSimpleApp
../security/client.properties cashier cashier1
```

- Windows

```
java -classpath ..\lib\objectgrid.jar;..\applib\sec_sample.jar
com.ibm.ws.objectgrid.security.sample.guide.SecureSimpleApp
..\security\client.properties cashier cashier1
```

The following exception results:

Attention: In the following example, some lines of code are continued on the next line for publication purposes.

```
Exception in thread "P=387313:O=0:CT" com.ibm.websphere.objectgrid.TransactionException:
rolling back transaction, see caused by exception
        at com.ibm.ws.objectgrid.SessionImpl.rollbackPMapChanges(SessionImpl.java:1422)
        at com.ibm.ws.objectgrid.SessionImpl.commit(SessionImpl.java:1149)
        at com.ibm.ws.objectgrid.SessionImpl.mapPostInvoke(SessionImpl.java:2260)
        at com.ibm.ws.objectgrid.ObjectMapImpl.update(ObjectMapImpl.java:1062)
        at com.ibm.ws.objectgrid.security.sample.guide.SimpleApp.run(SimpleApp.java:42)
        at com.ibm.ws.objectgrid.security.sample.guide.SecureSimpleApp.main(SecureSimpleApp.java:27)
Caused by: com.ibm.websphere.objectgrid.ClientServerTransactionCallbackException:
   Client Services - received exception from remote server:
     com.ibm.websphere.objectgrid.TransactionException: transaction rolled back,
                        see caused by Throwable
        at com.ibm.ws.objectgrid.client.RemoteTransactionCallbackImpl.processReadWriteResponse(
            RemoteTransactionCallbackImpl.java:1399)
        at com.ibm.ws.objectgrid.client.RemoteTransactionCallbackImpl.processReadWriteRequestAndResponse(
            RemoteTransactionCallbackImpl.java:2333)
        at
com.ibm.ws.objectgrid.client.RemoteTransactionCallbackImpl.commit(RemoteTransactionCallbackImpl.java:557)
        at com.ibm.ws.objectgrid.SessionImpl.commit(SessionImpl.java:1079)
        ... 4 more
Caused by: com.ibm.websphere.objectgrid.TransactionException: transaction rolled back, see caused by Throwable
        at com.ibm.ws.objectgrid.ServerCoreEventProcessor.processLogSequence(ServerCoreEventProcessor.java:1133)
        at com.ibm.ws.objectgrid.ServerCoreEventProcessor.processReadWriteTransactionRequest
                                        (ServerCoreEventProcessor.java:910)
        at
com.ibm.ws.objectgrid.ServerCoreEventProcessor.processClientServerRequest(ServerCoreEventProcessor.java:1285)

        at com.ibm.ws.objectgrid.ShardImpl.processMessage(ShardImpl.java:515)
        at com.ibm.ws.objectgrid.partition.IDLShardPOA._invoke(IDLShardPOA.java:154)
        at com.ibm.CORBA.poa.POAServerDelegate.dispatchToServant(POAServerDelegate.java:396)
        at com.ibm.CORBA.poa.POAServerDelegate.internalDispatch(POAServerDelegate.java:331)
        at com.ibm.CORBA.poa.POAServerDelegate.dispatch(POAServerDelegate.java:253)
        at com.ibm.rmi.iiop.ORB.process(ORB.java:503)
        at com.ibm.CORBA.iiop.ORB.process(ORB.java:1553)
        at com.ibm.rmi.iiop.Connection.respondTo(Connection.java:2680)
        at com.ibm.rmi.iiop.Connection.doWork(Connection.java:2554)
        at com.ibm.rmi.iiop.WorkUnitImpl.doWork(WorkUnitImpl.java:62)
        at com.ibm.rmi.iiop.WorkerThread.run(ThreadPoolImpl.java:202)
        at java.lang.Thread.run(Thread.java:803)
Caused by: java.security.AccessControlException: Access denied (
   com.ibm.websphere.objectgrid.security.MapPermission accounting.customer write)
        at java.security.AccessControlContext.checkPermission(AccessControlContext.java:155)
        at com.ibm.ws.objectgrid.security.MapPermissionCheckAction.run(MapPermissionCheckAction.java:141)
        at java.security.AccessController.doPrivileged(AccessController.java:275)
        at javax.security.auth.Subject.doAsPrivileged(Subject.java:727)
        at com.ibm.ws.objectgrid.security.MapAuthorizer$1.run(MapAuthorizer.java:76)
        at java.security.AccessController.doPrivileged(AccessController.java:242)
        at com.ibm.ws.objectgrid.security.MapAuthorizer.check(MapAuthorizer.java:66)
        at com.ibm.ws.objectgrid.security.SecuredObjectMapImpl.checkMapAuthorization(SecuredObjectMapImpl.java:429)
        at com.ibm.ws.objectgrid.security.SecuredObjectMapImpl.update(SecuredObjectMapImpl.java:490)
        at com.ibm.ws.objectgrid.SessionImpl.processLogSequence(SessionImpl.java:1913)
        at com.ibm.ws.objectgrid.SessionImpl.processLogSequence(SessionImpl.java:1805)
        at com.ibm.ws.objectgrid.ServerCoreEventProcessor.processLogSequence(ServerCoreEventProcessor.java:1011)
        ... 14 more
```

This exception occurs because the user "cashier" does not have write permission, so it cannot update the map customer.

Now your system supports authorization. You can define authorization policies to grant different permissions to different users. For more information about authorization, see Authorizing application clients.

What to do next

Complete the next step of the tutorial. See Java SE security tutorial - Step 6.
Previous topic: Java SE security tutorial - Step 4
Next topic: Java SE security tutorial - Step 6

Java SE security tutorial - Step 6

The following step explains how you can enable a security layer for communication between your environment's endpoints.

Before you begin

Be sure you have completed Java SE security tutorial - Step 5 prior to proceeding with this task.

About this task

The eXtreme Scale topology supports both Transport Layer Security/Secure Sockets Layer (TLS/SSL) for secure communication between ObjectGrid endpoints (client, container servers, and catalog servers). This step of the tutorial builds upon the previous steps to enable transport security.

Procedure

1. **Create TLS/SSL keys and keystores**.
   In order to enable transport security, you must create a keystore and trust store. This exercise only creates one key and trust-store pair. These stores are used for ObjectGrid clients, container servers, and catalog servers, and are created with the JDK keytool.
   - *Create a private key in the keystore*
     ```
     keytool -genkey -alias ogsample -keystore key.jks -storetype JKS -keyalg rsa -dname "CN=ogsample, OU=OGSample,
     O=acme, L=Your City, S=Your State, C=Your Country" -storepass ogpass -keypass ogpass -validity 3650
     ```
     Using this command, a keystore key.jks is created with a key "ogsample" stored in it. This keystore key.jks will be used as the SSL keystore.
   - *Export the public certificate*
     ```
     keytool -export -alias ogsample -keystore key.jks -file temp.key -storepass ogpass
     ```
     Using this command, the public certificate of key "ogsample" is extracted and stored in the file temp.key.
   - *Import the client's public certificate to the trust store*
     ```
     keytool -import -noprompt -alias ogsamplepublic -keystore trust.jks -file temp.key -storepass ogpass
     ```
     Using this command, the public certificate was added to keystore trust.jks. This trust.jks is used as the SSL trust store.
2. **Configure ObjectGrid property files**.
   In this step, you must configure the ObjectGrid property files to enable transport security.

   First, copy the key.jks and trust.jks files into the objectgridRoot/security directory.

   Set the following properties in the client.properties and server.properties file.

   ```
   transportType=SSL-Required

   alias=ogsample
   contextProvider=IBMJSSE2
   protocol=SSL
   keyStoreType=JKS
   keyStore=../security/key.jks
   keyStorePassword=ogpass
   trustStoreType=JKS
   trustStore=../security/trust.jks
   trustStorePassword=ogpass
   ```

   transportType: The value of transportType is set to "SSL-Required", which means the transport requires SSL. So all the ObjectGrid endpoints (clients, catalog servers, and container servers) should have SSL configuration set and all transport communication will be encrypted.

   The other properties are used to set the SSL configurations. See Transport layer security and secure sockets layer for a detailed explanation. Make sure you follow the instructions in this topic to update your orb.properties file.

   Make sure you follow this page to update your orb.properties file.

   In the server.properties file, you must add an additional property clientAuthentication and set it to false. On the server side, you do not need to trust the client.

   ```
   clientAuthentication=false
   ```
3. **Run the application**.
   The commands that you use in this step are the same as the commands in the Java SE security tutorial - Step 3 topic.

   a. Navigate to the `cd objectgridRoot/bin` directory, and use the following commands to start a catalog server:
   - Linux UNIX

     ```
     ./startOgServer.sh catalogServer -clusterSecurityFile ../security/security.xml
     -serverProps ../security/server.properties -JMXServicePort 11001
     -jvmArgs -Djava.security.auth.login.config="../security/og_jaas.config"
     ```
   - Windows

     ```
     startOgServer.bat catalogServer -clusterSecurityFile ..\security\security.xml
     -serverProps ..\security\server.properties -JMXServicePort 11001 -jvmArgs
     ```

```
-Djava.security.auth.login.config="..\security\og_jaas.config"
```

The security.xml and server.properties files were created in the Java SE security tutorial - Step 2 page.

Use the -JMXServicePort option to explicitly specify the JMX port for the server. This option is required to use the **xscmd** utility.

b. From the objectgridRoot/bin directory, start a secure ObjectGrid container server:

- Linux UNIX

```
./startOgServer.sh c0 -objectGridFile ../xml/SecureSimpleApp.xml
-deploymentPolicyFile ../xml/SimpleDP.xml -catalogServiceEndPoints
localhost:2809 -serverProps ../security/server.properties
-JMXServicePort 11002 -jvmArgs
-Djava.security.auth.login.config="../security/og_jaas.config"
-Djava.security.policy="../security/og_auth.policy"
```

- Windows

```
startOgServer.bat c0 -objectGridFile ..\xml\SecureSimpleApp.xml
-deploymentPolicyFile ..\xml\SimpleDP.xml -catalogServiceEndPoints localhost:2809
-serverProps ..\security\server.properties -JMXServicePort 11002
-jvmArgs -Djava.security.auth.login.config="..\security\og_jaas.config"
-Djava.security.policy="..\security\og_auth.policy"
```

c. From the objectgridRoot/bin directory, run the following command to complete client authentication:

- UNIX Linux

```
javaHome/java -classpath ../lib/objectgrid.jar:../applib/sec_sample.jar
com.ibm.websphere.objectgrid.security.sample.guide.SecureSimpleApp
../security/client.properties manager manager1
```

- Windows

```
javaHome\java -classpath ..\lib\objectgrid.jar;..\applib\sec_sample.jar
com.ibm.websphere.objectgrid.security.sample.guide.SecureSimpleApp
..\security\client.properties manager manager1
```

Because user "manager" has permission to all the maps in the accounting ObjectGrid, the application runs successfully.

4. **Use the xscmd utility to show the map sizes of the "accounting" data grid**.

a. From the objectgridRoot/bin directory, use the **xscmd** command to show the map sizes:

- UNIX Linux

```
./xscmd.sh -c showMapsizes -g accounting -m customer -prot SSL
-ts ../security/trust.jks -tsp ogpass -tst jks
-user manager -pwd manager1 -ks ../security/key.jks -ksp ogpass -kst JKS
-cxpv IBMJSSE2 -tt SSL-Required
```

- Windows

```
xscmd.bat -c showMapsizes -g accounting -m customer -prot SSL
-ts ..\security\trust.jks -tsp ogpass -tst jks
-user manager -pwd manager1 -ks ..\security\key.jks -ksp ogpass -kst JKS
-cxpv IBMJSSE2 -tt SSL-Required
```

You see the following output.

```
This administrative utility is provided as a sample only and is not to
be considered a fully supported component of the WebSphere eXtreme Scale product.
Connecting to Catalog service at localhost:1099
*********** Displaying Results for Grid - accounting, MapSet - customer ***********
*** Listing Maps for c0 ***
Map Name: customer Partition #: 0 Map Size: 1 Shard Type: Primary
Server Total: 1
Total Domain Count: 1
```

5. **Troubleshoot running the application with an incorrect keystore**.

If your truststore does not contain the public certificate of the private key in the keystore, an exception that the key cannot be trusted occurs.

To show this exception, create another keystore key2.jks.

```
keytool -genkey -alias ogsample -keystore key2.jks -storetype JKS -keyalg rsa -dname "CN=ogsample, OU=Your
Organizational Unit, O=Your Organization, L=Your City, S=Your State, C=Your Country" -storepass ogpass -keypass
ogpass -validity 3650
```

Then modify the server.properties file to make the keystore point to this new keystore key2.jks:

```
keyStore=../security/key2.jks
```

a. From the `cd objectgridRoot/bin` directory, assume that you run the following commands, which use an incorrect keystore, to start the catalog server:

- Linux UNIX

```
./startOgServer.sh c0 -objectGridFile ../xml/SecureSimpleApp.xml
-deploymentPolicyFile ../xml/SimpleDP.xml -catalogServiceEndPoints localhost:2809
-serverProps ../security/server.properties -JMXServicePort 11002 -jvmArgs
-Djava.security.auth.login.config="../security/og_jaas.config"
-Djava.security.policy="../security/og_auth.policy"
```

- Windows

```
startOgServer.bat c0 -objectGridFile ..\xml\SecureSimpleApp.xml
-deploymentPolicyFile ..\xml/SimpleDP.xml -catalogServiceEndPoints localhost:2809
 -serverProps ..\security\server.properties -JMXServicePort 11002 -jvmArgs
-Djava.security.auth.login.config="..\security\og_jaas.config"
-Djava.security.policy="..\security\og_auth.policy"
```

You receive the following exception:

```
CWPKI0022E: SSL HANDSHAKE FAILURE: A signer with SubjectDN "CN=ogsample,
OU=Your Organizational Unit, O=Your Organization, L=Your City, ST=Your State,
C=Your Country" was sent from target host:port "9.23.39.177:36407". The signer may
need to be added to local trust store
"/opt/IBM/WebSphere/eXtremeScale/ObjectGrid/security/trust.jks"
located in SSL configuration alias "DefaultSystemProperties" loaded from SSL
configuration file "System Properties". The extended error message from the SSL
handshake exception is: "PKIX path building failed: java.security.cert.CertPathBuilderException:
unable to find valid certification path to requested target".

CWPKI0040I: An SSL handshake failure occurred from a secure client. The server's SSL signer
has to be added to the client's trust store. A retrieveSigners utility is provided to download
signers from the server but requires administrative permission. Check with your administrator
to have this utility run to setup the secure environment before running the client. Alternatively,
the com.ibm.ssl.enableSignerExchangePrompt can be enabled in ssl.client.props for "DefaultSSLSettings"
in order to allow acceptance of the signer during the connection attempt.
```

To correct the exception, change the server.properties file back to use the key.jks file.

Previous topic: Java SE security tutorial - Step 5

Tutorial: Run eXtreme Scale clients and servers in the Liberty profile

8.5+ You can run WebSphere® eXtreme Scale in the Liberty profile.

Learning objectives

In this tutorial, you can expect to complete the following learning objectives:

- Install the Liberty profile and eXtreme Scale.
- Create eXtreme Scale servers and an application client for testing in the Liberty profile.
- Deploy WebSphere Application Server applications that run with eXtreme Scale in the Liberty profile.
- Configure session management for eXtreme Scale applications in the Liberty profile.

Time required

This tutorial takes approximately 30 minutes to finish. If you explore other concepts related to this tutorial, it could take longer to complete.

Prerequisites

To complete this tutorial, you must install the following products:

- Java Runtime Environment. For more information about supported Java™ environments, and where to get them, see Minimum supported Java levels in the Liberty profile: Runtime environment known restrictions.
- The Liberty profile
- WebSphere eXtreme Scale

It is also recommended that you have basic Java development knowledge.

- **Liberty profile**

  The Liberty profile is a highly composable, fast-to-start, dynamic application server runtime environment.
- **Module 1: Install the Liberty profile**

  You must install both the WebSphere Application Server Liberty profile and WebSphere eXtreme Scale to complete this tutorial.
- **Module 2: Create a web application server in the Liberty profile**

  You must create a server directory and server.xml file to develop the server definition for the Liberty profile.
- **Module 3: Add the Liberty web feature to the Liberty profile**

  Add the web feature to your server definition to identify web-based applications and add functions, such as session replication.
- **Module 4: Configure clients to use client APIs in the Liberty profile**

  You can configure your WebSphere eXtreme Scale clients to run in the Liberty profile.
- **Module 5: Run the data grid inside the Liberty profile**

  After you add the client and server configurations to the Liberty profile, you can run WebSphere eXtreme Scale in the Liberty profile.

Liberty profile

8.5+ 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
- z/OS z/OS workload 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.

z/OS On z/OS systems, the Liberty profile provides an operations environment. You can work natively with this environment using the MVS™ console. For application development, consider using the Eclipse-based developer tools on a separate distributed system, on Mac OS, or in a Linux shell on z/OS.

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.

Related tasks:
Configuring eXtreme Scale servers to use the Liberty profile
Installing the Liberty profile
Related reference:
Liberty profile server properties
Server properties file
| | Next >

Module 1: Install the Liberty profile

8.5+ You must install both the WebSphere® Application Server Liberty profile and WebSphere eXtreme Scale to complete this tutorial.

To install the Liberty profile, use IBM® Installation Manager to install WebSphere Application Server Version 8.5 or higher, and then install WebSphere eXtreme Scale for WebSphere Application Server Version 8.5 or higher.
Alternatively, you can install the Liberty profile by running a provided JAR file. However, you still must install WebSphere eXtreme Scale. For more information about downloading and installing the Liberty profile application-serving environment with a JAR file, see Installing the Liberty profile application-serving environment by running a JAR file.

Learning objectives

After completing the lessons in this module, you will know how to:

- Install the Liberty profile as the application serving environment for WebSphere eXtreme Scale. To run eXtreme Scale clients and servers in the Liberty profile, you must install WebSphere eXtreme Scale after you install the Liberty profile.

20 minutes
| | Next >
< Previous | Next >

Module 2: Create a web application server in the Liberty profile

8.5+ You must create a server directory and server.xml file to develop the server definition for the Liberty profile.

Learning objectives

After completing the lesson in this module, you will know how to:

- Define a server to run in the Liberty profile.

Prerequisites

To complete this module, you must:

- Install the Liberty profile.
- Install WebSphere® eXtreme Scale.

Lessons in this module

- **Lesson 2.1: Define a server to run in the Liberty profile**

  Create a server directory and a server.xml server definition file to run in the Liberty profile.

< Previous | Next >
< Previous | Next >

Lesson 2.1: Define a server to run in the Liberty profile

8.5+ Create a server directory and a server.xml server definition file to run in the Liberty profile.

In Module 1, you installed the Liberty profile and WebSphere® eXtreme Scale in the XSWLP/wlp directory. In this lesson, you will create a server directory and a server.xml server definition file in the XSWLP/wlp/bin subdirectory.

1. Open a command prompt from the XSWLP directory where you installed the Liberty profile.
2. Change the directory to the wlp/bin directory; for example, enter `cd wlp/bin`.
3. Enter the following command to create the server, where *defaultServer* is the server name:

   ```
   server create defaultServer
   ```

   If the command was successful, the following response is displayed:

   ```
   Server defaultServer created.
   ```

To verify that you created the server definition file, search for the server.xml file in the directory, XSWLP/wlp/usr/servers/defaultServer, where defaultServer is the server directory that you created in this lesson. The server.xml file contains the server configuration and Liberty profile features that client applications use in the Liberty profile. The defaultServer directory also includes a dropins folder, which you will use in the next lesson to run your eXtreme Scale client applications.

After you verify that the defaultServer directory exists, create a folder in that same directory called grids . In the Lesson 2.3, the grids folder is where you will place the XML descriptor files that eXtreme Scale servers need to run.

Module 3: Add the Liberty web feature to the Liberty profile

8.5+ Add the web feature to your server definition to identify web-based applications and add functions, such as session replication.

Learning objectives

After completing the lesson in this module, you will know how to:

- Define a web application to run in the Liberty profile.

Prerequisites

To complete this module, you must complete the following modules first

- Install the Liberty profile.
- Install WebSphere® eXtreme Scale.
- Create a web application server in the Liberty profile.

.

Lessons in this module

- **Lesson 3.1: Define a web application to run in the Liberty profile**

  Define the web feature to your server definition to enable application functions, such as session replication.

Lesson 3.1: Define a web application to run in the Liberty profile

8.5+ Define the web feature to your server definition to enable application functions, such as session replication.

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

The web feature has meta type properties that you can set on the `xsWebAppV85` element of the server.xml file. For more information, see Liberty profile web feature properties

Add the following web feature to the Liberty profile server.xml file. The web 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. 8.5+

```
<featureManager>
<feature>eXtremeScale.web-1.0</feature>
</featureManager>
```

Your web applications can now persist its session data in a WebSphere® eXtreme Scale grid.
See the following example of a server.xml file, which contains the web feature that you use when you connect to the data grid remotely. 8.5+

```
<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>servlet-3.0</feature>
        <feature>jsp-2.2</feature>
        <feature>eXtremeScale.web-1.1</feature>
    </featureManager>

    <httpEndpoint
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txslibertyaddwebfeattut_defaultHttpEn
dpoint"
        host="*"
        httpPort="${default.http.port}"
        httpsPort="${default.https.port}" />
```

```
    <xsWebAppV85 objectGridType="REMOTE" objectGridName="session" catalogHostPort="remoteHost:2809"
securityEnabled="false" />

</server>
```

< Previous | Next >
< Previous | Next >

Module 4: Configure clients to use client APIs in the Liberty profile

8.5+ You can configure your WebSphere® eXtreme Scale clients to run in the Liberty profile.

Learning objectives

After completing the lesson in this module, you will know how to:

- Configure the Liberty profile to run with eXtreme Scale clients.

Prerequisites

To complete this module, you must complete the following modules first:

- Install the Liberty profile.
- Install WebSphere eXtreme Scale.
- Create a web application server in the Liberty profile.
- Add the Liberty web feature to the web application.

Lessons in this module

- **Lesson 4.1: Configure the Liberty profile to run with eXtreme Scale clients**

  Use the WebSphere eXtreme Scale client feature to run the Liberty profile with eXtreme Scale clients.

< Previous | Next >
< Previous | Next >

Lesson 4.1: Configure the Liberty profile to run with eXtreme Scale clients

8.5+ Use the WebSphere® eXtreme Scale client feature to run the Liberty profile with eXtreme Scale clients.

This configuration provides only the client functionality. In this application, the server function runs in another process. Adding the client feature allows your application to access the eXtreme Scale APIs and connect to a remote grid.

This client configuration provides a single process that includes what you need to unit test a web application using an eXtreme Scale data grid. When you add the client feature, it starts a catalog server and a container server when the configuration is deployed into the grid directory. In addition, after you add the client feature, the application can write to the eXtreme Scale APIs.

1. Add the client feature to the Liberty server. Add the following code to the Liberty server:

   ```
   <server description="eXtreme Scale Container Server">

   <featureManager>
   <feature>eXtremeScale.client-1.0</feature>
   </featureManager>


   </server>
   ```

2. (Optional) Alternatively, you can use the eXtreme Scale server feature to reference the client configuration. When you add the following server configuration, the client functionality is automatically included:

   ```
   <server description="eXtreme Scale Container Server">

   <featureManager>
   <feature>eXtremeScale.server-1.0</feature>
   </featureManager>


   </server>
   ```

3. (Optional) To configure security for your clients, then use the client.xml file to specify the path to the server properties file, which contains all of the security settings. For more information, see Configuring client security on a catalog service domain.

You configured the Liberty profile by adding the client feature to the Liberty server.

Module 5: Run the data grid inside the Liberty profile

8.5+ After you add the client and server configurations to the Liberty profile, you can run WebSphere® eXtreme Scale in the Liberty profile.

Learning objectives

After completing the lessons in this module, you will know how to complete the following tasks:

- Configure eXtreme Scale servers to use the Liberty profile.
- Configure a Liberty profile web application server to use eXtreme Scale for session replication..

Prerequisites

To complete this module, you must complete the following modules in this tutorial:

- Install the Liberty profile.
- Install WebSphere eXtreme Scale.
- Create a web application server in Liberty.
- Add the Liberty profile web feature to the web application.
- Configure clients to use client APIs in the Liberty profile.

Lessons in this module

- **Lesson 5.1: Configure eXtreme Scale servers to use the Liberty profile**

  To run the data grid in a Liberty profile, you must add the server feature to configure WebSphere eXtreme Scale servers that use Liberty profile configuration files.
- **Lesson 5.2: Configuring a Liberty profile web application server to use eXtreme Scale for session replication**

  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 the Liberty profile.

< Previous | Next >
< Previous | Next >

Lesson 5.1: Configure eXtreme Scale servers to use the Liberty profile

8.5+ To run the data grid in a Liberty profile, you must add the server feature to configure WebSphere® eXtreme Scale servers that use Liberty profile configuration files.

1. Configure a catalog server with default settings using the following attributes in the server.xml file, which tells eXtreme Scale to create and start a catalog server:

```
<server description="eXtreme Scale Catalog Server with default settings">

    <!-- Enable features -->
    <featureManager>
        <feature>eXtremeScale.server-1.1</feature>
    </featureManager>

    <xsServer isCatalog="true" listenerPort="${com.ibm.ws.xs.server.listenerPort}" />

    <logging traceSpecification="*=info" maxFileSize="200" maxFiles="10" />

</server>
```

   Notice that the listenerPort element is referenced in the server.xml; however, you configure this value in the bootstrap.properties file. It can be useful to separate elements such as port numbers out of the server.xml file so that multiple processes that run with an identical configuration can share the server.xml file, but still have unique settings.

2. Configure the listenerPort attribute in the bootstrap.properties file.
   In the previous example, tracing is specified in the Liberty profile configuration, and the listenerPort attribute specifies a variable. This variable is configured in the bootstrap.properties file in the server configuration directory, *wlp_install_root*/usr/server/serverName. See the following example of the bootstrap.properties file:

```
# Licensed Materials - Property of IBM
#
# "Restricted Materials of IBM"
#
# Copyright IBM Corp. 2011 All Rights Reserved.
#
```

```
# US Government Users Restricted Rights - Use, duplication or
# disclosure restricted by GSA ADP Schedule Contract with
# IBM Corp.
#
# ------------------------------------------------------------------
#
# port for the OSGi console
# osgi.console=5678

com.ibm.ws.xs.server.listenerPort=2809
```

In this example, the osgi.console port is commented out, which means that the Liberty profile listens on the specified port for telnet clients to connect to an OSGi console. This behavior is useful for diagnosing OSGi-related errors.

3. 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 com.ibm.ws.xs.server.config element. See the following example from the server.xml file:

```
<server>
...
<com.ibm.ws.xs.server.config ... serverProps="/path/to/myServerProps.properties" ... />
</server>
```

Restriction: The Liberty configuration model has restrictions in the way properties are specified. Therefore, if you require the following properties, you must specify them in the server properties file:

foreignDomain.endpoints
: Specifies the names of catalog service domains to which you want to link in the multimaster replication topology.

xioChannel.xioContainerTCPNonSecure.Port
: Specifies the unsecured listener port number of eXtremeIO on the server. If you do not set the value, an ephemeral port is used. This property is used only when the transportType property is set to `TCP/IP. xioChannel.xioContainerTCPSecure.Port`.

Some properties that were formerly configurable in a stand-alone environment must be configured with the Liberty profile configuration instead of the eXtreme Scale configuration mechanisms.

- Logging and tracing settings must be specified with the logging element in the server.xml file, rather than being specified in the eXtreme Scale server properties file or com.ibm.ws.xs.server.config element. For more information, see Liberty profile: Trace and logging in the WebSphere Application Server Information Center.
- The working directory, like logging and tracing, is a server-wide setting, and therefore, they must be specified in a server-wide way.

If the previous settings are specified incorrectly, eXtreme Scale logs a warning message, which indicates that the settings are ignored.

4. (Optional) To configure security with your servers, then use the server.xml file to specify the path to the server properties file, which contains all of the security settings. When WebSphere eXtreme Scale is deployed in a WebSphere Application Server environment, you can simplify the authentication flow and transport layer security configuration from WebSphere Application Server. For more information, see Security integration with WebSphere Application Server.

Your eXtreme Scale servers are ready to run in the Liberty profile.

< Previous | Next >
< Previous

Lesson 5.2: Configuring a Liberty profile web application server to use eXtreme Scale for session replication

8.5+ 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 the Liberty profile.

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 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.

1. 8.5 Enable the HTTP session feature in the Liberty profile.
2. Configure a unique clone ID in the Liberty server.xml file.
3. Generate and merge plug-in configuration files for deployment to the application server plug-in.

Your eXtreme Scale applications that run in the Liberty profile are enabled for session replication.

< Previous
< Previous | Next >

Tutorial: Integrate WebSphere eXtreme Scale security with WebSphere Application Server

This tutorial demonstrates how to secure a WebSphere® eXtreme Scale server deployment in a WebSphere Application Server environment.

Learning objectives

The learning objectives for this tutorial follow:

- Configure WebSphere eXtreme Scale to use WebSphere Application Server authentication plug-ins
- Configure WebSphere eXtreme Scale transport security to use WebSphere Application Server CSIv2 configuration
- Use Java™ Authentication and Authorization Service (JAAS) authorization in WebSphere Application Server
- Use a custom login module for group-based JAAS authorization
- Use WebSphere eXtreme Scale **xscmd** utility in WebSphere Application Server environment

Time required

This tutorial takes approximately 4 hours from start to finish.
< Previous | Next >
< Previous | Next >

Introduction: Integrate WebSphere eXtreme Scale security with WebSphere Application Server using the WebSphere Application Server Authentication plug-ins

In this tutorial, you integrate WebSphere® eXtreme Scale security with WebSphere Application Server. First, you configure authentication with a simple web application that uses authenticated user credentials from the current thread to connect to the ObjectGrid. Then, you investigate the encryption of data that is transferred between the client and server with transport layer security. To give users varying levels of permissions, you can configure Java Authentication and Authorization Service (JAAS). After completing the configuration, you can use the **xscmd** utility to monitor your data grids and maps.

This tutorial assumes that all of your WebSphere eXtreme Scale clients, container servers, and catalog servers are deployed in the WebSphere Application Server environment.

Learning objectives

The learning objectives for this tutorial follow:

- Configure WebSphere eXtreme Scale to use WebSphere Application Server authentication plug-ins
- Configure WebSphere eXtreme Scale transport security to use WebSphere Application Server CSIv2 configuration
- Use Java™ Authentication and Authorization Service (JAAS) authorization in WebSphere Application Server
- Use a custom login module for group-based JAAS authorization
- Use WebSphere eXtreme Scale **xscmd** utility in WebSphere Application Server environment

Time required

This tutorial takes approximately 4 hours from start to finish.

Skill level

Intermediate.

Audience

Developers and administrators that are interested in the security integration between WebSphere eXtreme Scale and WebSphere Application Server.

System requirements and topology

- WebSphere Application Server Version 6.1 or Version 7.0.0.11 or later
- Update the Java runtime to apply the following fix: IZ79819: IBMJDK FAILS TO READ PRINCIPAL STATEMENT WITH WHITESPACE FROM SECURITY FILE

This tutorial uses four WebSphere Application Server application servers and one deployment manager to demonstrate the sample.

Prerequisites

A basic understanding of the following items is helpful before you start this tutorial:

- WebSphere eXtreme Scale programming model
- Basic WebSphere eXtreme Scale security concepts
- Basic WebSphere Application Server security concepts

For a background information about WebSphere eXtreme Scale and WebSphere Application Server security integration, see Security integration with WebSphere Application Server.

Modules in this tutorial

- **Module 1: Prepare WebSphere Application Server**

  Before you start the tutorial to integrate with WebSphere eXtreme Scale, you must create a basic security configuration in WebSphere Application Server.
- **Module 2: Configure WebSphere eXtreme Scale to use WebSphere Application Server Authentication plug-ins**

  After you have created the WebSphere Application Server configuration, you can integrate WebSphere eXtreme Scale authentication with WebSphere Application Server.
- **Module 3: Configure transport security**

  Configure transport security to secure data transfer between the clients and servers in the configuration.
- **Module 4: Use Java Authentication and Authorization Service (JAAS) authorization in WebSphere Application Server**

  Now that you have configured authentication for clients, you can further configure authentication to give different users varying permissions. For example, an operator user might only be able to view data, while an administrator user can perform all operations.

Related concepts:

Security overview
Related information:

WebSphere Application Server: Securing applications and their environment
< Previous | Next >
< Previous | Next >

Module 1: Prepare WebSphere Application Server

Before you start the tutorial to integrate with WebSphere® eXtreme Scale, you must create a basic security configuration in WebSphere Application Server.

Learning objectives

With the lessons in this module, you learn how to:

- Configure WebSphere Application Server security to use an internal file-based federated repository as a user account registry.
- Create user groups and users.
- Create clusters for the application and WebSphere eXtreme Scale servers.

Time required

This module takes approximately 60 minutes.

Lessons in this module

- **Lesson 1.1: Understand the topology and get the tutorial files**

  To prepare your environment for the tutorial, you must configure WebSphere Application Server security. You configure administration and application security using internal file-based federated repositories as a user account registry.
- **Lesson 1.2: Configure the WebSphere Application Server environment**

  To prepare your environment for the tutorial, you must configure WebSphere Application Server security. Enable administration and application security using internal file-based federated repositories as a user account registry. Then, you can create server clusters to host the client application and container servers.

< Previous | Next >
< Previous | Next >

Lesson 1.1: Understand the topology and get the tutorial files

To prepare your environment for the tutorial, you must configure WebSphere® Application Server security. You configure administration and application security using internal file-based federated repositories as a user account registry.

This lesson guides you through the sample topology and applications that are used to in the tutorial. To begin running the tutorial, you must download the applications and place the configuration files in the correct locations for your environment. You can download the sample application from the WebSphere eXtreme Scale wiki.

WebSphere Application Server sample topology

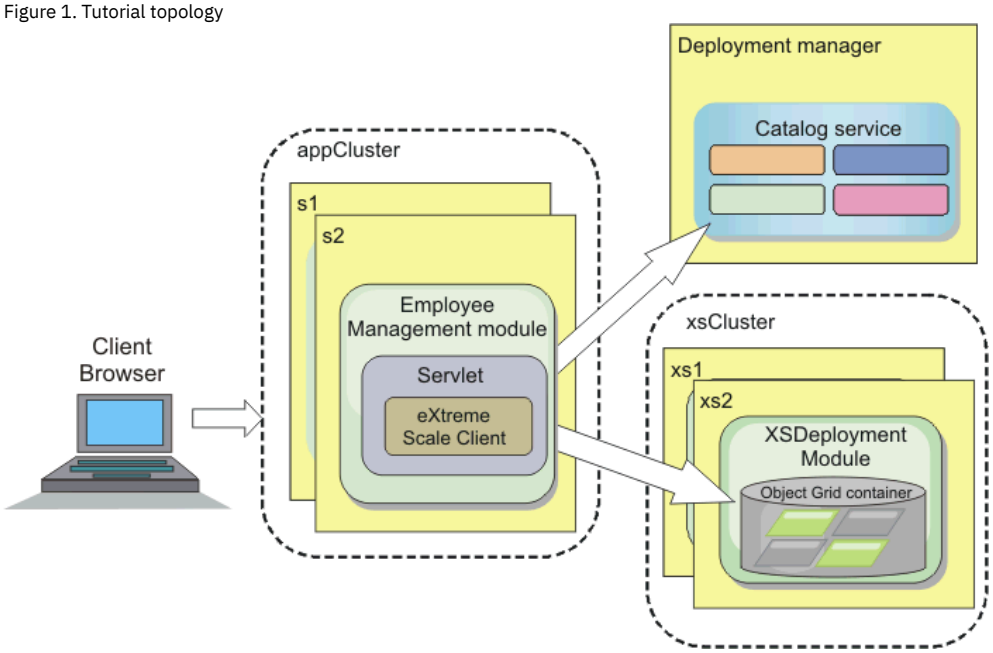
This tutorial guides you through creating four WebSphere Application Server application servers to demonstrate using the sample applications with security enabled. These application servers are grouped into two clusters, each with two servers:

- **appCluster cluster**: Hosts the EmployeeManagement sample enterprise application. This cluster has two application servers: s1 and s2.
- **xsCluster cluster**: Hosts the eXtreme Scale container servers. This cluster has two application servers: xs1 and xs2.

In this deployment topology, the s1 and s2 application servers are the client servers that access data that is being stored in the data grid. The xs1 and xs2 servers are the container servers that host the data grid.
The catalog server is deployed in the deployment manager process by default. This tutorial uses the default behavior. Hosting the catalog server in the deployment manager is not a recommended practice in a production environment. In a production environment, you should create a catalog service domain to define where catalog servers start. See Creating catalog service domains in WebSphere Application Server for more information.

Alternative configuration: You can host all of the application servers in a single cluster, such as in the appCluster cluster. With this configuration, all of the servers in the cluster are both clients and container servers. This tutorial uses two clusters to distinguish between the application servers that are hosting the clients and container servers.

Figure 1. Tutorial topology



Applications

In this tutorial, you are using two applications and one shared library file:

- **EmployeeManagement.ear**: The EmployeeManagement.ear application is a simplified Java™ 2 Platform, Enterprise Edition (J2EE) enterprise application. It contains a web module to manage the employee profiles. The web module contains the management.jsp file to display, insert, update, and delete employee profiles that are stored in the container servers.
- **XSDeployment.ear**: This application contains an enterprise application module with no application artifacts. The cache objects are packaged in the EmployeeData.jar file. The EmployeeData.jar file is deployed as a shared library for the XSDeployment.ear file, so that the XSDeployment.ear file can access the classes. The purpose of this application is to package the eXtreme Scale configuration files. When this enterprise application is started, the eXtreme Scale configuration files are automatically detected by the eXtreme Scale run time, so the container servers are created. These configuration files include the objectGrid.xml and objectGridDeployment.xml files.
- **EmployeeData.jar**: This jar file contains one class: the com.ibm.websphere.sample.xs.data.EmployeeData class. This class represents employee data that is stored in the grid. This Java archive (JAR) file is deployed with the EmployeeManagement.ear and XSDeployment.ear files as a shared library.

Get the tutorial files

1. Download the WASSecurity.zip and security.zip files. You can download the sample application from the WebSphere eXtreme Scale wiki.
2. Extract the WASSecurity.zip file to a directory for viewing the binary and source artifacts, for example the /wxs_samples/ directory. This directory is referred to as *samples_home* for the remainder of the tutorial. For a description of the contents of the WASSecurity.zip file and how to load the source into your Eclipse workspace, see the README.txt file in the package.
3. Extract the security.zip file to the *samples_home* directory. The security.zip file contains the following security configuration files that are used in this tutorial:
   - catServer2.props
   - server2.props
   - client2.props
   - securityWAS2.xml
   - xsAuth2.props

About the configuration files

The objectGrid.xml and objectGridDeployment.xml files create the data grids and maps that store the application data.

These configuration files must be named objectGrid.xml and objectGridDeployment.xml. When the application server starts, eXtreme Scale detects these files in the META-INF directory of the EJB and web modules. If these files are found, it assumed that the Java virtual machine (JVM) acts as a container server for the

defined data grids in the configuration files.

objectGrid.xml file

The objectGrid.xml file defined one ObjectGrid named `Grid`. The `Grid` data grid has one map, the `Map1` map, that stores the employee profile for the application.

```
<?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="Grid" txTimeout="15">
            <backingMap name="Map1" />
        </objectGrid>
    </objectGrids>

</objectGridConfig>
```

objectGridDeployment.xml file

The objectGridDeployment.xml file specifies how to deploy the `Grid` data grid. When the grid is deployed, it has five partitions and one synchronous replica.

```
<?xml version="1.0" encoding="UTF-8"?>

<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy ../deploymentPolicy.xsd"
 xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">

    <objectgridDeployment objectgridName="Grid">
        <mapSet name="mapSet" numberOfPartitions="5" minSyncReplicas="0" maxSyncReplicas="1" >
            <map ref="Map1"/>
        </mapSet>
    </objectgridDeployment>

</deploymentPolicy>
```

Lesson checkpoint

In this lesson, you learned about the topology for the tutorial and added the configuration files and sample applications to your environment.

If you want to learn more about automatically starting container servers, see Configuring WebSphere Application Server applications to automatically start container servers.

< Previous | Next >

< Previous | Next >

Lesson 1.2: Configure the WebSphere Application Server environment

To prepare your environment for the tutorial, you must configure WebSphere® Application Server security. Enable administration and application security using internal file-based federated repositories as a user account registry. Then, you can create server clusters to host the client application and container servers.

The following steps were written using WebSphere Application Server Version 7.0. However, you can also apply the concepts apply to earlier versions of WebSphere Application Server.

Configure WebSphere Application Server security

1. Configure WebSphere Application Server security.
    a. In the WebSphere Application Server administrative console, click Security > Global Security.
    b. Select Federated repositories as the Available realm definition. Click Set as current.
    c. Click Configure.. to go to the Federated repositories panel.
    d. Enter the Primary administrative user name, for example, `admin`. Click Apply.
    e. When prompted, enter the administrative user password and click OK. Save your changes.
    f. On the Global Security page, verify that Federated repositories setting is set to the current user account registry.
    g. Select the following items: Enable administrative security, Enable application security, and Use Java 2 security to restrict application access to local resources. Click Apply and save your changes.
    h. Restart the deployment manager and any running application servers.

   The WebSphere Application Server administrative security is enabled using the internal file-based federated repositories as the user account registry.
2. Create two user groups: adminGroup and operatorGroup.
    a. Click Users and groups > Manage groups > Create...
    b. Type `adminGroup` as the group name. Enter `Administration group` as the description. Click Create.
    c. Click Create alike. Type `operatorGroup` as the group name. Enter `Operator group` as the description. Click Create.
    d. Click Close.
3. Create users `admin1` and `operator1`.
    a. Click Users and groups > Manage users > Create...
    b. Create a user called `admin1` with the first name `Joe` and last name `Doe` with the password `admin1`. Click Create.
    c. Create a second user. Click Create alike to create a a user called `operator1` with the first name `Jane` and last name `Doe` with the password `operator1`. Click Create. Click Close.
4. Add users to the user groups. Add the `admin1` user to the `adminGroup` and the `operator1` user to the `operatorGroup`.

a. Click Users and groups > Manage users.
b. Search for users to add to groups. Click Search.. and set the search for value to an asterisk (*) to display all the users.
c. From the search result, click the `admin1` user, and click the Groups tab. Click Add to add the group.
d. Search the groups to find the available groups. Click the `adminGroup` and click Add.
e. Repeat these steps to add the `operator1` user to the `operatorGroup` user group.

5. Save your changes, log out of the administrative console, and restart the deployment manager and node agent to enable the security settings.

You enabled security and created users and user groups have administrative and operator access to your WebSphere Application Server configuration.

Create server clusters

Create two server clusters in your WebSphere Application Server configuration: The `appCluster` cluster to host the sample application for the tutorial and the `xsCluster` cluster to host the data grid.

1. In the WebSphere Application Server administrative console, open the clusters panel. Click Servers > Clusters > WebSphere application server clusters > New.
2. Type `appCluster` as the cluster name, leave the Prefer local option selected, and click Next.
3. Create servers in the cluster. Create a server named `s1`, keeping the default options. Add an additional cluster member named `s2`.
4. Complete the remaining steps in the wizard to create the cluster. Save the changes.
5. Repeat these steps to create the `xsCluster` cluster. This cluster has two servers, named `xs1` and `xs2`.

Lesson checkpoint

You enabled global security for the WebSphere Application Server cell, created users and user groups, and created clusters to host the application and data grid.

< Previous | Next >
< Previous | Next >

Module 2: Configure WebSphere eXtreme Scale to use WebSphere Application Server Authentication plug-ins

After you have created the WebSphere® Application Server configuration, you can integrate WebSphere eXtreme Scale authentication with WebSphere Application Server.

When a WebSphere eXtreme Scale client connects to a container server that requires authentication, the client must provide a credential generator represented by the com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator interface. A credential generator is a factory to create a client credential. A client credential can be: a user name and password pair, a Kerberos ticket, a client certificate, or client identification data in any format that the client and server agree upon. See the Credential API documentation for more details. In this sample, the WebSphere eXtreme Scale client is the EmployeeManagment web application that is deployed in the appCluster cluster. The client credential is a WebSphere security token that represents the web user identity.

Learning objectives

With the lessons in this module, you learn how to:

- Configure client server security.
- Configure catalog server security.
- Configure container server security.
- Install and run the sample application.

Time required

This module takes approximately 60 minutes.

Lessons in this module

- **Lesson 2.1: Configure client server security**

  The client properties file indicates the CredentialGenerator implementation class to use.
- **Lesson 2.2: Configure catalog server security**

  A catalog server contains two different levels of security information: The security properties that are common to all the WebSphere eXtreme Scale servers, including the catalog service and container servers, and the security properties that are specific to the catalog server.
- **Lesson 2.3: Configure container server security**

  When a container server connects to the catalog service, the container server gets all the security configurations that are configured in the Object Grid Security XML file, such as authenticator configuration, the login session timeout value, and other configuration information. A container server also has its own server-specific security properties in the server property file.
- **Lesson 2.4: Install and run the sample**

  After authentication is configured, you can install and run the sample application.

Related reference:

Client properties file
Server properties file
Related information:

Lesson 2.1: Configure client server security
Credential API documentation
Lesson 2.2: Configure catalog server security
< Previous | Next >
< Previous | Next >

Lesson 2.1: Configure client server security

The client properties file indicates the CredentialGenerator implementation class to use.

Configure the client properties file with the -Dobjectgrid.client.props JVM property. The file name specified for this property is an absolute file path, such as *samples_home*/security/client2.props. See Client properties file for more information about the client properties file.
Related reference:

Client properties file
Related information:

Module 2: Configure WebSphere eXtreme Scale to use WebSphere Application Server Authentication plug-ins
Credential API documentation

Client properties file contents

This example uses WebSphere Application Server security tokens as the client credential. The client2.props file is in the *samples_home*/security directory. The client2.props file includes the following settings:

securityEnabled
: When set to `true`, indicates that the client must send available security information to the server.

credentialAuthentication
: When set to `Supported`, indicates that the client supports credential authentication.

credentialGeneratorClass
: Indicates the com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredentialGenerator class so the client retrieves the security tokens from the thread. See Security integration with WebSphere Application Server for more information about how security tokens are retrieved.

Setting the client properties file using Java™ virtual machine (JVM) properties

In the administrative console, complete the following steps to both the `s1` and `s2` servers in the `appCluster` cluster. If you are using a different topology, complete the following steps to all of the application servers to which the `EmployeeManagement` application will be deployed.

1. Servers > WebSphere application servers > *server_name* > Java and Process Management > Process definition > Java Virtual Machine.
2. Create the following generic JVM property to set the location of the client properties file:

   ```
   -Dobjectgrid.client.props=samples_home/security/client2.props
   ```

3. Click OK and save your changes.

Lesson checkpoint

You edited the client properties file and configured the servers in the `appCluster` cluster to use the client properties file. This properties file indicates the CredentialGenerator implementation class to use.

< Previous | Next >
< Previous | Next >

Lesson 2.2: Configure catalog server security

A catalog server contains two different levels of security information: The security properties that are common to all the WebSphere® eXtreme Scale servers, including the catalog service and container servers, and the security properties that are specific to the catalog server.

The security properties that are common to the catalog servers and container servers are configured in the security XML descriptor file. An example of common properties is the authenticator configuration, which represents the user registry and authentication mechanism. See Security descriptor XML file for more information about the security properties.
To configure the security XML descriptor file, create a -Dobjectgrid.cluster.security.xml.url property in the Java™ virtual machine (JVM) argument. The file name specified for this property is in an URL format, such as `file:///samples_home/security/securityWAS2.xml`.

Related reference:

Server properties file
Related information:

Module 2: Configure WebSphere eXtreme Scale to use WebSphere Application Server Authentication plug-ins

securityWAS2.xml file

In this tutorial, the securityWAS2.xml file is in the *samples_home*/security directory. The content of the securityWAS2.xml file with the comments removed follows:

```
<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 following properties are defined in the securityWAS2.xml file:

securityEnabled
: The securityEnabled property is set to `true`, which indicates to the catalog server that the WebSphere eXtreme Scale global security is enabled.

authenticator
: The authenticator is configured as the com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenAuthenticator class. With this built-in implementation of the Authenticator plug-in, the WebSphere eXtreme Scale server can convert the security tokens to a Subject object. See Security integration with WebSphere Application Server for more information about how the security tokens are converted.

catServer2.props file

The server property file stores the server-specific properties, which include the server-specific security properties. See Server properties file for more information. You can configure the server property file with the `-Dobjectgrid.server.props` property in the JVM argument. Specify the file name value for this property is an absolute path, such as *samples_home*`/security/catServer2.props`. For this tutorial, a catServer2.props file is included in the *samples_home*/security directory. The content of the catServer2.props file with comments removed follows:

securityEnabled
: The securityEnabled property is set to `true` to indicate that this catalog server is a secure server.

credentialAuthentication
: The credentialAuthentication property is set to `Required`, so any client that is connecting to the server is required to provide a credential.

secureTokenManagerType
: The secureTokenManagerType is set to `none` to indicate that the authentication secret is not encrypted when joining the existing servers.

authenticationSecret
: The authenticationSecret property is set to `ObjectGridDefaultSecret`. This secret string is used to join the eXtreme Scale server cluster. When a server joins the data grid, it is challenged to present the secret string. If the secret string of the joining server matches the string in the catalog server, the joining server is accepted. If the string does not match, the join request is rejected.

transportType
: The transportType property is set to `TCP/IP` initially. Later in the tutorial, transport security is enabled.

Setting the server properties file with JVM properties

Set the server properties file on the deployment manager server. If you are using a different topology than the topology for this tutorial, set the server properties file on all of the application servers that you are using to host catalog servers.

1. Open the Java virtual machine configuration for the server. In the administrative console, click System administration > Deployment manager > Java and Process Management > Process definition > Java Virtual Machine.
2. Add the following generic JVM arguments:

   ```
   -Dobjectgrid.cluster.security.xml.url=file:///samples_home/security/securityWAS2.xml
   -Dobjectgrid.server.props=samples_home/security/catServer2.props
   ```

3. Click OK and save your changes.

Lesson checkpoint

You configured catalog server security by associating the securityWAS2.xml and catServer2.props files with the deployment manager, which hosts the catalog server process in the WebSphere Application Server configuration.

< Previous | Next >
< Previous | Next >

Lesson 2.3: Configure container server security

When a container server connects to the catalog service, the container server gets all the security configurations that are configured in the Object Grid Security XML file, such as authenticator configuration, the login session timeout value, and other configuration information. A container server also has its own server-specific security properties in the server property file.

Configure the server property file with the `-Dobjectgrid.server.props` Java virtual machine (JVM) property. The file name for this property is an absolute file path, such as *samples_home*/security/server2.props.
In this tutorial, the container servers are hosted in the `xs1` and `xs2` servers in the `xsCluster` cluster.

server2.props file

The server2.props file is in the *samples_home*/security directory under the WASSecurity directory. The properties that are defined in the server2.props file follow:

securityEnabled
: The securityEnabled property is set to `true` to indicate that this container server is a secure server.

credentialAuthentication
: The credentialAuthentication property is set to `Required`, so any client that is connecting to the server is required to provide a credential.

secureTokenManagerType
: The secureTokenManagerType is set to `none` to indicate that the authentication secret is not encrypted when joining the existing servers.

authenticationSecret
: The authenticationSecret property is set to `ObjectGridDefaultSecret`. This secret string is used to join the eXtreme Scale server cluster. When a server joins the data grid, it is challenged to present the secret string. If the secret string of the joining server matches the string in the catalog server, the joining server is accepted. If the string does not match, the join request is rejected.

Setting the server properties file with JVM properties

Set the server properties file on the xs1 and xs2 servers. If you are not using the topology for this tutorial, set the server properties file on all of the application servers that you are using to host container servers.

1. Open the Java™ virtual machine page for the server. Servers > Application servers > *server_name* > Java and Process Management > Process definition > Java Virtual Machine
2. Add the generic JVM arguments:

   `-Dobjectgrid.server.props=`*`samples_home`*`/security/server2.props`

3. Click OK and save your changes.

Lesson checkpoint

Now the WebSphere eXtreme Scale server authentication is secured. By configuring this security, all the applications that try to connect to the WebSphere eXtreme Scale servers are required to provide a credential. In this tutorial, the WSTokenAuthenticator is the authenticator. As a result, the client is required to provide a WebSphere Application Server security token.

< Previous | Next >
< Previous | Next >

Lesson 2.4: Install and run the sample

After authentication is configured, you can install and run the sample application.

Creating a shared library for the EmployeeData.jar file

1. In the WebSphere Application Server administrative console, open the Shared Libraries page. Click Environment > Shared libraries.
2. Choose the cell scope.
3. Create the shared library. Click New. Enter `EmployeeManagementLIB` as the Name. Enter the path to the EmployeeData.jar in the classpath, for example, *samples_home*/WASSecurity/EmployeeData.jar.
4. Click Apply.

Installing the sample

1. Install the EmployeeManagement.ear file.
   a. To begin the installation, click Applications > New application > New Enterprise Application. Choose the detailed path for installing the application.
   b. On the Map modules to servers step, specify the `appCluster` cluster to install the EmployeeManagementWeb module.
   c. On the Map shared libraries step, select the `EmployeeManagementWeb` module.
   d. Click Reference shared libraries. Select the `EmployeeManagementLIB` library.
   e. Map the `webUser` role to All Authenticated in Application's Realm.
   f. Click OK.

   The clients run in the `s1` and `s2` servers in this cluster.
2. Install the sample XSDeployment.ear file.
   a. To begin the installation, click Applications > New application > New Enterprise Application. Choose the detailed path for installing the application.
   b. On the Map modules to servers step, specify the `xsCluster` cluster to install the `XSDeploymentWeb` web module.

c. On the Map shared libraries step, select the `XSDeploymentWeb` module.
d. Click Reference shared libraries. Select the `EmployeeManagementLIB` library.
e. Click OK.

The `xs1` and `xs2` servers in this cluster host the container servers.

3. Restart the deployment manager. When the deployment manager starts, the catalog server also starts. If you look at the SystemOut.log file of the deployment manager, you can see the following message that indicates that the eXtreme Scale server properties file is loaded.

```
CWOBJ0913I: Server property files have been loaded:
/wxs_samples/security/catServer2.props.
```

4. Restart the xsCluster cluster. When the xsCluster starts, the XSDeployment application starts, and a container server is started on the xs1 and xs2 servers respectively. If you look at the SystemOut.log file of the xs1 and xs2 servers, the following message that indicates the server properties file is loaded is displayed:

```
CWOBJ0913I: Server property files have been loaded:
/wxs_samples/security/server2.props.
```

5. Restart the appClusters cluster. When the cluster appCluster starts, the EmployeeManagement application also starts. If you look at the SystemOut.log file of the s1 and s2 servers, you can see the following message that indicates that the client properties file is loaded.

```
CWOBJ0924I: The client property file {0} has been loaded.
```

You can ignore the warning messages regarding the authenticationRetryCount, transportType, and clientCertificateAuthentication properties. The default values are be used because the values were not specified in the properties file.

If you are using WebSphere eXtreme Scale Version 7.0, the English-only CWOBJ9000I message displays to indicate that the client property file has been loaded. If you do not see the expected message, verify that you configured the -Dobjectgrid.server.props or -Dobjectgrid.client.props property in the JVM argument. If you do have the properties configured, make sure the dash (-) is a UTF character.

Running the sample application

1. Run the management.jsp file. In a web browser, access `http://<your_servername>:<port>/EmployeeManagementWeb/management.jsp`. For example, you might use the following URL: `http://localhost:9080/EmployeeManagementWeb/management.jsp`.
2. Provide authentication to the application. Enter the credentials of the user that you mapped to the webUser role. By default, this user role is mapped to all authenticated users. Type `admin1` as your user ID and `admin1` as your password. A page to display, add, update, and delete employees displays.
3. Display employees. Click Display an Employee. Enter `emp1@acme.com` as the email address, and click Submit. A message displays that the employee cannot be found.
4. Add an employee. click Add an Employee. Enter `emp1@acme.com` as the email address, enter `Joe` as the first name, and `Doe` as the last name. Click Submit. A message displays that an employee with the `emp1@acme.com` address has been added.
5. Display the new employee. Click Display an Employee. Enter `emp1@acme.com` as the email address with empty fields for the first and last names, and click Submit. A message displays that the employee has been found, and the correct names are displayed in the first name and last name fields.
6. Delete the employee. Click Delete an employee. Enter `emp1@acme.com` and click Submit. A message is displayed that the employee has been deleted.

Lesson checkpoint

You installed and ran the sample application. Because this tutorial uses WebSphere Application Server integration, you cannot see the scenario when a client fails to authenticate to the eXtreme Scale server. If the user authenticates to the WebSphere Application Server successfully, eXtreme Scale is also successfully authenticated.

< Previous | Next >
< Previous | Next >

Module 3: Configure transport security

Configure transport security to secure data transfer between the clients and servers in the configuration.

In the previous module in the tutorial, you enabled WebSphere® eXtreme Scale authentication. With authentication, any application that tries to connect to the WebSphere eXtreme Scale server is required to provide a credential. Therefore, no unauthenticated client can connect to the WebSphere eXtreme Scale server. The clients must be an authenticated application that is running in a WebSphere Application Server cell.

With the configuration up to this module, the data transfer between the clients in the appCluster cluster and servers in the xsCluster cluster is not encrypted. This configuration might be acceptable if your WebSphere Application Server clusters are installed on servers behind a firewall. However, in some scenarios, non-encrypted traffic is not accepted for some reasons even though the topology is protected by firewall. For example, a government policy might enforce encrypted traffic. WebSphere eXtreme Scale supports Transport Layer Security/Secure Sockets Layer (TLS/SSL) for secure communication between ObjectGrid endpoints, which include client servers, container servers, and catalog servers.

In this sample deployment, the eXtreme Scale clients and container servers are all running in the WebSphere Application Server environment. Client or server properties are not necessary to configure the SSL settings because the eXtreme Scale transport security is managed by the Application Server Common Secure Interoperability Protocol Version 2 (CSIV2) transport settings. WebSphere eXtreme Scale servers use the same Object Request Broker (ORB) instance as the application servers in which they run. Specify all the SSL settings for client and container servers in the WebSphere Application Server configuration using these CSIv2 transport settings. The catalog server has its own proprietary transport paths that do not use Internet Inter-ORB Protocol (IIOP) or Remote Method Invocation (RMI). Because of these proprietary transport paths, the catalog server cannot be managed by the WebSphere Application Server CSIV2 transport settings. Therefore, you must configure the SSL properties in the server properties file for the catalog server.

Learning objectives

After completing the lessons in this module, you know how to:

- Configure CSIv2 inbound and outbound transport.
- Add SSL properties to the catalog server properties file.
- Check the ORB properties file.
- Run the sample.

Time required

This module takes approximately 60 minutes.

Prerequisites

This step of the tutorial builds upon the previous modules. Complete the previous modules in this tutorial before you configure transport security.

Lessons in this module

- **Lesson 3.1: Configure CSIv2 inbound and outbound transport**

  To configure Transport Layer Security/Secure Sockets Layer (TLS/SSL) for the server transport, set the Common Secure Interoperability Protocol Version 2 (CSIv2) inbound transport and CSIv2 outbound transport to `SSL-Required` for all the WebSphere Application Server servers that host clients, catalog servers, and container servers.
- **Lesson 3.2: Add SSL properties to the catalog server properties file**

  The catalog server 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 Secure Sockets Layer (SSL) properties in the server properties file for the catalog server.
- **Lesson 3.3: Run the sample**

  Restart all the servers and run the sample application again. You should be able to run through the steps without any problems.

< Previous | Next >
< Previous | Next >
< Previous | Next >

Lesson 3.1: Configure CSIv2 inbound and outbound transport

To configure Transport Layer Security/Secure Sockets Layer (TLS/SSL) for the server transport, set the Common Secure Interoperability Protocol Version 2 (CSIv2) inbound transport and CSIv2 outbound transport to `SSL-Required` for all the WebSphere® Application Server servers that host clients, catalog servers, and container servers.

In the tutorial example topology, you must set these properties for the, s1, s2, xs1, and xs2 application servers. The following steps configure the inbound and outbound transports for all the servers in the configuration.

Set the inbound and outbound transports in the administrative console. Make sure that administrative security is enabled.

- **WebSphere Application Server Version 6.1**: Click Security > Secure Administration > Application.. > RMI/IIOP Security and change the transport type to SSL-Required.
- **WebSphere Application Server Version 7.0**: Click Security > Global Security > RMI/IIOP Security > CSIv2 inbound communications. Change the transport type under the CSIv2 Transport Layer to SSL-Required. Repeat this step to configure CSIv2 outbound communications.

You can use centrally managed endpoint security settings, or you can configure SSL repositories. See Common Secure Interoperability Version 2 transport inbound settings for more information.

< Previous | Next >
< Previous | Next >
< Previous | Next >

Lesson 3.2: Add SSL properties to the catalog server properties file

The catalog server 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 Secure Sockets Layer (SSL) properties in the server properties file for the catalog server.

To configure catalog server security, additional steps are necessary because the catalog server has its own proprietary transport paths. These transport paths cannot be managed by the Application Server CSIV2 transport settings.

1. Edit the SSL properties in the catServer2.props file. To configure catalog server security, uncomment the following SSL properties in the catalog server properties file. For this tutorial, the catalog server properties are in the catServer2.props file. Update the keyStore and trustStore properties to refer to the

proper location in your environment.

```
#alias=default
#contextProvider=IBMJSSE2
#protocol=SSL
#keyStoreType=PKCS12
#keyStore=/<WAS_HOME>/IBM/WebSphere/AppServer/profiles/<DMGR_NAME>/config/
cells/<CELL_NAME>/nodes/<NODE_NAME>/key.p12
#keyStorePassword=WebAS
#trustStoreType=PKCS12
#trustStore=/<WAS_HOME>/IBM/WebSphere/AppServer/profiles/<DMGR_NAME>/config/
cells/<CELL_NAME>/nodes/<NODE_NAME>/trust.p12
#trustStorePassword=WebAS
#clientAuthentication=false
```

The catServer2.props file is using the default WebSphere Application Server node level keystore and truststore. If you are deploying a more complex deployment environment, you must choose the correct keystore and truststore. In some cases, you must create a keystore and truststore and import the keys from keystores from the other servers. Notice that the `WebAS` string is the default password of the WebSphere Application Server keystore and truststore. See Default self-signed certificate configuration for more details.

2. In the catServer2.props file, update the value of the transportType property. For previous steps of the tutorial, the value was set to `TCP/IP`. Change the value to `SSL-Required`.
3. Restart the deployment manager to activate the changes to the catalog server security settings.

Lesson checkpoint

You configured the SSL properties for the catalog server.

< Previous | Next >
< Previous | Next >

Lesson 3.3: Run the sample

Restart all the servers and run the sample application again. You should be able to run through the steps without any problems.

See Lesson 2.4: Install and run the sample for more information about running and installing the sample application.

Lesson checkpoint

You ran the sample application with transport security enabled.

< Previous | Next >
< Previous | Next >

Module 4: Use Java Authentication and Authorization Service (JAAS) authorization in WebSphere Application Server

Now that you have configured authentication for clients, you can further configure authentication to give different users varying permissions. For example, an operator user might only be able to view data, while an administrator user can perform all operations.

After authenticating a client, as in the previous module in this tutorial, you can give security privileges through eXtreme Scale authorization mechanisms. The previous module of this tutorial demonstrated how to enable authentication for a data grid using integration with WebSphere® Application Server. As a result, no unauthenticated client can connect to the eXtreme Scale servers or submit requests to your system. However, every authenticated client has the same permission or privileges to the server, such as reading, writing, or deleting data that is stored in the ObjectGrid maps. Clients can also issue any type of query. This part of the tutorial demonstrates how to use eXtreme Scale authorization to give authenticated users varying privileges. WebSphere eXtreme Scale uses a permission-based authorization mechanism. You can assign different permission categories that are represented by different permission classes. This module features the MapPermission class. For a list of all possible permissions, see Client authorization programming.

In WebSphere eXtreme Scale, the com.ibm.websphere.objectgrid.security.MapPermission class represents permissions to the eXtreme Scale resources, specifically the methods of the ObjectMap or JavaMap interfaces. WebSphere eXtreme Scale defines the following permission strings to access the methods of ObjectMap and JavaMap:

- **read**: Grants permission to read the data from the map.
- **write**: Grants permission to update the data in the map.
- **insert**: Grants permission to insert the data into the map.
- **remove**: Grants permission to remove the data from the map.
- **invalidate**: Grants permission to invalidate the data from the map.
- **all**: Grants all permissions to read, write, insert, remote, and invalidate.

The authorization occurs when an eXtreme Scale client uses a data access API, such as the ObjectMap ,JavaMap, or EntityManager APIs. The run time checks corresponding map permissions when the method is called. If the required permissions are not granted to the client, an AccessControlException exception results. This tutorial demonstrates how to use Java Authentication and Authorization Service (JAAS) authorization to grant authorization map access for different users.

Learning objectives

After completing the lessons in this module, you know how to:

- Enable authorization for WebSphere eXtreme Scale.
- Enable user-based authorization.
- Configure group-based authorization.

Time required

This module takes approximately 60 minutes.

Prerequisites

You must complete the prior modules in this tutorial before configuring authentication.

Lessons in this module

- **Lesson 4.1: Enable WebSphere eXtreme Scale authorization**

  To enable authorization in WebSphere eXtreme Scale, you must enable security on a specific ObjectGrid.
- **Lesson 4.2: Enable user-based authorization**

  In the authentication module of this tutorial, you created two users: `operator1` and `admin1`. You can assign varying permissions to these users with Java Authentication and Authorization Service (JAAS) authorization.
- **Lesson 4.3: Configure group-based authorization**

  In the previous lesson, you assigned individual user-based authorization with user principals in the Java Authentication and Authorization Service. (JAAS) authorization policy. However, when you have hundreds or thousands of users, use group-based authorization, which authorizes access based on groups instead of individual users.

Related concepts:

Client authorization programming
< Previous | Next >
< Previous | Next >

Lesson 4.1: Enable WebSphere eXtreme Scale authorization

To enable authorization in WebSphere® eXtreme Scale, you must enable security on a specific ObjectGrid.

To enable authorization on the ObjectGrid, you must set the securityEnabled attribute to true for that particular ObjectGrid in the XML file. For this tutorial, you can either use the XSDeployment_sec.ear file in the *samples_home*/WASSecurity directory, which has already has security set in the objectGrid.xml file, or you can edit the existing objectGrid.xml file to enable security. This lesson demonstrates how to edit the file to enable security.

1. Extract the files in the XSDeployment.ear file, and then unzip the XSDeploymentWeb.war file.
2. Open the objectGrid.xml file and set the securityEnabled attribute to `true` on the ObjectGrid level. See an example of this attribute in the following example:

```
<?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="Grid" securityEnabled="true">
            <backingMap name="Map1" />
        </objectGrid>
    </objectGrids>

</objectGridConfig>
```

   If you have multiple ObjectGrids defined, then you must set this attribute on each data grid.
3. Repackage the XSDeploymentWeb.war and XSDeployment.ear files to include your changes. Name the file XSDeployment_sec.ear so you do not overwrite the original package.
4. Uninstall the existing XSDeployment application and install the XSDeployment_sec.ear file. See Lesson 2.4: Install and run the sample for more information about deploying applications.

Lesson checkpoint

You enabled security on the ObjectGrid, which also enables authorization on the data grid.

< Previous | Next >

Lesson 4.2: Enable user-based authorization

In the authentication module of this tutorial, you created two users: `operator1` and `admin1`. You can assign varying permissions to these users with Java™ Authentication and Authorization Service (JAAS) authorization.

Defining the Java Authentication and Authorization Service (JAAS) authorization policy using user principals

You can assign permissions to the users that you previously created. Assign the `operator1` user read permissions only to all maps. Assign the `admin1` user all permissions. Use the JAAS authorization policy file to grant permissions to principals.
Edit the JAAS authorization file. The xsAuth2.policy file is in the *samples_home*/security directory:

```
grant codebase http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction
Principal com.ibm.ws.security.common.auth.WSPrincipalImpl "defaultWIMFileBasedRealm/operator1" {
    permission com.ibm.websphere.objectgrid.security.MapPermission "Grid.Map1", "read";
};

grant codebase http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction
Principal com.ibm.ws.security.common.auth.WSPrincipalImpl "defaultWIMFileBasedRealm/admin1" {
    permission com.ibm.websphere.objectgrid.security.MapPermission "Grid.Map1", "all";
};
```

In this file, the `http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction` codebase is a specially reserved URL for ObjectGrid. All ObjectGrid permissions that are granted to principals should use this special code base. The following permissions are assigned in this file:

- The first grant statement grants `read` map permission to the `operator1` principal. The `operator1` user has only map read permission to the Map1 map the Grid ObjectGrid instance.
- The second grant statement grants all map permission to the admin1 principal. The `admin1` user has all permissions to the Map1 map in the Grid ObjectGrid instance.
- The principal name is `defaultWIMFileBasedRealm/operator1`, but not `Operator1`. WebSphere Application Server automatically adds the realm name to the principal name when federated repositories are used as the user account registry. Adjust this value if needed.

Setting the JAAS authorization policy file using JVM properties

Use the following steps to set JVM properties for the xs1 and xs2 servers, which are in the xsCluster cluster. If you are using a topology that is different from the sample topology that is used in this tutorial, set the file on all of your container servers.

1. In the administrative console, click Servers > Application servers > *server_name* > Java and Process Management > Process definition > Java Virtual Machine.
2. Add generic JVM arguments.
   Note: When containers are running in WebSphere Application Server, you cannot use the -Djava.security.policy argument because this file overrides the WebSphere Application Server administrative access authorization. Therefore, use -Djava.security.auth.policy to set the JAAS authorization policy.
   Enter the following generic JVM arguments or replace the `-Djava.security.auth.policy` entry with the following text:

   ```
   -Djava.security.auth.policy=samples_home/security/xsAuth2.policy
   ```

3. Click OK and save your changes.

Running the sample application to test authorization

You can use the sample application to test the authorization settings. The administrator user continues to have all permissions in the Map1 map, including displaying and adding employees. The operator user should only be able to view employees because that user was assigned read permission only.

1. Restart all of the application servers that are running container servers.
2. Open the `EmployeeManagementWeb` application. In a web browser, open `http://<host>:<port>/EmployeeManagementWeb/management.jsp`.
3. Log in to the application as an administrator. Use the user name `admin1` and password `admin1`.
4. Attempt to display an employee. Click Display an Employee and search for the `authemp1@acme.com` email address. A message displays that the user cannot be found.
5. Add an employee. Click Add an Employee. Add the email `authemp1@acme.com`, the first name `Joe`, and the last name `Doe`. Click Submit. A message displays that the employee has been added.
6. Log in as the operator user. Open a second Web browser window and open `http://<host>:<port>/EmployeeManagermentWeb/management.jsp`. Use the user name `operator1` and password `operator1`.
7. Attempt to display an employee. Click Display an Employee and search for the `authemp1@acme.com` email address. The employee is displayed.
8. Add an employee. Click Add an Employee. Add the email `authemp2@acme.com`, the first name `Joe`, and the last name `Doe`. Click Submit. The following message displays:

   ```
   An exception occurs when Add the employee. See below for detailed exception messages.
   ```

   The following exception is in the exception chain:

   ```
   java.security.AccessControlException: Access denied
   (com.ibm.websphere.objectgrid.security.MapPermission Grid.Map1 insert)
   ```

This message displays because the `operator1` user does not have permission to insert data into the Map1 map.

If you are running with a version of WebSphere Application Server that is earlier than Version 7.0.0.11, you might see a java.lang.StackOverflowError error on the container server. This error is caused by a problem with the IBM Developer Kit. The problem is fixed in the IBM Developer Kit that is shipped with WebSphere Application Server Version 7.0.0.11 and later.

Lesson checkpoint

In this lesson, you configured authorization by assigning permissions to specific users.

< Previous | Next >
< Previous | Next >

Lesson 4.3: Configure group-based authorization

In the previous lesson, you assigned individual user-based authorization with user principals in the Java™ Authentication and Authorization Service. (JAAS) authorization policy. However, when you have hundreds or thousands of users, use group-based authorization, which authorizes access based on groups instead of individual users.

Unfortunately, the Subject object that is authenticated from the WebSphere® Application Server only contains a user principal. This object does not contain a group principal. You can add a custom login module to populate the group principal into the Subject object.

For this tutorial, the custom login module is named com.ibm.websphere.samples.objectgrid.security.lm.WASAddGroupLoginModule. The module is in the groupLM.jar file. Place this JAR file in the WAS-INSTALL/lib/ext directory.

The WASAddGroupLoginModule retrieves the public group credential from the WebSphere Application Server subject and creates a Group principal, com.ibm.websphere.samples.objectgrid.security.WSGroupPrincipal, to represent the group. This group principal can then be used for group authorization. The groups are defined in the xsAuthGroup2.policy file:

```
grant codebase "http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction"
principal com.ibm.websphere.sample.xs.security.WSGroupPrincipal
                "defaultWIMFileBasedRealm/cn=operatorGroup,o=defaultWIMFileBasedRealm" {
    permission com.ibm.websphere.objectgrid.security.MapPermission "Grid.Map1", "read";
};

grant codebase "http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction"
principal com.ibm.websphere.sample.xs.security.WSGroupPrincipal
        "defaultWIMFileBasedRealm/cn=adminGroup,o=defaultWIMFileBasedRealm" {
    permission com.ibm.websphere.objectgrid.security.MapPermission "Grid.Map1", "all";
};
```

The principal name is the `WSGroupPrincipal`, which represents the group.

Adding the custom login module

The custom login module must be added to each of the following system login module entries: If you are using Lightweight Third Party Authentication (LTPA), add the entry to the RMI_INBOUND login modules. LTPA is the default authentication mechanism for WebSphere Application Server Version 7.0. For a WebSphere Application Server Network Deployment configuration, you only need to configure the LTPA authentication mechanism configuration entries.
Use the following steps to configure the supplied com.ibm.websphere.samples.objectgrid.security.lm.WASAddGroupLoginModule login module:

1. In the administrative console, click Security > Global Security > Java Authentication and Authorization Service > System logins > *login_module_name* > JAAS login modules > New.
2. Enter the class name as `com.ibm.websphere.sample.xs.security.lm.WASAddGroupLoginModule`.
3. Optional: Add a property `debug` and set the value to `true`.
4. Click Apply to add the new module to the login module list.

Setting the JAAS Authorization Policy file using JVM Properties

In the administrative console, perform the following steps to xs1 and xs2 servers in the xsCluster. If a different deployment topology is used, perform the following steps to the application servers that host the container servers.

1. In the administrative console, click Servers > Application servers > *server_name* > Java and Process management > Process definition > Java virtual machine
2. Add generic JVM arguments.
   Note: When containers are running in WebSphere Application Server, you cannot use the -Djava.security.policy argument because this file overrides the WebSphere Application Server administrative access authorization. Therefore, use -Djava.security.auth.policy to set the JAAS authorization policy.
   Enter the following generic JVM arguments or replace the `-Djava.security.auth.policy` entry with the following text:

   ```
   -Djava.security.auth.policy=samples_home/security/xsAuthGroup2.policy
   ```

3. Click OK and save your changes.

Testing group authorization with the sample application

You can test that group authorization is configured by the login module with the sample application.

1. Restart the container servers. For this tutorial, the container servers are the xs1 and xs2 servers.
2. Log in to the sample application. In a web browser, open `http://<host>:<port>/EmployeeManagementWeb/management.jsp` and login with the user name `admin1` and password `admin1`.
3. Display an employee. Click Display an Employee and search for the `authemp2@acme.com` email address. A message displays that the user cannot be found.
4. Add an employee. Click Add an Employee. Add the email `authemp2@acme.com`, the first name `Joe`, and the last name `Doe`. Click Submit. A message displays that the employee has been added.
5. Log in as the operator user. Open a second web browser window and open the following URL: `http://<host>:<port>/EmployeeManagermentWeb/management.jsp`. Use the user name `operator1` and password `operator1`.
6. Attempt to display an employee. Click Display an Employee and search for the `authemp2@acme.com` email address. The employee is displayed.
7. Add an employee. Click Add an Employee. Add the email `authemp3@acme.com`, the first name `Joe`, and the last name `Doe`. Click Submit. The following message displays:

   ```
   An exception occurs when Add the employee. See below for detailed exception messages.
   ```

   The following exception is in the exception chain:

   ```
   java.security.AccessControlException: Access denied
   (com.ibm.websphere.objectgrid.security.MapPermission Grid.Map1 insert)
   ```

   This message displays because the operator user does not have permission to insert data into the Map1 map.

Lesson checkpoint

You configured groups to simplify the assignment of permission to the users of your application.

< Previous | Next >
< Previous

Module 5: Use the xscmd tool to monitor data grids and maps

You can use the **xscmd** tool to show the primary data grids and map sizes of the `Grid` data grid. The **xscmd** tool uses the MBean to query all of the data grid artifacts, such as primary shards, replica shards, container servers, map sizes, and so on.

In this tutorial, the container and catalog servers are running in WebSphere® Application Server application servers. The WebSphere eXtreme Scale run time registers the Managed Beans (MBean) with the MBean server that is created by the WebSphere Application Server run time. The security that is used by the **xscmd** tool is provided by the WebSphere Application Server MBean security. Therefore, WebSphere eXtreme Scale specific security configuration is not necessary.

1. Using a command-line tool, open the *DMGR_PROFILE*/bin directory.
2. Run the **xscmd** tool.
   Use the **-c showPlacement -sf P** command to list the placement of the primary shards. Linux UNIX

   ```
   xscmd.sh -g Grid -ms mapSet -c showPlacement -sf P
   ```

   Windows

   ```
   xscmd.bat -g Grid -ms mapSet -c showPlacement -sf P
   ```

   Before you can view the output, you are prompted to log in with your WebSphere Application Server ID and password.

Related tasks:

Monitoring with the xscmd utility
Administering with the xscmd utility

Lesson checkpoint

You used the **xscmd** tool in WebSphere Application Server.

< Previous
< Previous | Next >

Tutorial: Integrate WebSphere eXtreme Scale security in a mixed environment with an external authenticator

This tutorial demonstrates how to secure WebSphere® eXtreme Scale servers that are partially deployed in a WebSphere Application Server environment.

In the deployment for this tutorial, the container servers are deployed in WebSphere Application Server. The catalog server is deployed as stand-alone server, and is started in a Java Standard Edition (Java SE) environment.
Because the catalog server is not deployed in WebSphere Application Server, you cannot use the WebSphere Application Server Authentication plug-ins. For more information about the process of configuring WebSphere Application Server Authentication plug-ins, see Tutorial: Integrate WebSphere eXtreme Scale

security with WebSphere Application Server. In this tutorial, a different authenticator is required for catalog server authentication. You configure a keystore authenticator to authenticate the clients.

Learning objectives

The learning objectives for this tutorial follow:

- Configure WebSphere eXtreme Scale to use the KeyStoreLoginAuthenticator plug-in
- Configure WebSphere eXtreme Scale transport security to use WebSphere Application Server CSIv2 configuration and the WebSphere eXtreme Scale properties file
- Use Java™ Authentication and Authorization Service (JAAS) authorization in WebSphere Application Server
- Use the **xscmd** utility to monitor the data grids and maps that you created in the tutorial.

Time required

This tutorial takes approximately 4 hours from start to finish.
< Previous | Next >
< Previous | Next >

Introduction: Security in a mixed environment

In this tutorial, you integrate WebSphere® eXtreme Scale security in a mixed environment. The container servers run within WebSphere Application Server, and the catalog service runs in stand-alone mode. Because the catalog server is in stand-alone mode, you must configure an external authenticator.

Important: If both your container servers and catalog server are running within WebSphere Application Server, you can use the WebSphere Application Server Authentication plug-ins or an external authenticator. For more information about using the WebSphere Application Server Authentication plug-ins, see Tutorial: Integrate WebSphere eXtreme Scale security with WebSphere Application Server.

Learning objectives

The learning objectives for this tutorial follow:

- Configure WebSphere eXtreme Scale to use the KeyStoreLoginAuthenticator plug-in
- Configure WebSphere eXtreme Scale transport security to use WebSphere Application Server CSIv2 configuration and the WebSphere eXtreme Scale properties file
- Use Java™ Authentication and Authorization Service (JAAS) authorization in WebSphere Application Server
- Use the **xscmd** utility to monitor the data grids and maps that you created in the tutorial.

Time required

This tutorial takes approximately 4 hours from start to finish.

Skill level

Intermediate.

Audience

Developers and administrators that are interested in the security integration between WebSphere eXtreme Scale and WebSphere Application Server and configuring external authenticators.

System requirements

- WebSphere Application Server Version 6.1 or Version 7.0.0.11 or later with the following fixes applied:interim fix PM20613 and interim fix PM15818.
- The catalog server must be running on a stand-alone installation, not an installation that is integrated with WebSphere Application Server.
- Update the Java runtime to apply the following fix: IZ79819: IBMJDK FAILS TO READ PRINCIPAL STATEMENT WITH WHITESPACE FROM SECURITY FILE
- The stand-alone node that runs the catalog service must use the IBM Software Development Kit Version 1.6 J9. This Software Development Kit is included in the WebSphere Application Server installation. The catalog server node must be a stand-alone installation because you cannot run the **startOgServer** command within an installation of WebSphere eXtreme Scale on WebSphere Application Server.

This tutorial uses four WebSphere Application Server application servers and one deployment manager to demonstrate the sample.

Prerequisites

A basic understanding of the following items is helpful before you start this tutorial:

- WebSphere eXtreme Scale programming model
- Basic WebSphere eXtreme Scale security concepts
- Basic WebSphere Application Server security concepts

For a background information about WebSphere eXtreme Scale and WebSphere Application Server security integration, see Security integration with WebSphere Application Server.

Modules in this tutorial

- **Module 1: Prepare the mixed WebSphere Application Server and stand-alone environment**

  Before you start the tutorial, you must create a basic topology that includes container servers that run within WebSphere Application Server. In this tutorial, the catalog servers run in stand-alone mode.
- **Module 2: Configure WebSphere eXtreme Scale authentication in a mixed environment**

  By configuring authentication, you can reliably determine the identity of the requester. WebSphere eXtreme Scale supports both client-to-server and server-to-server authentication.
- **Module 3: Configure transport security**

  Configure transport security to secure data transfer between the clients and servers in the configuration.
- **Module 4: Use Java Authentication and Authorization Service (JAAS) authorization in WebSphere Application Server**

  Now that you have configured authentication for clients, you can further configure authorization to give different users varying permissions. For example, an "operator" user might only be able to view data, while a "manager" user can perform all operations.

< Previous | Next >
< Previous | Next >

Module 1: Prepare the mixed WebSphere Application Server and stand-alone environment

Before you start the tutorial, you must create a basic topology that includes container servers that run within WebSphere® Application Server. In this tutorial, the catalog servers run in stand-alone mode.

Learning objectives

With the lessons in this module, you learn how to:

- Understand the mixed topology and the files that are necessary for the tutorial
- Configure WebSphere Application Server to run the container servers

Time required

This module takes approximately 60 minutes.

Lessons in this module

- **Lesson 1.1: Understand the topology and get the tutorial files**

  To prepare your environment for the tutorial, you must configure the catalog and container servers for the topology.
- **Lesson 1.2: Configure the WebSphere Application Server environment**

  To prepare your environment for the tutorial, you must configure WebSphere Application Server security. Enable administration and application security using internal file-based federated repositories as a user account registry. Then, you can create server clusters to host the client application and container servers. You also must create and start the catalog servers.

< Previous | Next >
< Previous | Next >

Lesson 1.1: Understand the topology and get the tutorial files

To prepare your environment for the tutorial, you must configure the catalog and container servers for the topology.

This lesson guides you through the sample topology and applications that are used to in the tutorial. To begin running the tutorial, you must download the applications and place the configuration files in the correct locations for your environment. You can download the sample application from the IBM elastic caching community.

Topology

In this tutorial, you create the following clusters in the WebSphere Application Server cell:

- **appCluster cluster**: Hosts the EmployeeManagement sample enterprise application. This cluster has two application servers: s1 and s2.

- **xsCluster cluster**: Hosts the eXtreme Scale container servers. This cluster has two application servers: xs1 and xs2.

In this deployment topology, the s1 and s2 application servers are the client servers that access data that is being stored in the data grid. The xs1 and xs2 servers are the container servers that host the data grid.
Alternative configuration: You can host all of the application servers in a single cluster, such as in the appCluster cluster. With this configuration, all of the servers in the cluster are both clients and container servers. This tutorial uses two clusters to distinguish between the application servers that are hosting the clients and container servers.

In this tutorial, you configure a catalog service domain that consists of a remote server that is not in the WebSphere Application Server cell. This configuration is not the default, which results in the catalog servers running on the deployment manager and other processes in the WebSphere Application Server cell. See Creating catalog service domains in WebSphere Application Server for more information about creating a catalog service domain that consists of remote servers.

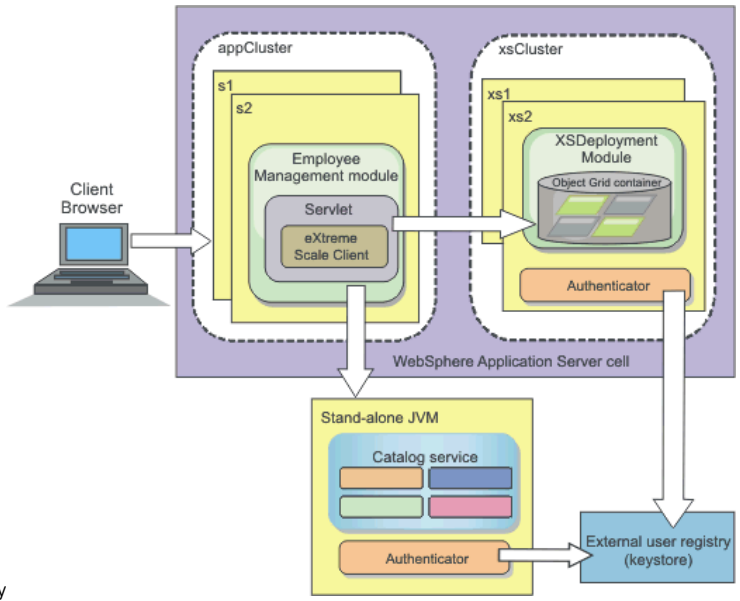


Figure 1. Tutorial topology

Applications

In this tutorial, you are using two applications and one shared library file:

- **EmployeeManagement.ear**: The EmployeeManagement.ear application is a simplified Java™ 2 Platform, Enterprise Edition (J2EE) enterprise application. It contains a web module to manage the employee profiles. The web module contains the management.jsp file to display, insert, update, and delete employee profiles that are stored in the container servers.
- **XSDeployment.ear**: This application contains an enterprise application module with no application artifacts. The cache objects are packaged in the EmployeeData.jar file. The EmployeeData.jar file is deployed as a shared library for the XSDeployment.ear file, so that the XSDeployment.ear file can access the classes. The purpose of this application is to package the eXtreme Scale configuration file and property file. When this enterprise application is started, the eXtreme Scale configuration files are automatically detected by the eXtreme Scale run time, so the container servers are created. These configuration files include the objectGrid.xml and objectGridDeployment.xml files.
- **EmployeeData.jar**: This jar file contains one class: the com.ibm.websphere.sample.xs.data.EmployeeData class. This class represents employee data that is stored in the grid. This Java archive (JAR) file is deployed with the EmployeeManagement.ear and XSDeployment.ear files as a shared library.

Get the tutorial files

1. Download the WASSecurity.zip and security_extauth.zip files from the WebSphere eXtreme Scale wiki.
2. Extract the WASSecurity.zip file to a directory for viewing the binary and source artifacts, for example a wxs_samples/ directory. This directory is referred to as *samples_home* for the remainder of the tutorial. Refer to the README.txt file in the package for a description of the contents and how to load the source into your Eclipse workspace. The following ObjectGrid configuration files are in the META-INF directory:
   - objectGrid.xml
   - objectGridDeployment.xml
3. Create a directory to store the property files that are used to secure this environment. For example, you might create the /opt/wxs/security directory.
4. Extract the security_extauth.zip file to *samples_home*. The security_extauth.zip file contains the following security configuration files that are used in this tutorial:. These configuration files follow:
   - catServer3.props
   - server3.props
   - client3.props
   - security3.xml
   - xsAuth3.props
   - xsjaas3.config
   - sampleKS3.jks

About the configuration files

The objectGrid.xml and objectGridDeployment.xml files create the data grids and maps that store the application data.

These configuration files must be named objectGrid.xml and objectGridDeployment.xml. When the application server starts, eXtreme Scale detects these files in the META-INF directory of the EJB and web modules. If these files are found, it assumed that the Java virtual machine (JVM) acts as a container server for the defined data grids in the configuration files.

objectGrid.xml file

The objectGrid.xml file defined one ObjectGrid named `Grid`. The `Grid` data grid has one map, the `Map1` map, that stores the employee profile for the application.

```
<?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="Grid" txTimeout="15">
            <backingMap name="Map1" />
        </objectGrid>
    </objectGrids>

</objectGridConfig>
```

objectGridDeployment.xml file

The objectGridDeployment.xml file specifies how to deploy the `Grid` data grid. When the grid is deployed, it has five partitions and one synchronous replica.

```
<?xml version="1.0" encoding="UTF-8"?>

<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy ../deploymentPolicy.xsd"
 xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">

    <objectgridDeployment objectgridName="Grid">
        <mapSet name="mapSet" numberOfPartitions="5" minSyncReplicas="0" maxSyncReplicas="1" >
            <map ref="Map1"/>
        </mapSet>
    </objectgridDeployment>

</deploymentPolicy>
```

Lesson checkpoint

In this lesson, you learned about the topology for the tutorial and added the configuration files and sample applications to your environment.

< Previous | Next >
< Previous | Next >

Lesson 1.2: Configure the WebSphere Application Server environment

To prepare your environment for the tutorial, you must configure WebSphere® Application Server security. Enable administration and application security using internal file-based federated repositories as a user account registry. Then, you can create server clusters to host the client application and container servers. You also must create and start the catalog servers.

The following steps were written using WebSphere Application Server Version 7.0. However, you can also apply the concepts apply to earlier versions of WebSphere Application Server.

Configure WebSphere Application Server security

Create and augment profiles for the deployment manager and nodes with WebSphere eXtreme Scale. See Installing WebSphere eXtreme Scale or WebSphere eXtreme Scale Client with WebSphere Application Server for more information.
Configure WebSphere Application Server security.

a. In the WebSphere Application Server administrative console, click Security > Global Security.
b. Select Federated repositories as the Available realm definition. Click Set as current.
c. Click Configure.. to go to the Federated repositories panel.
d. Enter the Primary administrative user name, for example, `admin`. Click Apply.
e. When prompted, enter the administrative user password and click OK. Save your changes.
f. On the Global Security page, verify that Federated repositories setting is set to the current user account registry.
g. Select the following items: Enable administrative security, Enable application security, and Use Java 2 security to restrict application access to local resources. Click Apply and save your changes.
h. Restart the deployment manager and any running application servers.

The WebSphere Application Server administrative security is enabled using the internal file-based federated repositories as the user account registry.

Create server clusters

Create two server clusters in your WebSphere Application Server configuration: The `appCluster` cluster to host the sample application for the tutorial and the `xsCluster` cluster to host the data grid.

1. In the WebSphere Application Server administrative console, open the clusters panel. Click Servers > Clusters > WebSphere application server clusters > New.
2. Type `appCluster` as the cluster name, leave the Prefer local option selected, and click Next.
3. Create servers in the cluster. Create a server named `s1`, keeping the default options. Add an additional cluster member named `s2`.
4. Complete the remaining steps in the wizard to create the cluster. Save the changes.
5. Repeat these steps to create the `xsCluster` cluster. This cluster has two servers, named `xs1` and `xs2`.

Create a catalog service domain

After configuring the server cluster and security, you must define where catalog servers start.

Define a catalog service domain in WebSphere eXtreme Scale

1. In the WebSphere Application Server administrative console, click System administration > WebSphere eXtreme Scale > Catalog service domains.
2. Create the catalog service domain. Click New. Create the catalog service domain with the name `catalogService1`, and enable the catalog service domain as the default.
3. Add remote servers to the catalog service domain. Select Remote server. Provide the host name where the catalog server is running. Use the listener port value of `16809` for this example.
4. Click OK and save your changes.

Lesson checkpoint

You enabled security in WebSphere Application Server, and created the server topolgy for WebSphere eXtreme Scale.

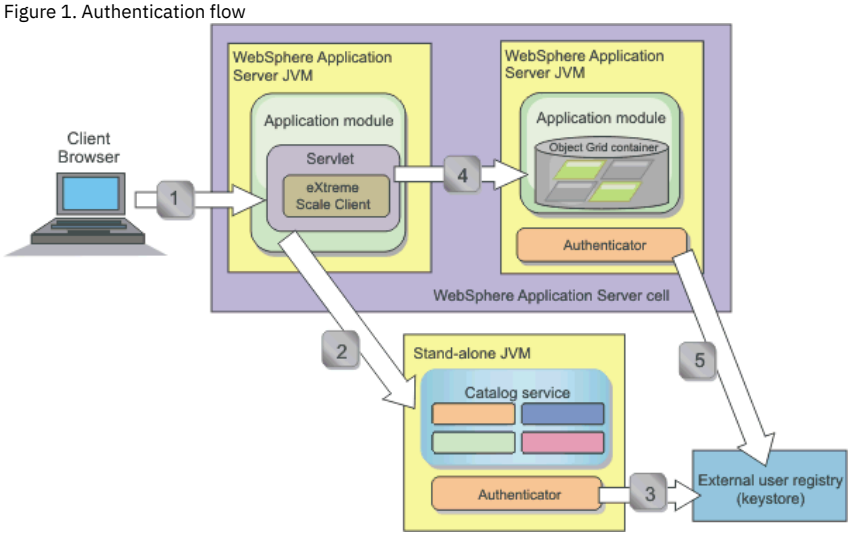
< Previous | Next >
< Previous | Next >

Module 2: Configure WebSphere eXtreme Scale authentication in a mixed environment

By configuring authentication, you can reliably determine the identity of the requester. WebSphere® eXtreme Scale supports both client-to-server and server-to-server authentication.

Authentication flow

Figure 1. Authentication flow



The previous diagram shows two application servers. The first application server hosts the web application, which is also a WebSphere eXtreme Scale client. The second application server hosts a container server. The catalog server is running in a stand-alone Java virtual machine (JVM) instead of WebSphere Application Server.

The arrows marked with numbers in the diagram indicate the authentication flow:

1. An enterprise application user accesses the web browser, and logs in to the first application server with a user name and password. The first application server sends the client user name and password to the security infrastructure to authenticate to the user registry. This user registry is a keystore. As a result, the security information is stored on the WebSphere Application Server thread.
2. The JavaServer Pages (JSP) file acts as a WebSphere eXtreme Scale client to retrieve the security information from the client property file. The JSP application that is acting as the WebSphere eXtreme Scale client sends the WebSphere eXtreme Scale client security credential along with the request to the catalog server. Sending the security credential with the request is considered a *runAs* model. In a runAs model, the web browser client runs as a WebSphere eXtreme Scale client to access the data stored in the container server. The client uses a Java virtual machine (JVM)-wide client credential to connect to the WebSphere eXtreme Scale servers. Using the runAs model is like connecting to a database with a data source level user ID and password.
3. The catalog server receives the WebSphere eXtreme Scale client credential, which includes the WebSphere Application Server security tokens. Then, the catalog server calls the authenticator plug-in to authenticate the client credential. The authenticator connects to the external user registry and sends the

client credential to the user registry for authentication.
4. The client sends the user ID and password to the container server that is hosted in the application server.
5. The container service, hosted in the application server, receives the WebSphere eXtreme Scale client credential, which is the user id and password pair. Then, the container server calls the authenticator plug-in to authenticate the client credential. The authenticator connects to the keystore user registry and sends the client credential to the user registry for authentication

Learning objectives

With the lessons in this module, you learn how to:

- Configure WebSphere eXtreme Scale client security.
- Configure WebSphere eXtreme Scale catalog server security.
- Configure WebSphere eXtreme Scale container server security.
- Install and run the sample application.

Time required

This module takes approximately 60 minutes.

Lessons in this module

- **Lesson 2.1: Configure WebSphere eXtreme Scale client security**

  You configure the client properties with a properties file. The client properties file indicates the CredentialGenerator implementation class to use.
- **Lesson 2.2: Configure catalog server security**

  A catalog server contains two different levels of security information: The first level contains the security properties that are common to all the WebSphere eXtreme Scale servers, including the catalog service and container servers. The second level contains the security properties that are specific to the catalog server.
- **Lesson 2.3: Configure container server security**

  When a container server connects to the catalog service, the container server gets all the security configurations that are configured in the Object Grid Security XML file. The ObjectGrid Security XML file defines authenticator configuration, the login session timeout value, and other configuration information. A container server also has its own server-specific security properties in the server property file.
- **Lesson 2.4: Install and run the sample**

  After authentication is configured, you can install and run the sample application.

< Previous | Next >
< Previous | Next >

Lesson 2.1: Configure WebSphere eXtreme Scale client security

You configure the client properties with a properties file. The client properties file indicates the CredentialGenerator implementation class to use.

Client properties file contents

The tutorial uses WebSphere Application Server security tokens for the client credential. The *samples_home*/security_extauth directory contains the client3.props file.

The client3.props file includes the following settings:

securityEnabled
: Enables WebSphere eXtreme Scale client security. The value is set to `true` to indicate that the client must send available security information to the server.

credentialAuthentication
: Specifies the client credential authentication support. The value is set to `Supported` to indicate that the client supports credential authentication.

credentialGeneratorClass
: Specifies the name of the class that implements the com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator interface. The value is set to the com.ibm.websphere.objectgrid.security.plugins.builtins. UserPasswordCredentialGenerator class so that the client retrieves the security information from the UserPasswordCredentialGenerator class.

credentialGeneratorProps
: Specifies the user name and password: `manager manager1`. The user name is `manager`, and the password is `manager1`. You can also use the **FilePasswordEncoder.bat|sh** command to encode this property using an exclusive or (xor) algorithm.

Setting the client properties file using Java™ virtual machine (JVM) properties

In the administrative console, complete the following steps to both the `s1` and `s2` servers in the `appCluster` cluster. If you are using a different topology, complete the following steps to all of the application servers to which the `EmployeeManagement` application is deployed.

1. Click Servers > WebSphere application servers > *server_name* > Java and Process Management > Process definition > Java Virtual Machine.

2. Create the following generic JVM property to set the location of the client properties file:

   ```
   -Dobjectgrid.client.props=samples_home/security_extauth/client3.props
   ```

   When you connect to a secure data grid, you must configure the client application to provide a valid client security configuration. You can configure the client security configuration through the client application, or you can defined the configuration in a client properties file that has the same value of the JVM property, objectgrid.client.props. When you use the objectgrid.client.props property, the ObjectGridManager obtains the client security configuration from the client properties file and uses this information to connect to the data grid.

3. Click OK and save your changes.

Lesson checkpoint

You edited the client properties file and configured the servers in the `appCluster` cluster to use the client properties file. This properties file indicates the CredentialGenerator implementation class to use.

< Previous | Next >
< Previous | Next >

Lesson 2.2: Configure catalog server security

A catalog server contains two different levels of security information: The first level contains the security properties that are common to all the WebSphere® eXtreme Scale servers, including the catalog service and container servers. The second level contains the security properties that are specific to the catalog server.

The security properties that are common to the catalog servers and container servers are configured in the security XML descriptor file. An example of common properties is the authenticator configuration, which represents the user registry and authentication mechanism. See Security descriptor XML file for more information about the security properties.
To configure the security XML descriptor file in a Java SE environment, use a -clusterSecurityFile option when you run the **startOgServer** command. Specify a value in a file format, such as *samples_home*/security_extauth/security3.xml.

security3.xml file

In this tutorial, the security3.xml file is in the *samples_home*/security_extauth directory. The content of the security3.xml file with the comments removed follows:

```
<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.KeyStoreLoginAuthenticator">
                </authenticator>
        </security>
</securityConfig>
```

The following properties are defined in the security3.xml file:

securityEnabled
: The securityEnabled property is set to `true`, which indicates to the catalog server that the WebSphere eXtreme Scale global security is enabled.

authenticator
: The authenticator is configured as the com.ibm.websphere.objectgrid.security.plugins.builtins.KeyStoreLoginAuthenticator class. With this built-in implementation of the Authenticator plug-in, the user ID and password is passed to verify that it is configured in the keystore file. The KeyStoreLoginAuthenticator class uses a `KeyStoreLogin` login module alias, so a Java Authentication and Authorization Service (JAAS) login configuration is required.

catServer3.props file

The server property file stores the server-specific properties, which include the server-specific security properties. See Server properties file for more information. You can use -serverProps option to specify the catalog server property when you run the **startOgServer** command. For this tutorial, a catServer3.props file is in the c directory. The content of the catServer3.props file with the comments removed follows:

```
securityEnabled=true
credentialAuthentication=Required
transportType=TCP/IP
secureTokenManagerType=none
authenticationSecret=ObjectGridDefaultSecret
```

securityEnabled
: The securityEnabled property is set to `true` to indicate that this catalog server is a secure server.

credentialAuthentication
: The credentialAuthentication property is set to `Required`, so any client that is connecting to the server is required to provide a credential. Iin the client property file, the credentialAuthentication value is set to `Supported`, so the server receives the credentials that are sent by the client.

secureTokenManagerType
: The secureTokenManagerType is set to `none` to indicate that the authentication secret is not encrypted when joining the existing servers.

authenticationSecret
: The authenticationSecret property is set to `ObjectGridDefaultSecret`. This secret string is used to join the eXtreme Scale server cluster. When a server joins the data grid, it is challenged to present the secret string. If the secret string of the joining server matches the string in the catalog server, the joining server is accepted. If the string does not match, the join request is rejected.

transportType
: The transportType property is set to `TCP/IP` initially. Later in the tutorial, transport security is enabled.

xsjaas3.config file

Because the KeyStoreLoginAuthenticator implementation uses a login module, you must configure the login model with a JAAS authentication login configuration file. The contents of the xsjaas3.config file follows:

```
KeyStoreLogin{
com.ibm.websphere.objectgrid.security.plugins.builtins.KeyStoreLoginModule required
     keyStoreFile="samples_home/security_extauth/sampleKS3.jks" debug = true;
};
```

If you used a location for *samples_home* other than /wxs_samples/, you need to update the location of the keyStoreFile. This login configuration indicates that the com.ibm.websphere.objectgrid.security.plugins.builtins.KeyStoreLoginModule module is used as the login module. The keystore file is set to the sampleKS3.jks file.

Windows Important: If you are using Windows, the directory path does not support backslashes. If you have used backslashes, you must escape any backslash (\ ) characters in the path. For example, if you want to use the path C:\opt\ibm, enter C:\\opt\\ibm in the properties file. Windows directories with spaces are not supported.

The sampleKS3.jks sample keystore file stores two user IDs and the passwords: manager/manager1 and cashier/cashier1.

You can use the following **keytool** commands to create this keystore:

- `keytool -genkey -v -keystore ./sampleKS3.jks -storepass sampleKS1 -alias manager -keypass manager1 -dname CN=manager,O=acme,OU=OGSample -validity 10000`
- `keytool -genkey -v -keystore ./sampleKS3.jks -storepass sampleKS1 -alias operator -keypass operator1 -dname CN=operator,O=acme,OU=OGSample -validity 10000`

Start the catalog server with security enabled

To start the catalog server, issue the **startOgServer** command with the -clusterSecurityFile and -serverProps parameters to pass in the security properties.

Use a stand-alone installation of WebSphere eXtreme Scale to run the catalog server. When using the stand-alone installation image, you must use the IBM SDK. You can use the SDK that is included with WebSphere Application Server by setting the *JAVA_HOME* variable to point to the IBM SDK. For example, `set JAVA_HOME=was_root/IBM/WebSphere/AppServer/java/`

1. Go to the bin directory.

   ```
   cd wxs_home/bin
   ```

2. Run the **startOgServer** command.

   Linux UNIX

   ```
   ./startOgServer.sh cs1 -listenerPort 16809 -JMXServicePort 16099 -catalogServiceEndPoints
   cs1:[HOST_NAME]:16601:16602 -clusterSecurityFile samples_home/security_extauth/security3.xml
   -serverProps samples_home/security_extauth/catServer3.props -jvmArgs
   -Djava.security.auth.login.config="samples_home/security_extauth/xsjaas3.config"
   ```

   Windows

   ```
   startOgServer.bat cs1 -listenerPort 16809 -JMXServicePort 16099 -catalogServiceEndPoints
   cs1:[HOST_NAME]:16601:16602 -clusterSecurityFile samples_home/security_extauth/security3.xml
   -serverProps samples_home/security_extauth/catServer3.props -jvmArgs
   -Djava.security.auth.login.config="samples_home/security_extauth/xsjaas3.config"
   ```

After you run the **startOgServer** command, a secure server starts with listener port 16809, client port 16601, peer port 16602, and JMX port 16099. If a port conflict exists, change the port number to an unused port number.

Stop a catalog server that has security enabled

You can use the **stopOgServer** command to stop the catalog server.

1. Go to the bin directory.

   ```
   cd wxs_home/bin
   ```

2. Run the **stopOgServer** command. Linux UNIX

   ```
   stopOgServer.sh cs1 -catalogServiceEndPoints localhost:16809 -clientSecurityFile
   samples_home/security_extauth/client3.props
   ```

   Windows

   ```
   stopOgServer.bat cs1 -catalogServiceEndPoints localhost:16809 -clientSecurityFile
   samples_home/security_extauth/client3.props
   ```

Lesson checkpoint

You configured catalog server security by associating the security3.xml, catServer3.props, xsjaas3.config files with the catalog service.

< Previous | Next >
< Previous | Next >

Lesson 2.3: Configure container server security

When a container server connects to the catalog service, the container server gets all the security configurations that are configured in the Object Grid Security XML file. The ObjectGrid Security XML file defines authenticator configuration, the login session timeout value, and other configuration information. A container server also has its own server-specific security properties in the server property file.

Configure the server property file with the -Dobjectgrid.server.props Java virtual machine (JVM) property. The file name specified for this property is an absolute file path, such as *samples_home*/security_extauth/server3.props.
In this tutorial, the container servers are hosted in the `xs1` and `xs2` servers in the `xsCluster` cluster.

server3.props file

The server3.props file is in the *samples_home*/security_extauth/ directory. The content of the server3.props file follows:

```
securityEnabled=true
credentialAuthentication=Required
secureTokenManagerType=none
authenticationSecret=ObjectGridDefaultSecret
```

securityEnabled
: The securityEnabled property is set to `true` to indicate that this container server is a secure server.

credentialAuthentication
: The credentialAuthentication property is set to `Required`, so any client that is connecting to the server is required to provide a credential. In the client property file, the credentialAuthentication property is set to `Supported`, so the server receives the credential that is sent by the client.

secureTokenManagerType
: The secureTokenManagerType is set to `none` to indicate that the authentication secret is not encrypted when joining the existing servers.

authenticationSecret
: The authenticationSecret property is set to `ObjectGridDefaultSecret`. This secret string is used to join the eXtreme Scale server cluster. When a server joins the data grid, it is challenged to present the secret string. If the secret string of the joining server matches the string in the catalog server, the joining server is accepted. If the string does not match, the join request is rejected.

Setting the server properties file with JVM properties

Set the server properties file on the xs1 and xs2 servers. If you are not using the topology for this tutorial, set the server properties file on all of the application servers that you are using to host container servers.

1. Open the Java™ virtual machine page for the server. Servers > WebSphere application servers > *server_name* > Java and Process Management > Process definition > Java Virtual Machine.
2. Add the generic JVM argument:

   ```
   -Dobjectgrid.server.props=samples_home/security_extauth/server3.props
   ```

3. Click OK and save your changes.

Adding the custom login module

The container server uses the same KeyStoreAuthenticator implementation as the catalog server. The KeyStoreAuthenticator implementation uses a KeyStoreLogin login module alias, so you must add a custom login module to the application login model entries.

1. In the WebSphere Application Server administrative console, click Security > Global security > Java Authentication and Authorization Service.
2. Click Application logins.
3. Click New, add an alias `KeyStoreLogin`. Click Apply.
4. Under JAAS login modules, click New.
5. Enter `com.ibm.websphere.objectgrid.security.plugins.builtins.KeyStoreLoginModule` as the module class name, and choose SUFFICIENT as the authentication strategy. Click Apply.
6. Add the `keyStoreFile` custom property with value *samples_home*`/security_extauth/sampleKS.jks`.
   Windows Important: If you are using Windows, the directory path does not support backslashes. If you have used backslashes, you must escape any backslash (\ ) characters in the path. For example, if you want to use the path C:\opt\ibm, enter C:\\opt\\ibm in the properties file. Windows directories with spaces are not supported.
7. Optional: Add the `debug` custom property with value `true`.
8. Save the configuration.

Lesson checkpoint

Now the WebSphere eXtreme Scale server authentication is secured. By configuring this security, all the applications that try to connect to the WebSphere eXtreme Scale servers are required to provide a credential. In this tutorial, the KeyStoreLoginAuthenticator is the authenticator. As a result, the client is required

to provide a user name and password.

< Previous | Next >
< Previous | Next >

Lesson 2.4: Install and run the sample

After authentication is configured, you can install and run the sample application.

Creating a shared library for the EmployeeData.jar file

1. In the WebSphere Application Server administrative console, open the Shared Libraries page. Click Environment > Shared libraries.
2. Choose the cell scope.
3. Create the shared library. Click New. Enter `EmployeeManagementLIB` as the Name. Enter the path to the EmployeeData.jar in the classpath, for example, *samples_home*/WASSecurity/EmployeeData.jar.
4. Click Apply.

Installing the sample

1. Install the EmployeeManagement_extauth.ear file under the *samples_home*/security_extauth directory.
   Important: The EmployeeManagement_extauth.ear file is different from the *samples_home*/WASSecurity/EmployeeManagement.ear file. The manner in which the ObjectGrid session is retrieved has been updated to use the credential that is cached in the client property file in the EmployeeManagement_extauth.ear application. See the comments in the com.ibm.websphere.sample.xs.DataAccessor class in the *samples_home*/WASSecurity/EmployeeManagementWeb project to see the code that was updated for this change.
   a. To begin the installation, click Applications > New application > New Enterprise Application. Choose the detailed path for installing the application.
   b. On the Map modules to servers step, specify the `appCluster` cluster to install the EmployeeManagementWeb module.
   c. On the Map shared libraries step, select the `EmployeeManagementWeb` module.
   d. Click Reference shared libraries. Select the `EmployeeManagementLIB` library.
   e. Map the `webUser` role to All Authenticated in Application's Realm.
   f. Click OK.
   The clients run in the `s1` and `s2` servers in this cluster.
2. Install the sample XSDeployment.ear file that is in the *samples_home*/WASSecurity directory.
   a. To begin the installation, click Applications > New application > New Enterprise Application. Choose the detailed path for installing the application.
   b. On the Map modules to servers step, specify the `xsCluster` cluster to install the `XSDeploymentWeb` web module.
   c. On the Map shared libraries step, select the `XSDeploymentWeb` module.
   d. Click Reference shared libraries. Select the `EmployeeManagementLIB` library.
   e. Click OK.
   The `xs1` and `xs2` servers in this cluster host the container servers.
3. Verify that the catalog server is started. For more information about starting a catalog server for this tutorial, see Start the catalog server with security enabled.
4. Restart the xsCluster cluster. When the xsCluster starts, the XSDeployment application starts, and a container server is started on the xs1 and xs2 servers respectively. If you look at the SystemOut.log file of the xs1 and xs2 servers, the following message that indicates the server properties file is loaded is displayed:

   ```
   CWOBJ0913I: Server property files have been loaded:
   samples_home/security_extauth/server3.props.
   ```

5. Restart the appClusters cluster. When the cluster appCluster starts, the EmployeeManagement application also starts. If you look at the SystemOut.log file of the s1 and s2 servers, you can see the following message that indicates that the client properties file is loaded.

   ```
   CWOBJ0924I: The client property file {0} has been loaded.
   ```

   If you are using WebSphere eXtreme Scale Version 7.0, the English-only CWOBJ9000I message displays to indicate that the client property file has been loaded. If you do not see the expected message, verify that you configured the -Dobjectgrid.server.props or -Dobjectgrid.client.props property in the JVM argument. If you do have the properties configured, make sure the dash (-) is a UTF character.

Running the sample application

1. Run the management.jsp file. In a web browser, access `http://<your_servername>:<port>/EmployeeManagementWeb/management.jsp`. For example, you might use the following URL: `http://localhost:9080/EmployeeManagementWeb/management.jsp`.
2. Provide authentication to the application. Enter the credentials of the user that you mapped to the webUser role. By default, this user role is mapped to all authenticated users. Type any valid user name and password, such as the administrative user name and password. A page to display, add, update, and delete employees displays.
3. Display employees. Click Display an Employee. Enter `emp1@acme.com` as the email address, and click Submit. A message displays that the employee cannot be found.
4. Add an employee. click Add an Employee. Enter `emp1@acme.com` as the email address, enter `Joe` as the given name, and `Doe` as the surname. Click Submit. A message displays that an employee with the `emp1@acme.com` address has been added.
5. Display the new employee. Click Display an Employee. Enter `emp1@acme.com` as the email address with empty fields for the first and surnames, and click Submit. A message displays that the employee has been found, and the correct names are displayed in the given name and surname fields.
6. Delete the employee. Click Delete an employee. Enter `emp1@acme.com` and click Submit. A message is displayed that the employee has been deleted.

Because the catalog server transport type is set to `TCP/IP`, verify that the server s1 and s2 outbound transport setting is not set to `SSL-Required`. Otherwise, an exception occurs. If you look at the system out file of the catalog server, logs/cs1/SystemOut.log file, the following debug output to indicates the key store

authentication:

```
SystemOut     O [KeyStoreLoginModule] initialize: Successfully loaded key store
SystemOut     O [KeyStoreLoginModule] login: entry
SystemOut     O [KeyStoreLoginModule] login: user entered user name: manager
SystemOut     O   Print out the certificates:
...
```

Lesson checkpoint

You installed and ran the sample application.

< Previous | Next >
< Previous | Next >

Module 3: Configure transport security

Configure transport security to secure data transfer between the clients and servers in the configuration.

In the previous module in the tutorial, you enabled WebSphere® eXtreme Scale authentication. With authentication, any application that tries to connect to the WebSphere eXtreme Scale server is required to provide a credential. Therefore, no unauthenticated client can connect to the WebSphere eXtreme Scale server. The clients must be an authenticated application that is running in a WebSphere Application Server cell.

With the configuration up to this module, the data transfer between the clients in the appCluster cluster and servers in the xsCluster cluster is not encrypted. This configuration might be acceptable if your WebSphere Application Server clusters are installed on servers behind a firewall. However, in some scenarios, non-encrypted traffic is not accepted for some reasons even though the topology is protected by firewall. For example, a government policy might enforce encrypted traffic. WebSphere eXtreme Scale supports Transport Layer Security/Secure Sockets Layer (TLS/SSL) for secure communication between ObjectGrid endpoints, which include client servers, container servers, and catalog servers.

In this sample deployment, the eXtreme Scale clients and container servers are all running in the WebSphere Application Server environment. Client or server properties are not necessary to configure the SSL settings because the eXtreme Scale transport security is managed by the Application Server Common Secure Interoperability Protocol Version 2 (CSIV2) transport settings. WebSphere eXtreme Scale servers use the same Object Request Broker (ORB) instance as the application servers in which they run. Specify all the SSL settings for client and container servers in the WebSphere Application Server configuration using these CSIv2 transport settings. You must configure the SSL properties in the server properties file for the catalog server.

Learning objectives

After completing the lessons in this module, you know how to:

- Configure CSIv2 inbound and outbound transport.
- Add SSL properties to the catalog server properties file.
- Check the ORB properties file.
- Run the sample.

Time required

This module takes approximately 60 minutes.

Prerequisites

This step of the tutorial builds upon the previous modules. Complete the previous modules in this tutorial before you configure transport security.

Lessons in this module

- **Lesson 3.1: Configure CSIv2 inbound and outbound transport**

  To configure Transport Layer Security/Secure Sockets Layer (TLS/SSL) for the server transport, set the Common Secure Interoperability Protocol Version 2 (CSIv2) inbound transport and CSIv2 outbound transport to `SSL-Required` for all the WebSphere Application Server servers that host clients, catalog servers, and container servers.
- **Lesson 3.2: Add SSL properties to the catalog server properties file**

  The catalog server is running outside of WebSphere Application Server, so you must configure the SSL properties in the server properties file.
- **Lesson 3.3: Run the sample**

  Restart all the servers and run the sample application again. You should be able to run through the steps without any problems.

< Previous | Next >
< Previous | Next >
< Previous | Next >

Lesson 3.1: Configure CSIv2 inbound and outbound transport

To configure Transport Layer Security/Secure Sockets Layer (TLS/SSL) for the server transport, set the Common Secure Interoperability Protocol Version 2 (CSIv2) inbound transport and CSIv2 outbound transport to `SSL-Required` for all the WebSphere® Application Server servers that host clients, catalog servers, and container servers.

In the tutorial example topology, you must set these properties for the, s1, s2, xs1, and xs2 application servers. The following steps configure the inbound and outbound transports for all the servers in the configuration.

Set the inbound and outbound transports in the administrative console. Make sure that administrative security is enabled.

- **WebSphere Application Server Version 6.1**: Click Security > Secure Administration > Application.. > RMI/IIOP Security and change the transport type to SSL-Required.
- **WebSphere Application Server Version 7.0**: Click Security > Global Security > RMI/IIOP Security > CSIv2 inbound communications. Change the transport type under the CSIv2 Transport Layer to SSL-Required. Repeat this step to configure CSIv2 outbound communications.

You can use centrally managed endpoint security settings, or you can configure SSL repositories. See Common Secure Interoperability Version 2 transport inbound settings for more information.
< Previous | Next >
< Previous | Next >
< Previous | Next >

Lesson 3.2: Add SSL properties to the catalog server properties file

The catalog server is running outside of WebSphere® Application Server, so you must configure the SSL properties in the server properties file.

The other reason to configure the SSL properties in the server properties file is because the catalog server 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 Secure Sockets Layer (SSL) properties in the server properties file for the catalog server.

SSL properties in the catServer3.props file

```
alias=default
contextProvider=IBMJSSE2
protocol=SSL
keyStoreType=PKCS12
keyStore=/was_root/IBM/WebSphere/AppServer/profiles/
<deployment_manager_name>/config/cells/<cell_name>/nodes/
<node_name>/key.p12
keyStorePassword=WebAS
trustStoreType=PKCS12
trustStore=/was_root/IBM/WebSphere/AppServer/profiles/
<deployment_manager_name>/config/cells/<cell_name>/nodes/
<node_name>/trust.p12
trustStorePassword=WebAS
clientAuthentication=false
```

The catServer3.props file is using the default WebSphere Application Server node level keystore and truststore. If you are deploying a more complex deployment environment, you must choose the correct keystore and truststore. In some cases, you must create a keystore and truststore and import the keys from keystores from the other servers. Notice that the `WebAS` string is the default password of the WebSphere Application Server keystore and truststore. See Default self-signed certificate configuration for more details.
These entries are already included in the *samples_home*/security_extauth/catServer3.props file as comments. You can uncomment the entries and make the appropriate updates for your installation to the *was_root*, <*deployment_manager_name*>, <*cell_name*>, and <*node_name*> variables.

After configuring the SSL properties, change the transportType property value from `TCP/IP` to `SSL-Required`.

SSL properties in the client3.props file

You must also configure the SSL properties in the client3.props file because this file is used when you stop the catalog server that is running outside of WebSphere Application Server.

These properties have no effect on the client servers that are running in WebSphere Application Server because they are using the WebSphere Application Server Common Security Interoperability Protocol Version 2 (CSIV2) transport settings. However, when you stop the catalog server you must provide a client properties file on the **stopOgServer** command. Set the following properties in the <SAMPLES_HOME>/security_extauth/client3.props file to match the values specified above in the catServer3.props file:

```
#contextProvider=IBMJSSE2
#protocol=SSL
#keyStoreType=PKCS12
#keyStore=/was_root/IBM/WebSphere/AppServer/profiles/
<deployment_manager_name>/config/cells/<cell_name>/nodes/
<node_name>/key.p12
#keyStorePassword=WebAS
#trustStoreType=PKCS12
#trustStore=/was_root/IBM/WebSphere/AppServer/profiles/
<deployment_manager_name>/config/cells/<cell_name>/nodes/
<node_name>/trust.p12
#trustStorePassword=WebAS
```

As with the catServer3.props file, you can use the comments that are already provided in the *samples_home*/security_extauth/client3.props file with appropriate updates to *was_root*, <*deployment_manager_name*>, <*cell_name*>, and <*node_name*> variables to match your environment.

Lesson checkpoint

You configured the SSL properties for the catalog server.

< Previous | Next >
< Previous | Next >

Lesson 3.3: Run the sample

Restart all the servers and run the sample application again. You should be able to run through the steps without any problems.

See Lesson 2.4: Install and run the sample for more information about running and installing the sample application.
< Previous | Next >
< Previous | Next >

Module 4: Use Java Authentication and Authorization Service (JAAS) authorization in WebSphere Application Server

Now that you have configured authentication for clients, you can further configure authorization to give different users varying permissions. For example, an "operator" user might only be able to view data, while a "manager" user can perform all operations.

After authenticating a client, as in the previous module in this tutorial, you can give security privileges through eXtreme Scale authorization mechanisms. The previous module of this tutorial demonstrated how to enable authentication for a data grid using integration with WebSphere® Application Server. As a result, no unauthenticated client can connect to the eXtreme Scale servers or submit requests to your system. However, every authenticated client has the same permission or privileges to the server, such as reading, writing, or deleting data that is stored in the ObjectGrid maps. Clients can also issue any type of query. This part of the tutorial demonstrates how to use eXtreme Scale authorization to give authenticated users varying privileges. WebSphere eXtreme Scale uses a permission-based authorization mechanism. You can assign different permission categories that are represented by different permission classes. This module features the MapPermission class. For a list of all possible permissions, see Client authorization programming.

In WebSphere eXtreme Scale, the com.ibm.websphere.objectgrid.security.MapPermission class represents permissions to the eXtreme Scale resources, specifically the methods of the ObjectMap or JavaMap interfaces. WebSphere eXtreme Scale defines the following permission strings to access the methods of ObjectMap and JavaMap:

- **read**: Grants permission to read the data from the map.
- **write**: Grants permission to update the data in the map.
- **insert**: Grants permission to insert the data into the map.
- **remove**: Grants permission to remove the data from the map.
- **invalidate**: Grants permission to invalidate the data from the map.
- **all**: Grants all permissions to read, write, insert, remote, and invalidate.

The authorization occurs when an eXtreme Scale client uses a data access API, such as the ObjectMap ,JavaMap, or EntityManager APIs. The run time checks corresponding map permissions when the method is called. If the required permissions are not granted to the client, an AccessControlException exception results. This tutorial demonstrates how to use Java Authentication and Authorization Service (JAAS) authorization to grant authorization map access for different users.

Learning objectives

After completing the lessons in this module, you know how to:

- Enable authorization for WebSphere eXtreme Scale.
- Enable user-based authorization.

Time required

This module takes approximately 60 minutes.

Lessons in this module

- **Lession 4.1: Enable WebSphere eXtreme Scale authorization**

  To enable authorization in WebSphere eXtreme Scale, you must enable security on a specific ObjectGrid.
- **Lesson 4.2: Enable user-based authorization**

  In the authentication module of this tutorial, you created two users: `operator` and `manager`. You can assign varying permissions to these users with Java Authentication and Authorization Service (JAAS) authorization.

Lession 4.1: Enable WebSphere eXtreme Scale authorization

To enable authorization in WebSphere® eXtreme Scale, you must enable security on a specific ObjectGrid.

To enable authorization on the ObjectGrid, you must set the securityEnabled attribute to true for that particular ObjectGrid in the XML file. For this tutorial, you can either use the XSDeployment_sec.ear file from the *samples_home*/WASSecurity directory, which has already has security set in the objectGrid.xml file, or you can edit the existing objectGrid.xml file to enable security. This lesson demonstrates how to edit the file to enable security.

1. Optional: Extract the files in the XSDeployment.ear file, and then unzip the XSDeploymentWeb.war file.
2. Optional: Open the objectGrid.xml file and set the securityEnabled attribute to `true` on the ObjectGrid level. See an example of this attribute in the following example:

```
<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="Grid" txTimeout="15" securityEnabled="true">
                                    <backingMap name="Map1" />
        </objectGrid>
    </objectGrids>

</objectGridConfig>
```

   If you have multiple ObjectGrids defined, then you must set this attribute on each grid.
3. Optional: Repackage the XSDeploymentWeb.war and XSDeployment.ear files to include your changes.
4. Required: Uninstall the XSDeployment.ear file and then install the updated XSDeployment.ear. You can either use the file you modified in the previous steps, or you can install the XSDeployment_sec.ear file that is provided in the *samples_home*/WASSecurity directory. See Lesson 2.4: Install and run the sample for more information about installing the application.
5. Restart all of the application servers to enable WebSphere eXtreme Scale authorization.

Lesson checkpoint

You enabled security on the ObjectGrid, which also enables authorization on the data grid.

< Previous | Next >
< Previous | Next >

Lesson 4.2: Enable user-based authorization

In the authentication module of this tutorial, you created two users: `operator` and `manager`. You can assign varying permissions to these users with Java™ Authentication and Authorization Service (JAAS) authorization.

Defining the Java Authentication and Authorization Service (JAAS) authorization policy using user principals

You can assign permissions to the users that you previously created. Assign the `operator` user only read permissions to all maps. Assign the `manager` user all permissions. Use the JAAS authorization policy file to grant permissions to principals.
Edit the JAAS authorization file. The xsAuth3.policy file is in the *samples_home*/security_extauth directory.

```
grant codebase "http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction"
    principal javax.security.auth.x500.X500Principal
    "CN=operator,O=acme,OU=OGSample" {
    permission com.ibm.websphere.objectgrid.security.MapPermission "Grid.Map1", "read";
};

grant codebase "http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction"
    principal javax.security.auth.x500.X500Principal
    "CN=manager,O=acme,OU=OGSample" {
    permission com.ibm.websphere.objectgrid.security.MapPermission "Grid.Map1", "all";
};
```

In this file, the `http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction` codebase is a specially reserved URL for ObjectGrid. All ObjectGrid permissions that are granted to principals should use this special code base. The following permissions are assigned in this file:

- The first grant statement grants `read` map permission to the `"CN=operator,O=acme,OU=OGSample"` principal. The `"CN=operator,O=acme,OU=OGSample"` user has only map read permission to the Map1 map the Grid ObjectGrid instance.
- The second grant statement grants all map permission to the `"CN=manager,O=acme,OU=OGSample"` principal. The `"CN=manager,O=acme,OU=OGSample"` user has all permissions to the Map1 map in the Grid ObjectGrid instance.

Setting the JAAS authorization policy file using JVM properties

Use the following steps to set JVM properties for the xs1 and xs2 servers, which are in the xsCluster cluster. If you are using a topology that is different from the sample topology that is used in this tutorial, set the file on all of your container servers.

1. In the administrative console, click Servers > Application servers > *server_name* > Java and process management > Process definition > Java virtual machine.
2. Add the following generic JVM arguments:

   ```
   -Djava.security.policy=samples_home/security_extauth/xsAuth3.policy
   ```

3. Click OK and save your changes.

Running the sample application to test authorization

You can use the sample application to test the authorization settings. The manager user continues to have all permissions in the Map1 map, including displaying and adding employees. The operator user should only be able to view employees because that user was assigned read permission only.

1. Restart all of the application servers that are running container servers. For this tutorial, restart the `xs1` and `xs2` servers.
2. Open the `EmployeeManagementWeb` application. In a web browser, open `http://<host>:<port>/EmployeeManagermentWeb/management.jsp`.
3. Log in to the application using any valid user name and password.
4. Attempt to display an employee. Click Display an Employee and search for the `authemp1@acme.com` email address. A message displays that the user cannot be found.
5. Add an employee. Click Add an Employee. Add the email `authemp1@acme.com`, the given name `Joe`, and the surname `Doe`. Click Submit. A message displays that the employee has been added.
6. Edit the *samples_home*/security_extauth/client3.props file. Change the value of credentialGeneratorProps property from `manager manager1` to `operator operator1`. After you edit the file, the servlet uses user name "operator" and password "operator1" to authenticate to the WebSphere eXtreme Scale servers.
7. Restart the appCluster cluster to pick up the changes in the *samples_home*/security_extauth/client3.props file.
8. Attempt to display an employee. Click Display an Employee and search for the `authemp1@acme.com` email address. The employee is displayed.
9. Add an employee. Click Add an Employee. Add the email `authemp2@acme.com`, the given name `Joe`, and the surname `Doe`. Click Submit. The following message displays:

   ```
   An exception occurs when Add the employee. See below for detailed exception messages.
   ```

   The detailed exception text follows:

   ```
   java.security.AccessControlException: Access denied
   (com.ibm.websphere.objectgrid.security.MapPermission Grid.Map1 insert)
   ```

   This message displays because the `operator` user does not have permission to insert data into the Map1 map.

If you are running with a version of WebSphere Application Server that is earlier than Version 7.0.0.11, you might see a java.lang.StackOverflowError error on the container server. This error is caused by a problem with the IBM Developer Kit. The problem is fixed in the IBM Developer Kit that is shipped with WebSphere Application Server Version 7.0.0.11 and later.

Lesson checkpoint

In this lesson, you configured authorization by assigning permissions to specific users.

< Previous | Next >
< Previous | Next >

Module 5: Use the xscmd utility to monitor data grids and maps

You can use the **xscmd** utility to show the primary data grids and map sizes of the `Grid` data grid. The **xscmd** tool uses the MBean to query all of the data grid artifacts, such as primary shards, replica shards, container servers, map sizes, and other data.

In this tutorial, the catalog server is running as a stand-alone Java SE server. The container servers are running in WebSphere® Application Server application servers.
For the catalog server, a MBean server is created in the stand-alone Java virtual machine (JVM). When you use the **xscmd** tool on the catalog server, WebSphere eXtreme Scale security is used.

For the container servers, the WebSphere eXtreme Scale run time registers the Managed Beans (MBean) with the MBean server that is created by the WebSphere Application Server run time. The security that is used by the **xscmd** tool is provided by the WebSphere Application Server MBean security.

1. Using a command-line tool, open the *DMGR_PROFILE*/bin directory.
2. Run the **xscmd** tool. Use the -c showPlacement -st P parameters as in the following examples:

   Linux UNIX

   ```
   xscmd.sh -c showPlacement -cep localhost:16099 -g Grid -ms mapSet -sf P
   -user manager -pwd manager1
   ```

   Windows

   ```
   xscmd.bat -c showPlacement -cep localhost:16099 -g Grid -m mapSet -sf P
   -user manager -pwd manager1
   ```

8.5+ Attention:
If you use the following command to access the data grid, you might also be authorized to perform administrative actions, such as listAllJMXAddresses:

```
./xscmd.sh -user <user> -password <password> <other_parameters>
```

If this operation works for this user, then any **xscmd** operation might also be performed by the same user. For more information, see Troubleshooting security
The user name and password are passed to the catalog server for authentication.

3. View the command results.

```
*** Showing all primaries for grid - Grid & mapset - mapSet
Partition Container Host Server
0 myCell02\myNode04\xs2_C-1 myhost.mycompany.com myCell02\myNode04\xs2
1 myCell02\myNode04\xs2_C-1 myhost.mycompany.com myCell02\myNode04\xs2
2 myCell02\myNode04\xs2_C-1 myhost.mycompany.com myCell02\myNode04\xs2
3 myCell02\myNode04\xs2_C-1 myhost.mycompany.com myCell02\myNode04\xs2
4 myCell02\myNode04\xs2_C-1 myhost.mycompany.com myCell02\myNode04\xs2
```

4. Run the **xscmd** tool. Use the -c showMapSizes parameter as in the following examples:
Linux UNIX

```
xscmd.sh -c showMapSizes -cep localhost:16099 -g Grid -ms mapSet  -user manager -pwd manager1
```

Windows

```
xscmd.bat -c showMapSizes -cep localhost:16099 -g Grid -ms mapSet -user manager -pwd manager1
```

The user name and password are passed to the catalog server for authentication. After you run the command, you are prompted for the WebSphere Application Server user ID and password to authenticate to WebSphere Application Server. You must provide this login information because the -c showMapSizes option gets the map size from each container server, which requires the WebSphere Application Server security.

5. Optional: You can change the PROFILE/properties/sas.client.props file to run the command without the user ID and password being required. Change the com.ibm.CORBA.loginSource property from `prompt` to `properties` and then provide the user ID and password. An example of the properties in the PROFILE/properties/sas.client.props file follows:

```
com.ibm.CORBA.loginSource=properties
# RMI/IIOP user identity
com.ibm.CORBA.loginUserid=Admin
com.ibm.CORBA.loginPassword=xxxxxx
```

6. Optional: If you are using the **xscmd** command on a WebSphere eXtreme Scale stand-alone installation, then you must add the following options:
   - If you are using WebSphere eXtreme Scale security:

     ```
     -user
     -pwd
     ```

   - If you are using WebSphere eXtreme Scale security with custom credential generation:

     ```
     -user
     -pwd
     -cgc
     -cgp
     ```

   - If SSL is enabled:

     ```
     -tt
     -cxpv
     -prot
     -ks
     -ksp
     -kst
     -ts
     -tsp
     -tst
     ```

If WebSphere eXtreme Scale security and SSL are both enabled, then both set of parameters are required.

Related tasks:

Monitoring with the xscmd utility
Administering with the xscmd utility

Lesson checkpoint

You used the **xscmd** tool to monitor data grids and maps in your configuration.

< Previous | Next >
< Previous | Next >

Tutorial: Running eXtreme Scale bundles in the OSGi framework

The OSGi sample builds on the Google Protocol Buffers serializer samples. When you complete this set of lessons, you will have run the serializer sample plug-ins in the OSGi framework.

Learning objectives

This sample demonstrates the OSGi bundles. The serializer plug-in is incidental and is not required. The OSGi sample is available on the WebSphere eXtreme Scale samples gallery. You must download the sample, and extract it into the *wxs_home*/samples directory. The root directory for the OSGi sample is wxs_home/samples/OSGiProto.

The command examples in this tutorial assume that you are running on the UNIX operating system. You must adjust the command example to run on a Windows operating system.

After completing the lessons in this tutorial, you will understand the OSGi sample concepts and know how to complete the following objectives:

- Install the WebSphere® eXtreme Scale server bundle into the OSGi container to start the eXtreme Scale server.
- Set up your eXtreme Scale development environment to run the sample client.
- Use the xscmd command to query the service ranking of the sample bundle, upgrade it to a new service ranking, and verify the new service ranking.

Time required

This module takes approximately 60 minutes to complete.

Prerequisites

In addition to downloading and extracting the serializer samples, this tutorial also has the following prerequisites:

- Install and extract the eXtreme Scale product
- Set up the Eclipse Equinox Environment
- **Introduction: Starting and configuring the eXtreme Scale server and container to run plug-ins in the OSGi framework**

  In this tutorial you start an eXtreme Scale server in the OSGi framework, start an eXtreme Scale container, and wire the sample plug-ins with eXtreme Scale runtime environment.
- **Module 1: Preparing to install and configure eXtreme Scale server bundles**

  Complete this module to explore OSGi sample bundles and examine configuration files that you use to configure the eXtreme Scale server.
- **Module 2: Installing and starting eXtreme Scale bundles in the OSGi framework**

  Use the lessons in this module to install the eXtreme Scale server bundle into the OSGi container, and start the WebSphere eXtreme Scale server.
- **Module 3: Running the eXtreme Scale sample client**

  The WebSphere eXtreme Scale server is now running in an OSGi environment. Complete the steps in this module to run an WebSphere eXtreme Scale client that inserts data into the grid.
- **Module 4: Querying and upgrading the sample bundle**

  Complete the lessons in this module to use the **xscmd** command to query the service ranking of the sample bundle, upgrade it to a new service ranking, and verify the new service ranking.

< Previous | Next >
< Previous | Next >

Introduction: Starting and configuring the eXtreme Scale server and container to run plug-ins in the OSGi framework

In this tutorial you start an eXtreme Scale server in the OSGi framework, start an eXtreme Scale container, and wire the sample plug-ins with eXtreme Scale runtime environment.

Learning objectives

After completing the lessons in this tutorial you will understand the OSGi sample concepts and know how to complete the following objectives:

- Install the WebSphere® eXtreme Scale server bundle into the OSGi container to start the eXtreme Scale server.
- Set up your eXtreme Scale development environment to run the sample client.
- Use the xscmd command to query the service ranking of the sample bundle, upgrade it to a new service ranking, and verify the new service ranking.

Time required

This tutorial takes approximately 60 minutes to finish. If you explore other concepts related to this tutorial, it might take longer to complete.

Skill level

Intermediate.

Audience

Developers and administrators who want to build, install, and run eXtreme Scale bundles into the OSGi framework.

System requirements

- Luminis OSGi Configuration Admin command line client, version 0.2.5
- Apache Felix File Install, version 3.0.2
- When using Eclipse Gemini as the Blueprint container provider, the following are required:
  - Eclipse Gemini Blueprint, version 1.0.0
  - Spring Framework, version 3.0.5
  - SpringSource AOP Alliance API, version 1.0.0
  - SpringSource Apache Commons Logging, version 1.1.1
- When using Apache Aries as the Blueprint Container provider, you must have the following requirements:
  - Apache Aries, latest snapshot
  - ASM library
  - PAX logging

Prerequisites

To complete this tutorial, you must download the sample, and extracted it into the wxs_home/samples directory. The root directory for the OSGi sample is wxs_home/samples/OSGiProto.

Expected results

When you complete this tutorial, you will have installed the sample bundles and run an eXtreme Scale client to insert data into the grid. You can also expect to query and update those sample bundles using the dynamic capabilities that the OSGi container provides.

Modules in this tutorial

- **Module 1: Preparing to install and configure eXtreme Scale server bundles**

  Complete this module to explore OSGi sample bundles and examine configuration files that you use to configure the eXtreme Scale server.
- **Module 2: Installing and starting eXtreme Scale bundles in the OSGi framework**

  Use the lessons in this module to install the eXtreme Scale server bundle into the OSGi container, and start the WebSphere eXtreme Scale server.
- **Module 3: Running the eXtreme Scale sample client**

  The WebSphere eXtreme Scale server is now running in an OSGi environment. Complete the steps in this module to run an WebSphere eXtreme Scale client that inserts data into the grid.
- **Module 4: Querying and upgrading the sample bundle**

  Complete the lessons in this module to use the **xscmd** command to query the service ranking of the sample bundle, upgrade it to a new service ranking, and verify the new service ranking.

Related concepts:

OSGi framework overview
Related tasks:

Installing the Eclipse Equinox OSGi framework with Eclipse Gemini for clients and servers
Related reference:

Server properties file
< Previous | Next >
< Previous | Next >

Module 1: Preparing to install and configure eXtreme Scale server bundles

Complete this module to explore OSGi sample bundles and examine configuration files that you use to configure the eXtreme Scale server.

Learning objectives

After completing the lessons in this module, you will understand the concepts and know how to complete the following objectives:

- Locate and explore the bundles that are included in the OSGi sample.
- Examine configuration files that are used to configure the eXtreme Scale grid and server.

Lessons in this module

- **Lesson 1.1: Understand the OSGi sample bundles**

  Complete this lesson to locate and explore the bundles that are provided in the OSGi sample.
- **Lesson 1.2: Understand the OSGi configuration files**

  The OSGi sample includes configuration files that you use to start and configure the WebSphere® eXtreme Scale grid and server.

< Previous | Next >
< Previous | Next >

Lesson 1.1: Understand the OSGi sample bundles

Complete this lesson to locate and explore the bundles that are provided in the OSGi sample.

OSGi sample bundles

Other than the bundles that are configured in the config.ini file, which is shown in the topic about setting up the Eclipse Equinox environment, the following additional bundles are used in the OSGi sample:

objectgrid.jar
: The WebSphere eXtreme Scale server runtime bundle. This bundle is located in the *wxs_home*/lib directory.

com.google.protobuf_2.4.0a.jar
: The Google Protocol Buffers, version 2.4.0a bundle. This bundle is located in the *wxs_sample_osgi_root*/lib directory.

ProtoBufSamplePlugins-1.0.0.jar
: Version 1.0.0 of the user plug-in bundle with sample ObjectGridEventListener and MapSerializerPlugin plug-in implementations. This bundle is located in the *wxs_sample_osgi_root*/lib directory. The services are configured with service ranking 1.
This version uses the standard Blueprint XML to configure the eXtreme Scale plug-in services. The service class is a user-implemented class for WebSphere eXtreme Scale interface, com.ibm.websphere.objectgrid.plugins.osgi.PluginServiceFactory. The user-implemented class creates a bean for each request and works similar to a prototype-scoped bean.

ProtoBufSamplePlugins-2.0.0.jar
: Version 2.0.0 of the user plug-in bundle with sample ObjectGridEventListener and MapSerializerPlugin plug-in implementations. This bundle is located in the *wxs_sample_osgi_root*/lib directory. The services are configured with service ranking 2.
This version uses the standard Blueprint XML to configure the eXtreme Scale plug-in services. The service class is using a WebSphere eXtreme Scale, built-in class, com.ibm.websphere.objectgrid.plugins.osgi.PluginServiceFactoryImpl, which uses the BlueprintContainer service. Using the standard Blueprint XML configuration, the beans can be configured either as a prototype scope or singleton scope. The bean is not configured as a shard scope.

ProtoBufSamplePlugins-Gemini-3.0.0.jar
: Version 3.0.0 of the user plug-in bundle with sample ObjectGridEventListener and MapSerializerPlugin plug-in implementations. This bundle is located in the *wxs_sample_osgi_root*/lib directory. The services are configured with service ranking 3.
This version uses the Eclipse Gemini-specific Blueprint XML to configure the eXtreme Scale plug-in services. The service class is using a WebSphere eXtreme Scale built-in class, com.ibm.websphere.objectgrid.plugins.osgi.PluginServiceFactoryImpl, which uses the BlueprintContainer service. The way to configure a shard scope bean is using a Gemini-specific approach. This version configures the myShardListener bean as a shard scope bean by providing {http://www.ibm.com/schema/objectgrid}shard as the scope value, and configuring a dummy attribute so that the custom scope is recognized by Gemini. This is due to the following Eclipse issue: https://bugs.eclipse.org/bugs/show_bug.cgi?id=348776

ProtoBufSamplePlugins-Aries-4.0.0.jar
: Version 4.0.0 of the user plug-in bundle with sample ObjectGridEventListener and MapSerializerPlugin plug-in implementations. This bundle is located in the *wxs_sample_osgi_root*/lib directory. The services are configured with service ranking 4.
This version uses standard Blueprint XML to configure the eXtreme Scale plug-in services. The service class is using a WebSphere eXtreme Scale, built-in class, com.ibm.websphere.objectgrid.plugins.osgi.PluginServiceFactoryImpl, which uses the BlueprintContainer service. Using the standard Blueprint XML configuration, the beans can be configured using a custom scope. This version configures the myShardListenerbean as a shard scoped bean by providing {http://www.ibm.com/schema/objectgrid}shard as the scope value.

ProtoBufSamplePlugins-Activator-5.0.0.jar
: Version 5.0.0 of the user plug-in bundle with sample ObjectGridEventListener and MapSerializerPlugin plug-in implementations. This bundle is located in the *wxs_sample_osgi_root*/lib directory. The services are configured with service ranking 5.
This version does not use Blueprint container at all. In this version, the services are registered using OSGi service registration. The service class is a user-implemented class for the WebSphere eXtreme Scale interface, com.ibm.websphere.objectgrid.plugins.osgi.PluginServiceFactory. The user-implemented class creates a bean for each request. It works similar to a prototype-scoped bean.

Lesson checkpoint

By exploring the bundles that are provided with the OSGi sample, you can better understand how to develop your own implementations that will run in the OSGi container.

You learned:

- About bundles that included with the OSGi sample
- The location of those bundles
- The service ranking that each bundle has been configured with

< Previous | Next >
< Previous | Next >

Lesson 1.2: Understand the OSGi configuration files

The OSGi sample includes configuration files that you use to start and configure the WebSphere® eXtreme Scale grid and server.

OSGi configuration files

In this lesson, you will explore the following configuration files that are included with the OSGI sample:

- collocated.server.properties
- protoBufObjectGrid.xml
- protoBufDeployment.xml
- blueprint.xml

collocated.server.properties

A server configuration is required to start a server. When the eXtreme Scale server bundle is started, it does not start a server. It waits for the configuration PID, `com.ibm.websphere.xs.server`, to be created with a server property file. This server property file specifies the server name, port number, and other server properties.
In most cases, you create a configuration to set the server property file. In rare cases, you might want only to start a server, with every property set to a default value. In that case, you can create a configuration called `com.ibm.websphere.xs.server` with value set to default.

For more details about the server property file, see the Server properties file topic.

The OSGi sample server properties file starts a single catalog. This sample property file starts a single catalog service and a container server in the OSGi framework process. eXtreme Scale clients connect to port 2809 and JMX clients connect to port 1099. The content of the sample server property file is:

```
serverName=collocatedServer
isCatalog=true
catalogClusterEndPoints=collocatedServer:localhost:6601:6602
traceSpec=ObjectGridOSGi=all=enabled
traceFile=logs/trace.log
listenerPort=2809
JMXServicePort=1099
```

protoBufObjectGrid.xml

The sample protoBufObjectGrid.xml ObjectGrid descriptor XML file contains the following content, with comments removed.

```
<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="Grid" txTimeout="15">

            <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsosgiconfigfiles_ObjectGridEventLis
tener"
                osgiService="myShardListener"/>

            <backingMap name="Map" readOnly="false"
                lockStrategy="PESSIMISTIC" lockTimeout="5"
                copyMode="COPY_TO_BYTES"
                pluginCollectionRef="serializer"/>

        </objectGrid>
    </objectGrids>

    <backingMapPluginCollections>
        <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsosgiconfigfiles_serializer">
            <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsosgiconfigfiles_MapSerializerPlugi
n"
                osgiService="myProtoBufSerializer"/>"/>
        </backingMapPluginCollection>
    </backingMapPluginCollections>
        </objectGridConfig>
```

There are two plug-ins configured in this ObjectGrid descriptor XML file:

ObjectGridEventListener
: The shard-level plug-in. For each ObjectGrid instance, there is an instance of ObjectGridEventListener. It is configured to use the OSGi service myShardListener. That means when the grid is created, the ObjectGridEventListener plug-in uses the OSGi service myShardListener with the highest service ranking available.

MapSerializerPlugin
: The map-level plug-in. For the backing map named`Map`, there is a MapSerializerPlugin plug-in configured. It is configured to use the OSGI service myProtoBufSerializer. That means when the map is created, the MapSerializerPlugin plug-in uses the service, myProtoBufSerializer, with the highest ranked service ranking available.

protoBufDeployment.xml

The deployment descriptor XML file describes the deployment policy for the grid named `Grid`, which uses five partitions. See the following code example of the XML file:

```
<deploymentPolicy
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy ../deploymentPolicy.xsd"
        xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">

  <objectgridDeployment objectgridName="Grid">
    <mapSet name="MapSet" numberOfPartitions="5">
      <map ref="Map"/>
    </mapSet>
  </objectgridDeployment>
</deploymentPolicy>
```

blueprint.xml

As an alternative to using the collocated.server.properties file in conjunction with configuration PID, `com.ibm.websphere.xs.server`, you can include the ObjectGrid XML and deployment XML files in an OSGi bundle, along with a Blueprint XML file as shown in the following example:

```
<blueprint
         xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"

xmlns:objectgrid="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsosgiconfigfiles_http
://www.ibm.com/schema/objectgrid"
       default-activation="lazy">

    <objectgrid:server
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsosgiconfigfiles_server"
isCatalog="true"
                name="server"
                tracespec="ObjectGridOSGi=all=enabled"
                tracefile="C:/Temp/logs/trace.log"
                workingDirectory="C:/Temp/working"
                jmxport="1099">
        <objectgrid:catalog host="localhost" port="2809"/>
    </objectgrid:server>

    <objectgrid:container
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsosgiconfigfiles_container"
        objectgridxml="/META-INF/objectgrid.xml"
       deploymentxml="/META-INF/deployment.xml"
         server="server"/>
</blueprint>
```

Lesson checkpoint

In this lesson, you learned about the configuration files that are used in the OSGi sample. Now, when you start and configure the eXtreme Scale grid and server, you will understand which files are being used in these processes and how these files interact with your plug-ins in the OSGi framework.

< Previous | Next >
< Previous | Next >

Module 2: Installing and starting eXtreme Scale bundles in the OSGi framework

Use the lessons in this module to install the eXtreme Scale server bundle into the OSGi container, and start the WebSphere® eXtreme Scale server.

Starting the server in the OSGi framework does not mean that your OSGi bundles are ready to run. You must configure the server properties and containers so that the OSGi bundles that you install are recognized and can run correctly.

Learning objectives

After completing the lessons in this module, you will understand the concepts and know how to complete the following tasks:

- Install eXtreme Scale bundles using the Equinox OSGi console.
- Configure the eXtreme Scale server.
- Configure the eXtreme Scale container.
- Install and start eXtreme Scale sample bundles.

Prerequisites

To complete this module, the following tasks are required before you begin:

- Install and extract the eXtreme Scale product
- Set up the Eclipse Equinox Environment

You must also prepare to access the following files to complete the lessons in this module:

- objectgrid.jar bundle. You install this eXtreme Scale bundle.

- collocated.server.properties file. You add the server properties to this configuration file.

You can expect to install and start the following bundles:

- protobuf-java-2.4.0a-bundle.jar bundle
- ProtoBufSamplePlugins-1.0.0.jar bundle

Lessons in this module

- **Lesson 2.1: Start the console and install the eXtreme Scale server bundle**

  In this lesson, you use the Equinox OSGi console to install the WebSphere eXtreme Scale server bundle.
- **Lesson 2.2: Customize and configure the eXtreme Scale server**

  Use this lesson to customize and add the server properties to the WebSphere eXtreme Scale server.
- **Lesson 2.3: Configure the eXtreme Scale container**

  Complete this lesson to configure a container, which includes the WebSphere eXtreme Scale ObjectGrid descriptor XML file and ObjectGrid deployment XML file. These files include the configuration for the grid and its topology.
- **Lesson 2.4: Install the Google Protocol Buffers and sample plug-in bundles**

  Complete this tutorial to install the protobuf-java-2.4.0a-bundle.jar bundle and the ProtoBufSamplePlugins-1.0.0.jar plug-in bundle using the Equinox OSGi console.
- **Lesson 2.5: Start the OSGi bundles**

  The WebSphere eXtreme Scale server is packaged as an OSGi server bundle. Complete this lesson to install the eXtreme Scale server bundle as well as other OSGi bundles that you have installed.

< Previous | Next >
< Previous | Next >

Lesson 2.1: Start the console and install the eXtreme Scale server bundle

In this lesson, you use the Equinox OSGi console to install the WebSphere® eXtreme Scale server bundle.

1. Use the following command to start the Equinox OSGi console:

   ```
   cd equinox_root
   java -jar plugins\org.eclipse.osgi_3.6.1.R36x_v20100806.jar -console
   ```

2. After the OSGi console is started, issue the `ss` command in the console, and the following bundles are started:
   Attention: If you completed the task, Installing eXtreme Scale bundles, then the bundle has already been activated. If the bundle is started, then stop the bundle before you complete this step.

   ```
    Eclipse Gemini output:
   osgi> ss
   Framework is launched.
   id State Bundle
   0 ACTIVE org.eclipse.osgi_3.6.1.R36x_v20100806
   1 ACTIVE org.eclipse.osgi.services_3.2.100.v20100503
   2 ACTIVE org.eclipse.osgi.util_3.2.100.v20100503
   3 ACTIVE org.eclipse.equinox.cm_1.0.200.v20100520
   4 ACTIVE com.springsource.org.apache.commons.logging_1.1.1
   5 ACTIVE com.springsource.org.aopalliance_1.0.0
   6 ACTIVE org.springframework.aop_3.0.5.RELEASE
   7 ACTIVE org.springframework.asm_3.0.5.RELEASE
   8 ACTIVE org.springframework.beans_3.0.5.RELEASE
   9 ACTIVE org.springframework.context_3.0.5.RELEASE
   10 ACTIVE org.springframework.core_3.0.5.RELEASE
   11 ACTIVE org.springframework.expression_3.0.5.RELEASE
   12 ACTIVE org.apache.felix.fileinstall_3.0.2
   13 ACTIVE net.luminis.cmc_0.2.5
   14 ACTIVE org.eclipse.gemini.blueprint.core_1.0.0.RELEASE
   15 ACTIVE org.eclipse.gemini.blueprint.extender_1.0.0.RELEASE
   16 ACTIVE org.eclipse.gemini.blueprint.io_1.0.0.RELEASE
   ```

   ```
    Apache Aries output:
   osgi> ss
   Framework is launched.
   id State Bundle
   0 ACTIVE org.eclipse.osgi_3.6.1.R36x_v20100806
   1 ACTIVE org.eclipse.osgi.services_3.2.100.v20100503
   2 ACTIVE org.eclipse.osgi.util_3.2.100.v20100503
   3 ACTIVE org.eclipse.equinox.cm_1.0.200.v20100520
   4 ACTIVE org.ops4j.pax.logging.pax-logging-api_1.6.3
   5 ACTIVE org.ops4j.pax.logging.pax-logging-service_1.6.3
   6 ACTIVE org.objectweb.asm.all_3.3.0
   7 ACTIVE org.apache.aries.blueprint_0.3.2.SNAPSHOT
   8 ACTIVE org.apache.aries.util_0.4.0.SNAPSHOT
   9 ACTIVE org.apache.aries.proxy_0.4.0.SNAPSHOT
   ```

```
10 ACTIVE org.apache.felix.fileinstall_3.0.2
11 ACTIVE net.luminis.cmc_0.2.5
```

3. Install the objectgrid.jar bundle. To start a server in the Java™ virtual machine (JVM), you need to install an eXtreme Scale server bundle. This eXtreme Scale server bundle can start a server and create containers. Use the following command to install the objectgrid.jar file:

   ```
   osgi> install file:///wxs_home/lib/objectgrid.jar
   ```

   See the following example:

   ```
   osgi> install file:///opt/wxs/ObjectGrid/lib/objectgrid.jar
   ```

   Equinox displays its bundle ID; for example:

   ```
   Bundle id is 19
   ```

   Remember: Your bundle ID might be different. The file path must be an absolute URL to the bundle path. Relative paths are not supported.

Lesson checkpoint

In this lesson, you used the Equinox OSGi console to install the objectgrid.jar bundle, which you will use to start a server and create a container later in this tutorial.

< Previous | Next >
< Previous | Next >

Lesson 2.2: Customize and configure the eXtreme Scale server

Use this lesson to customize and add the server properties to the WebSphere® eXtreme Scale server.

1. Edit the wxs_sample_osgi_root/projects/server/properties/collocated.server.properties file.
   a. Change the traceFile property to `equinox_root/logs/trace.log`.
2. Save the file.
3. Enter the following lines of code in the OSGI console to create the server configuration from the file. The following example is displayed on multiple lines for publication purposes.

   ```
   osgi> cm create com.ibm.websphere.xs.server
   osgi> cm put com.ibm.websphere.xs.server objectgrid.server.props
   wxs_sample_osgi_root/projects/server/properties/collocated.server.properties
   ```

4. To view the configuration, run the following command:

   ```
   osgi> cm get com.ibm.websphere.xs.server
   Configuration for service (pid) "com.ibm.websphere.xs.server"
   (bundle location = null)
   key                    value
   ----                   ----
   objectgrid.server.props  wxs_sample_osgi_root/projects/server/properties/collocated.server.properties
   service.pid              com.ibm.websphere.xs.server
   ```

Lesson checkpoint

In this lesson, you edited the wxs_sample_osgi_root/projects/server/properties/collocated.server.properties file to specify server settings, such as the working directory and the location for the trace log files.

< Previous | Next >
< Previous | Next >

Lesson 2.3: Configure the eXtreme Scale container

Complete this lesson to configure a container, which includes the WebSphere® eXtreme Scale ObjectGrid descriptor XML file and ObjectGrid deployment XML file. These files include the configuration for the grid and its topology.

To create a container, first create a configuration service using the managed service factory process identification number (PID), com.ibm.websphere.xs.container. The service configuration is a managed service factory, so you can create multiple service PIDs from the factory PID. Then, to start the container service, set the objectgridFile and deploymentPolicyFile PIDs to each service PID.

Complete the following steps to customize and add the server properties to the OSGi framework:

1. In the OSGI console, enter the following command to create the container from the file:

   ```
   osgi> cm createf com.ibm.websphere.xs.container
   PID: com.ibm.websphere.xs.container-1291179621421-0
   ```

2. Enter the following commands to bind the newly created PID to the ObjectGrid XML files.
   Remember: The PID number will be different from what is included in this example.

```
osgi> cm put com.ibm.websphere.xs.container-1291179621421-0 objectgridFile
wxs_sample_osgi_root/projects/server/META-INF/protoBufObjectgrid.xml

osgi> cm put com.ibm.websphere.xs.container-1291179621421-0 deploymentPolicyFile
wxs_sample_osgi_root/projects/server/META-INF/protoBufDeployment.xml
```

3. Use the following command to display the configuration:

```
osgi> cm get com.ibm.websphere.xs.container-1291760127968-0
Configuration for service (pid) "com.ibm.websphere.xs.container-1291760127968-0"
(bundle location = null)

key                    value
------                 ------
deploymentPolicyFile   /opt/wxs/ObjectGrid/samples/OSGiProto/server/META-INF/protoBufDeployment.xml
objectgridFile         /opt/wxs/ObjectGrid/samples/OSGiProto/server/META-INF/protoBufObjectgrid.xml
service.factoryPid     com.ibm.websphere.xs.container
service.pid            com.ibm.websphere.xs.container-1291760127968-0
```

Lesson checkpoint

In this lesson, you created a configuration service, which you used to create an eXtreme Scale container. Since the ObjectGrid XML files contain the configuration for the grid and its topology, you had to bind the container that you created to those ObjectGrid XML files. With this configuration, the eXtreme Scale container can recognize the OSGi bundles that you will run later in this tutorial.

< Previous | Next >
< Previous | Next >

Lesson 2.4: Install the Google Protocol Buffers and sample plug-in bundles

Complete this tutorial to install the protobuf-java-2.4.0a-bundle.jar bundle and the ProtoBufSamplePlugins-1.0.0.jar plug-in bundle using the Equinox OSGi console.

Install the Google Protocol Buffers plug-in

Complete the following steps to install the Google Protocol Buffers plug-in.
In the OSGI console, enter the following command to install the plug-in:

```
osgi> install file:///wxs_sample_osgi_root/lib/com.google.protobuf_2.4.0a.jar
```

The following output is displayed:

```
Bundle ID is 21
```

Sample plug-in bundles overview

The OSGi sample includes five sample bundles that include eXtreme Scale plug-ins, including a custom ObjectGridEventListener and MapSerializerPlugin plug-in. The MapSerializerPlugin plug-in uses the Google Protocol Buffers sample and messages provided by the MapSerializerPlugin sample.

The following bundles are located in *wxs_sample_osgi_root*/lib directory: ProtoBufSamplePlugins-1.0.0.jar and the ProtoBufSamplePlugins-2.0.0.jar.

The blueprint.xml file has the following content with comments removed:

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">
        <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsinstallgoogleOSGi_myShardListener"
class="com.ibm.websphere.samples.xs.proto.osgi.MyShardListenerFactory"/>
        <service ref="myShardListener" interface="com.ibm.websphere.objectgrid.plugins.osgi.PluginServiceFactory"
ranking="1">
        </service>

        <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsinstallgoogleOSGi_myProtoBufSerial
izer" class="com.ibm.websphere.samples.xs.proto.osgi.ProtoMapSerializerFactory">
                <property name="keyType" value="com.ibm.websphere.samples.xs.serializer.app.proto.DataObjects1$OrderKey"
/>
                <property name="valueType" value="com.ibm.websphere.samples.xs.serializer.app.proto.DataObjects1$Order"
/>
        </bean>

        <service ref="myProtoBufSerializer" interface="com.ibm.websphere.objectgrid.plugins.osgi.PluginServiceFactory"
                ranking="1">
        </service>
</blueprint>
```

The Blueprint XML file exports two services, myShardListener and myProtoBufSerializer. These two services are referenced in the protoBufObjectgrid.xml file.

Install the sample plug-in bundle

Complete the following steps to install the ProtoBufSamplePlugins-1.0.0.jar bundle.

Run the following command in the Equinox OSGi console to install the ProtoBufSamplePlugins-1.0.0.jar plugin bundle:

```
osgi> install file:///wxs_sample_osgi_root/lib/ProtoBufSamplePlugins-1.0.0.jar
```

The following output is displayed:

```
Bundle ID is 22
```

Lesson checkpoint

In this lesson, you installed the protobuf-java-2.4.0a-bundle.jar bundle and the ProtoBufSamplePlugins-1.0.0.jar plug-in bundle.

< Previous | Next >
< Previous | Next >

Lesson 2.5: Start the OSGi bundles

The WebSphere® eXtreme Scale server is packaged as an OSGi server bundle. Complete this lesson to install the eXtreme Scale server bundle as well as other OSGi bundles that you have installed.

1. Run the **ss** command to view the IDs for each bundle.

   ```
   osgi> ss

   Framework is launched.

   id State Bundle
   0 ACTIVE org.eclipse.osgi_3.6.1.R36x_v20100806
   1 ACTIVE org.eclipse.osgi.services_3.2.100.v20100503
   2 ACTIVE org.eclipse.osgi.util_3.2.100.v20100503
   3 ACTIVE org.eclipse.equinox.cm_1.0.200.v20100520
   4 ACTIVE com.springsource.org.apache.commons.logging_1.1.1
   5 ACTIVE com.springsource.org.aopalliance_1.0.0
   6 ACTIVE org.springframework.aop_3.0.5.RELEASE
   7 ACTIVE org.springframework.asm_3.0.5.RELEASE
   8 ACTIVE org.springframework.beans_3.0.5.RELEASE
   9 ACTIVE org.springframework.context_3.0.5.RELEASE
   10 ACTIVE org.springframework.core_3.0.5.RELEASE
   11 ACTIVE org.springframework.expression_3.0.5.RELEASE
   12 ACTIVE org.apache.felix.fileinstall_3.0.2
   13 ACTIVE net.luminis.cmc_0.2.5
   15 ACTIVE org.eclipse.gemini.blueprint.core_1.0.0.RELEASE
   16 ACTIVE org.eclipse.gemini.blueprint.extender_1.0.0.RELEASE
   17 ACTIVE org.eclipse.gemini.blueprint.io_1.0.0.RELEASE
   19 RESOLVED com.ibm.websphere.xs.server_7.1.1
   21 RESOLVED Google_ProtoBuf_2.4.0
   22 RESOLVED ProtoBufPlugins_1.0.0
   ```

2. Start each bundle that you have installed. You must start the bundles in a specific order. See the order of the bundle IDs from the previous example.
   a. Start the sample plug-in bundle, ProtoBufPlugins_1.0.0. Run the following command in the Equinox OSGi console to start the bundle. In this example, the bundle ID of the sample plug-in is 22.

      ```
      osgi> start 22
      ```

   b. Start the Google Protocol Buffers bundle, Google_ProtoBuf_2.4.0. Run the following command in the Equinox OSGi console to start the bundle. In this example, the bundle ID of the Google Protocol Buffers plug-in is 21.

      ```
      osgi> start 21
      ```

   c. Start the server bundle, com.ibm.websphere.xs.server_7.1.1. Run the following command in the OSGi console to start the server. In this example, the bundle ID of the eXtreme Scale server bundle is 19.

      ```
      osgi> start 19
      ```

After you start the server, the MyShardListener event listener is started and ready to insert or update records. You can see the following output on the OSGi console to confirm that the plug-in bundle has started successfully:

```
SystemOut O MyShardListener@1253853884(version=1.0.0) order
com.ibm.websphere.samples.xs.serializer.proto.DataObjects1$Order$Builder
@1aba1aba(22) inserted
```

Lesson checkpoint

In this lesson, you started two plug-in bundles and the server bundle in the eXtreme Scale container that you configured for the OSGi framework.

< Previous | Next >
< Previous | Next >

Module 3: Running the eXtreme Scale sample client

The WebSphere® eXtreme Scale server is now running in an OSGi environment. Complete the steps in this module to run an WebSphere eXtreme Scale client that inserts data into the grid.

Learning objectives

After completing the lessons in this module you will know how to complete the following tasks:

- Run a client application that connects to the grid and inserts and retrieves some data from it.
- Start an order using a non-OSGi client application.

Prerequisites

Complete Module 2: Installing and starting eXtreme Scale bundles in the OSGi framework.

Lessons in this module

- **Lesson 3.1: Set up Eclipse to run the client and build the samples**

  Complete this lesson to import the Eclipse project that you will use to run the client and build the sample plug-ins.
- **Lesson 3.2: Start a client and insert data into the grid**

  Complete this lesson to start a non-OSGi client and run a client application.

< Previous | Next >
< Previous | Next >

Lesson 3.1: Set up Eclipse to run the client and build the samples

Complete this lesson to import the Eclipse project that you will use to run the client and build the sample plug-ins.

The sample includes a Java™ SE client program that connects to the grid and inserts and retrieves data from it. It also includes projects that you can use to build and redeploy the OSGi bundles.
The provided project has been tested with Eclipse 3.x and later, and requires only the standard Java development project perspective. Complete the following steps to set up of your WebSphere® eXtreme Scale development environment.

1. Open Eclipse to a new or existing workspace.
2. From the File menu, select Import.
3. Expand the General folder. Select Existing Projects into Workspace, and click Next.
4. In the Select root directory field, type or browse to the *wxs_sample_osgi_root* directory. Click Finish. Several new projects are displayed in your workspace. Build errors will be fixed by defining two user libraries. Complete the next steps to define the user libraries.
5. From the Window menu, select Preferences.
6. Expand the Java > Build Path branch, and select User Libraries.
7. Define the eXtreme Scale user library.
   a. Click New.
   b. Type `eXtremeScale` in the User Library Name field, and click OK.
   c. Select the new user library, and click Add JARs.
      i. Browse and select the objectgrid.jar file from the *wxs_install_root*/lib directory. Click OK.
      ii. To include API documentation for the ObjectGrid APIs, select the API documentation location for the objectgrid.jar file that you added in the previous step. Click Edit.
      iii. In the location path box for the API documentation, select the Javadoc.zip file that is included in the following directory: *wxs_install_root*/docs/javadoc.zip.
8. Define the Google Protocol Buffers user library.
   a. Click New.
   b. Type `com.google.protobuf` in the User Library Name field, and click OK.
   c. Select the new user library, and click Add JARs.
      i. Browse and select the com.google.protobuf_2.4.0.a.jar file from the *wxs_sample_osgi_root*/lib directory. Click OK.

Lesson checkpoint

In this lesson, you imported the sample Eclipse project and defined the user libraries that fixed any build errors.

< Previous | Next >
< Previous | Next >

Lesson 3.2: Start a client and insert data into the grid

Complete this lesson to start a non-OSGi client and run a client application.

The Java™ client application is com.ibm.websphere.samples.xs.proto.client.Client. The Eclipse project, wxs.sample.osgi.protobuf.client, contains the Java client application. The main class file is com.ibm.websphere.samples.xs.proto.client.Client.
This client uses a client override, ObjectGrid descriptor XML file to override the OSGi configuration, so that the client can run in a non-OSGi environment. See the following content of the file with comments and headers removed.

```
<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="Grid" txTimeout="15">
            <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsstartclientgridOSGi_ObjectGridEven
tListener" className="" osgiService=""/>
            <backingMap name="Map" readOnly="false"
                lockStrategy="PESSIMISTIC" lockTimeout="5"
                copyMode="COPY_TO_BYTES" pluginCollectionRef="serializer"/>

        </objectGrid>
    </objectGrids>

    <backingMapPluginCollections>
        <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsstartclientgridOSGi_serializer">

                        <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsstartclientgridOSGi_MapSerializer"
                        className="com.ibm.websphere.samples.xs.serializer.proto.ProtoMapSerializer"
       osgiService="">
                                <property name="keyType" type="java.lang.String"
                                    value="com.ibm.websphere.samples.xs.serializer.proto.DataObjects2$OrderKey" />
                        <property name="valueType" type="java.lang.String"
                            value="com.ibm.websphere.samples.xs.serializer.proto.DataObjects2$Order" />
                        </bean>
                </backingMapPluginCollection>
    </backingMapPluginCollections>
</objectGridConfig>
```

Click Run As > Java Application to run the client application.
When you run the application, the following message is displayed. The message indicates that an order was inserted:

```
order
com.ibm.websphere.samples.xs.serializer.proto.DataObjects1$Order$Builder@5d165d16(5000000) inserted
```

Lesson checkpoint

In this lesson, you started the com.ibm.websphere.samples.xs.proto.client.Client application, which produced an order.

< Previous | Next >
< Previous | Next >

Module 4: Querying and upgrading the sample bundle

Complete the lessons in this module to use the **xscmd** command to query the service ranking of the sample bundle, upgrade it to a new service ranking, and verify the new service ranking.

Learning objectives

After completing the lessons in this module you will know how to complete the tasks:

- Query the current service ranking for a service.
- Query the current ranking for all services.
- Query all available rankings for a service.
- Query all available service rankings.
- Use the xscmd tool to verify whether specific service rankings are available.
- Update service rankings for sample OSGi services.

Prerequisites

Complete Module 3: Running the eXtreme Scale sample client.

Lessons in this module

- **Lesson 4.1: Query service rankings**

  Complete this lesson to query current service rankings as well as those service rankings that are available for upgrade.

- **Lesson 4.2: Determine whether specific service rankings are available**

  Complete this lesson to determine whether specific service rankings are available for the service names that you specify.
- **Lesson 4.3: Update the service rankings**

  Complete this lesson to update current service rankings that you queried.

< Previous | Next >
< Previous | Next >

Lesson 4.1: Query service rankings

Complete this lesson to query current service rankings as well as those service rankings that are available for upgrade.

- Query the current service ranking for a service. Enter the following command to query the current service ranking being used for service, myShardListener, which is used by the ObjectGrid named `Grid` and map set named `MapSet`.
  1. Switch to the following directory:

     ```
     cd wxs_home/bin
     ```

  2. Enter the following command to query the current service ranking for the service, myShardListener.

     ```
     ./xscmd.sh -c osgiCurrent -g Grid -ms MapSet -sn myShardListener
     ```

     The following output is displayed:

     ```
     OSGi Service Name: myShardListener
     ObjectGrid Name MapSet Name Server Name        Current Ranking
     --------------- ----------- -----------        ---------------
     Grid            MapSet      collocatedServer   1

     CWXSI0040I: The command osgiCurrent has completed successfully.
     ```

- Query the current ranking for all services. Enter the following command to query the current service ranking for all services that are used by the ObjectGrid named `Grid` and map set named `MapSet`.
  1. Switch to the following directory:

     ```
     cd wxs_home/bin
     ```

  2. Enter the following command to query the current service ranking for all services.

     ```
     ./xscmd.sh -c osgiCurrent -g Grid -ms MapSet
     ```

     The following output is displayed:

     ```
     OSGi Service Name    Current Ranking ObjectGrid Name MapSet Name Server Name
     -----------------    --------------- --------------- ----------- -----------
     myProtoBufSerializer 1               Grid            MapSet      collocatedServer
     myShardListener      1               Grid            MapSet      collocatedServer

     CWXSI0040I: The command osgiCurrent has completed successfully.
     ```

- Query all available rankings for a service. Enter the following command to query all of the available service rankings for the service named myShardListener.
  1. Switch to the following directory:

     ```
     cd wxs_home/bin
     ```

  2. Enter the following command to query all available rankings for a service.

     ```
     ./xscmd.sh -c osgiAll -sn myShardListener
     ```

     The following output is displayed:

     ```
     Server: collocatedServer
        OSGi Service Name Available Rankings
        ----------------- ------------------
        myShardListener   1

        Summary - All servers have the same service rankings.

     CWXSI0040I: The command osgiAll has completed successfully.
     ```

     The output is grouped by the server. In this example, only the following server exists: collocatedServer.
- Query all available service rankings. Enter the following command to query all of the available service rankings for all services.
  1. Switch to the following directory:

     ```
     cd wxs_home/bin
     ```

  2. Enter the following command to query all available service rankings.

     ```
     ./xscmd.sh -c osgiAll
     ```

     The following output is displayed:

```
Server: collocatedServer
   OSGi Service Name    Available Rankings
   -----------------    ------------------
   myProtoBufSerializer 1
   myShardListener      1

Summary - All servers have the same service rankings.
```

- Install and start Version 2 of the plug-in bundle. In the server OSGi console, install a new bundle that contains a new version of the Order class and the MapSerializerPlugin plug-in. See Lesson 2.4: Install the Google Protocol Buffers and sample plug-in bundles for details about how to install the ProtoBufSamplePlugins-2.0.0.jar bundle.
    1. After the installation, start the new bundle. The services for your new bundle are available, but they are not used by the eXtreme Scale server yet. You must run a service update request to use a service with a specific version.
- Now when you query all the available service rankings again, the service ranking 2 is added in the output.
    1. Switch to the following directory:

       ```
       cd wxs_home/bin
       ```

    2. Enter the following command to query all available service rankings.

       ```
       ./xscmd.sh -c osgiAll
       ```

       The following output is displayed:

       ```
       Server: collocatedServer
          OSGi Service Name    Available Rankings
          -----------------    ------------------
          myProtoBufSerializer 1, 2
          myShardListener      1, 2

       Summary - All servers have the same service rankings.
       ```

Lesson checkpoint

In this tutorial, you queried currently specified and all available service rankings. You also displayed the service ranking for a new bundle that you installed and started.

< Previous | Next >
< Previous | Next >

Lesson 4.2: Determine whether specific service rankings are available

Complete this lesson to determine whether specific service rankings are available for the service names that you specify.

1. Enter the following command to determine whether the service named myShardListener, with service ranking 2 and service named myProtoBufSerializer, with service ranking 2 are available. The service ranking list is passed in using -sr option.
    a. Switch to the following directory:

       ```
       cd wxs_home/bin
       ```

    b. Enter the following command to determine whether the services are available:

       ```
       ./xscmd.sh -c osgiCheck -sr "myShardListener;2,myProtoBufSerializer;2"
       ```

       The following output is displayed:

       ```
       CWXSI0040I: The command osgiCheck has completed successfully.
       ```

2. Enter the following command to determine whether the service named myShardListener, with service ranking 2 and the service named myProtoBufSerializer, with service ranking 3 are available.
    a. Switch to the following directory:

       ```
       cd wxs_home/bin
       ```

    b. Enter the following command to determine whether the services are available:

       ```
       ./xscmd.sh -c osgiCheck -sr "myShardListener;2,myProtoBufSerializer;3"
       ```

       The following output is displayed:

       ```
       Server           OSGi Service         Unavailable Rankings
       ------           ------------         --------------------
       collocatedServer myProtoBufSerializer          3
       ```

Lesson checkpoint

In this lesson, you specified the services myShardListener and myProtoBufSerializer, along with specific service rankings to determine whether those rankings were available.

< Previous | Next >
< Previous

Lesson 4.3: Update the service rankings

Complete this lesson to update current service rankings that you queried.

1. Update the service rankings of the services, myShardListener and myProtoBufSerializer, to service ranking 2. The service ranking list is passed in using -sr option.
   a. Switch to the following directory:

      ```
      cd wxs_home/bin
      ```

   b. Enter the following command to update the service rankings:

      ```
      ./xscmd.sh -c osgiUpdate -g Grid -ms MapSet -sr "myShardListener;2,myProtoBufSerializer;2"
      ```

      The following output is displayed:

      ```
      Update succeeded for the following service rankings:
      Service              Ranking
      -------              -------
      myProtoBufSerializer 2
      myShardListener      2

      CWXSI0040I: The command osgiUpdate has completed successfully.
      ```

      The following output is displayed on the OSGi console:

      ```
      SystemOut O MyShardListener@326505334(version=2.0.0) order
      com.ibm.websphere.samples.xs.serializer.proto.DataObjects2$Order$Builder@
      22342234(34) updated
      ```

      Notice that the MyShardListener service is now version 2.0.0, which has service ranking 2.
2. Run the **xscmd** command to query the current service ranking for all services that are used by the ObjectGrid named `Grid` and the map set named `MapSet`.
   a. Switch to the following directory:

      ```
      cd wxs_home/bin
      ```

   b. Enter the following command to query the service rankings for all services that are used by `Grid` and `MapSet`:

      ```
      ./xscmd.sh -c osgiCurrent -g Grid -ms MapSet
      ```

      The following output is displayed:

      ```
      OSGi Service Name    Current Ranking ObjectGrid Name MapSet Name Server Name
      -----------------    --------------- --------------- ----------- -----------
      myProtoBufSerializer 2               Grid            MapSet      collocatedServer
      myShardListener      2               Grid            MapSet      collocatedServer

      CWXSI0040I: The command osgiCurrent has completed successfully.
      ```

Lesson checkpoint

In this lesson, you updated the service rankings for services myShardListener and myProtoBufSerializer.

.
< Previous

Getting started

After you install the product, you can use the getting started sample to test the installation and use the product for the first time.

- Tutorial: Getting started with WebSphere eXtreme Scale
  After you install WebSphere® eXtreme Scale in a stand-alone environment, you can use the getting started sample application to verify your installation. The getting started sample application is an introduction to in-memory data grids. The getting started sample application is only included in full (client and server) installations of WebSphere eXtreme Scale.You can use the getting started sample application to verify the connection between your client installation and the appliance. The getting started sample application is an introduction to enterprise data grids.
- Getting started with developing applications
  To begin developing WebSphere eXtreme Scale applications, you must set up your development environment, learn about APIs that you can use, then develop and test your application.

| Next >

Tutorial: Getting started with WebSphere eXtreme Scale

After you install WebSphere® eXtreme Scale in a stand-alone environment, you can use the getting started sample application to verify your installation. The getting started sample application is an introduction to in-memory data grids. The getting started sample application is only included in full (client and server) installations of WebSphere eXtreme Scale.You can use the getting started sample application to verify the connection between your client installation and the appliance. The getting started sample application is an introduction to enterprise data grids.

Learning objectives

- Learn about the ObjectGrid descriptor XML file and deployment policy descriptor XML files that you use to configure your environment.
- Start catalog and container servers with the configuration files.
- Learn about developing a client application
- Run the client application to insert data into the data grid.
- Monitor your data grids with the web console.

Time required

60 minutes
| Next >
< Previous | Next >

Getting started tutorial lesson 1.1: Defining data grids with configuration files

The objectgrid.xml and deployment.xml files are required to start container servers.

The sample uses the objectgrid.xml and deployment.xml files that are in the *wxs_install_root*/ObjectGrid/gettingstarted/xml directory. These files are passed to the start commands to start container servers and a catalog server. The objectgrid.xml file is the ObjectGrid descriptor XML file. The deployment.xml file is the ObjectGrid deployment policy descriptor XML file. These files together define a distributed topology.

Related reference:

ObjectGrid descriptor XML file
Deployment policy descriptor XML file

ObjectGrid descriptor XML file

An ObjectGrid descriptor XML file is used to define the structure of the ObjectGrid that is used by the application. It includes a list of backing map configurations. These backing maps store the cache data. The following example is a sample objectgrid.xml file. The first few lines of the file include the required header for each ObjectGrid XML file. This example file defines the `Grid` ObjectGrid with `Map1` and `Map2` backing maps.

```
<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="Grid">
            <backingMap name="Map1" />
            <backingMap name="Map2" />
        </objectGrid>
    </objectGrids>

</objectGridConfig>
```

Deployment policy descriptor XML file

The deployment policy descriptor XML file is intended to be paired with the corresponding ObjectGrid XML, the objectgrid.xml file. In the following example, the first few lines of the deployment.xml file include the required header for each deployment policy XML file. The file defines the objectgridDeployment element for the `Grid` ObjectGrid that is defined in the objectgrid.xml file. The `Map1` and `Map2` BackingMaps that are defined within the `Grid` ObjectGrid are included in the `mapSet` mapSet.

```
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy
 ../deploymentPolicy.xsd"
xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">
<objectgridDeployment objectgridName="Grid">
       <mapSet name="mapSet" numberOfPartitions="13" minSyncReplicas="0"
                  maxSyncReplicas="1" >
            <map ref="Map1"/>
            <map ref="Map2"/>
        </mapSet>
    </objectgridDeployment>
</deploymentPolicy>
```

The numberOfPartitions attribute of the mapSet element specifies the number of partitions for the map set. This attribute is optional; the default value is `1`. The attribute value must be appropriate for the anticipated capacity of the data grid.

The minSyncReplicas attribute of the mapSet element specifies the minimum number of synchronous replicas for each partition in the map set. This attribute is optional; the default is `0`. Primary and replica shards are not placed until the catalog service domain can support the minimum number of synchronous replicas. To support the minSyncReplicas value, you need one more container server than the value of the minSyncReplicas attribute. If the number of synchronous replicas falls below the value of the minSyncReplicas attribute, write transactions are no longer allowed for that partition.

The maxSyncReplicas attribute of the mapSet element is to specify the maximum number of synchronous replicas for each partition in the map set. This attribute is optional; the default is `0`. No other synchronous replicas are placed for a partition after a catalog service 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 is not already met. The sample set the maxSyncReplicas to `1`, which means the catalog service domain places one synchronous replica at most. If you start more than one container server, only one synchronous replica is placed in one of the container server instances.

Lesson checkpoint

In this lesson, you learned:

- How to use the ObjectGrid descriptor XML file to define maps that store the cache data.
- How to use the deployment descriptor XML file to define the number of partitions and replicas for the data grid.

< Previous | Next >
< Previous | Next >

Getting started tutorial lesson 2.1: Creating a client application

To insert, delete, update, and retrieve data from your data grid, you must write a client application. The getting started sample includes a client application that you can use to learn about creating your own client application.

The Client.java file in the *wxs_install_root*/ObjectGrid/gettingstarted/client/src/ directory is the client program that demonstrates how to connect to a catalog server, obtain the ObjectGrid instance, and use the ObjectMap API. The ObjectMap API stores data as key-value pairs and is ideal for caching objects that have no relationships involved. The following steps discuss the contents of the Client.java file.

If you need to cache objects that have relationships, use the EntityManager API.

1. Connect to the catalog service by obtaining a ClientClusterContext instance.
   To connect to the catalog server, use the connect method of ObjectGridManager API. The connect method that is used requires only the catalog server endpoint in the format of *`hostname`*`:`*`port`*. You can indicate multiple catalog server endpoints by separating the list of *`hostname`*`:`*`port`* values with commas. The following code snippet demonstrates how to connect to a catalog server and obtain a ClientClusterContext instance:

   ```
   ClientClusterContext ccc = ObjectGridManagerFactory.getObjectGridManager().connect("localhost:2809", null, null);
   ```

   If the connections to the catalog servers succeed, the connect method returns a ClientClusterContext instance. The ClientClusterContext instance is required to obtain the ObjectGrid from the ObjectGridManager API.
2. Obtain an ObjectGrid instance.
   To obtain ObjectGrid instance, use the getObjectGrid method of the ObjectGridManager API. The getObjectGrid method requires both the ClientClusterContext instance and the name of the data grid instance. The ClientClusterContext instance is obtained during the connection to catalog server. The name of ObjectGrid instance is `Grid` that is specified in the objectgrid.xml file. The following code snippet demonstrates how to obtain the data grid by calling the getObjectGrid method of the ObjectGridManager API.

   ```
   ObjectGrid grid = ObjectGridManagerFactory.getObjectGridManager().getObjectGrid(ccc, "Grid");
   ```

3. Get a Session instance.
   You can get a Session from the obtained ObjectGrid instance. A Session instance is required to get the ObjectMap instance, and perform transaction demarcation. The following code snippet demonstrates how to get a Session instance by calling the getSession method of the ObjectGrid API.

   ```
   Session sess = grid.getSession();
   ```

4. Get an ObjectMap instance.
   After getting a Session, you can get an ObjectMap instance from a Session instance by calling getMap method of the Session API. You must pass the name of map as parameter to getMap method to get the ObjectMap instance. The following code snippet demonstrates how to obtain ObjectMap by calling the getMap method of the Session API.

   ```
   ObjectMap map1 = sess.getMap("Map1");
   ```

5. Use the ObjectMap methods.
   After an ObjectMap instance is obtained, you can use the ObjectMap API. Remember that the ObjectMap interface is a transactional map and requires transaction demarcation by using the begin and commit methods of the Session API. If there is no explicit transaction demarcation in the application, the ObjectMap operations run with auto-commit transactions.

   - The following code snippet demonstrates how to use the ObjectMap API with an auto-commit transaction.

     ```
     map1.insert(k, v);
     ```

   - The following code snippet demonstrates how to use the ObjectMap API with explicit transaction demarcation.

     ```
     sess.begin();
     map1.insert(key1, value1);
     ```

```
sess.commit();
```

6. Optional: Close the Session. After all of the Session and ObjectMap operations are complete, close the session with the Session.close() method. Running this method returns the resources that were being used by the session.

   ```
   sess.close();
   ```

   As a result, subsequent getSession() method calls return faster, and fewer Session objects are in the heap.

Related concepts:

Caching objects with no relationships involved (ObjectMap API)
Related tasks:

Getting started with developing applications
Tutorial: Storing order information in entities
Related information:

API documentation

Lesson checkpoint

In this lesson, you learned how to create a simple client application for performing data grid operations.
< Previous | Next >
< Previous | Next >

Getting started tutorial lesson 3.2: Running the getting started sample client application

Use the following steps to run a client to interact with the data grid. The catalog server, container server, and client all run on a single server in this example.

Open a terminal session or command-line window to run client commands.
The `runclient.sh|bat` script runs the simple create, retrieve, update, and delete (CRUD) client and starts the specified operation. The `runclient.sh|bat` script is run with the following parameters:

- UNIX Linux `./runclient.sh command value1 value2`
- Windows `runclient.bat command value1 value2`

For *command*, use one of the following options:

- Specify as `i` to insert *value2* into data grid with key *value1*
- Specify as `u` to update object that is keyed by *value1* to *value2*
- Specify as `d` to delete object that is keyed by *value1*
- Specify as `g` to retrieve and display object that is keyed by *value1*

1. Add data to the data grid.
   Important: If your system is using double byte character sets (DBCS), you might see garbled or corrupted text when you insert data into the data grid with the **runClient** script. This text can display in the output or in the cache. To work around this issue, update the Java call in the **runClient** script to include the -Xargencoding argument, and then specify the DBCS as a Unicode character set. For example, use the command: `\u runClient i key\u2e81 Hello\2e84World`
   - UNIX Linux `./runclient.sh i key1 helloWorld`
   - Windows `runclient.bat i key1 helloWorld`
2. Search and display the value:
   - UNIX Linux `./runclient.sh g key1`
   - Windows `runclient.bat g key1`
3. Update the value:
   - UNIX Linux `./runclient.sh u key1 goodbyeWorld`
   - Windows `runclient.bat u key1 goodbyeWorld`
4. Delete the value:
   - UNIX Linux `./runclient.sh d key1`
   - Windows `runclient.bat d key1`

Lesson checkpoint

Lessons learned

In this lesson, you learned:

- How to run the sample client application to insert, get, update, and delete data from the data grid.

< Previous | Next >
< Previous

Getting started tutorial lesson 4: Monitor your environment

You can use the **xscmd** utility and web console tools to monitor your data grid environment.

Related tasks:

Viewing statistics with the web console
Monitoring with the web console
Starting and logging on to the web console
Connecting the web console to catalog servers
Monitoring with the xscmd utility
Administering with the xscmd utility
Related reference:

Web console statistics
stopOgServer script

Monitoring with the web console

With the web console, you can chart current and historical statistics. This console provides some preconfigured charts for high-level overviews, and has a custom reports page that you can use to build charts from the available statistics. You can use the charting capabilities in the monitoring console of WebSphere eXtreme Scale to view the overall performance of the data grids in your environment.

Install the web console as an optional feature when you run the installation wizard.

1. Start the console server. The **startConsoleServer.bat|sh** script for starting the console server is in the *wxs_install_root*/ObjectGrid/bin directory of your installation.
2. Log on to the console.
    a. From your web browser, go to `https://your.console.host:7443`, replacing `your.console.host` with the host name of the server onto which you installed the console. If you had changed the port number, remember to update the port number in the URL.
    b. Log on to the console.
        - **User ID:** `admin`
        - **Password:** `admin`

    The console welcome page is displayed.
3. Edit the console configuration. Click Settings > Configuration to review the console configuration. The console configuration includes information such as:
    - Trace string for the WebSphere eXtreme Scale client, such as `*=all=disabled`
    - The Administrator name and password
    - The Administrator e-mail address
4. Establish and maintain connections to catalog servers that you want to monitor. Repeat the following steps to add each catalog server to the configuration.
    a. Click Settings > eXtreme Scale Catalog Servers.
    b. Add a new catalog server.
        i. Click the add icon () to register an existing catalog server.
        ii. Provide information, such as the host name and listener port. See Planning for network ports for more information about port configuration and defaults.
        iii. Click OK.
        iv. Verify that the catalog server has been added to the navigation tree.
5. Group the catalog servers that you created into a catalog service domain. You must create a catalog service domain when security is enabled in your catalog servers because security settings are configured in the catalog service domain.
    a. Click Settings > eXtreme Scale Domains page.
    b. Add a new catalog service domain.
        i. Click the add icon () to register a catalog service domain. Enter a name for the catalog service domain.
        ii. After you create the catalog service domain, you can edit the properties. The catalog service domain properties follow:

            Name
                Indicates the host name of the domain, as assigned by the administrator.
            Catalog servers
                Lists one or more catalog servers that belong to the selected domain. You can add the catalog servers that you created in the previous step.
            Generator class
                Specifies the name of the class that implements the CredentialGenerator interface. This class is used to get credentials for clients. If you specify a value in this field, the value overrides the crendentialGeneratorClass property in the client.properties file.
            Generator properties
                Specifies the properties for the CredentialGenerator implementation class. The properties are set to the object with the setProperties(String) method. The credentialGeneratorprops value is used only if the value of the credentialGeneratorClass property is not null. If you specify a value in this field, the value overrides the credentialGeneratorProps property in the client.properties file.
            eXtreme Scale client properties path
                Specifies the path to the client properties file that you edited to include SSL properties. For example, you might indicate the /ObjectGridProperties/sampleclient.properties file. If you want to stop the console from trying to use SSL connections, you can delete the value in this field. After you set the path, the console uses an unsecured connection.

        iii. Click OK.
        iv. Verify that the domain has been added to the navigation tree.

    To view information about an existing catalog service domain, click the name of the catalog service domain in the navigation tree on the Settings > eXtreme Scale Domains page.

6. View the connection status. The Current domain field indicates the name of the catalog service domain that is currently being used to display information in the web console. The connection status displays next to the name of the catalog service domain.
7. View statistics for the data grids and servers, or create a custom report.

Monitoring with the xscmd utility

1. Optional: If client authentication is enabled: Open a command-line window. On the command line, set appropriate environment variables.
2. Go to the *wxs_home*/bin directory.
   `cd wxs_home/bin`
3. Run various commands to display information about your environment.
   - Show all the online container servers for the Grid data grid and the mapSet map set:

     ```
     xscmd -c showPlacement -g Grid -ms mapSet
     ```

   - Display the routing information for the data grid.

     ```
     xscmd -c routetable -g Grid
     ```

   - Display the number of map entries in the data grid.

     ```
     xscmd -c showMapSizes -g Grid -ms mapSet
     ```

Stopping the servers

After you are done using the client application and monitoring the getting started sample environment, you can stop the servers.

- If you used the script files to start the servers, use <ctrl+c> to stop the catalog service process and container servers in the respective windows.
  Note: You can only use <ctrl+c> to stop command scripts that start with "run". For example, **runcat.bat**.
- If you used the **startOgServer** command to start your servers, use the **stopOgServer** command to stop the servers.

  Stop the container server:
  - UNIX Linux `stopOgServer.sh c0 -catalogServiceEndPoints localhost:2809`
  - Windows `stopOgServer.bat c0 -catalogServiceEndPoints localhost:2809`

  Stop the catalog server:
  - UNIX Linux `stopOgServer.sh cs1 -catalogServiceEndPoints localhost:2809`
  - Windows `stopOgServer.bat cs1 -catalogServiceEndPoints localhost:2809`

Lesson checkpoint

In this lesson, you learned:

- How to start the web console and connect it to the catalog server.
- How to monitor data grid and server statistics.
- How to stop the servers.

< Previous

Getting started with developing applications

To begin developing WebSphere® eXtreme Scale applications, you must set up your development environment, learn about APIs that you can use, then develop and test your application.

Before you begin

About this task

When you are developing WebSphere eXtreme Scale applications, you can use the embedded server APIs to create and start servers, ObjectGrid instances, and to insert data into the data grid. You can unit test your application and the associated configuration directly in the Eclipse environment. When you are ready to move your application to a broader environment, you can create configuration XML files that you import to create your deployment.

Procedure

1. Set up a development environment and access the API documentation.
   You can begin to use the APIs to develop your applications. You can also use the API documentation within the development environment.

   More information: Setting up a stand-alone development environment in Eclipse

   More information: Accessing API documentation

2. Create a simple application that starts servers, creates an ObjectGrid instance, and inserts data into the data grid.
   a. Use the ServerFactory API to start and stop servers.
   More information: Using the embedded server API to start and stop servers

b. Use the ObjectGridManager API to retrieve the ObjectGrid instance that you created.
More information: Interacting with an ObjectGrid using the ObjectGridManager interface

c. Use the ObjectMap API to insert data into the data grid.
More information: Caching objects with no relationships involved (ObjectMap API)

The ObjectMap API is the simplest way to access and write data to the data grid. If your objects have complex relationships, you can use the following APIs to read, write, and update data:

- Accessing data with indexes (Index API)
- Caching objects and their relationships (EntityManager API)
- Retrieving entities and objects (Query API)
- Accessing data with the REST data service

For more information about choosing between the different APIs, see Developing applications.

3. Unit test your application.
You can also use the **xscmd** utility to display information about the running servers, replicas, and so on. See Administering with the xscmd utility for more information.

4. When you are satisfied with your application within the development environment, create XML configuration files and update your application to use the configuration. The Getting Started sample application provides examples of these configuration files and a simple application that uses the configuration files.
More information: Tutorial: Getting started with WebSphere eXtreme Scale

5. Run your application using the XML configuration files. How you start your servers depends on the environment that you are using.
You can run your application in one of the following containers:

- Stand-alone Java virtual machine (JVM)
- Tomcat
- WebSphere Application Server
- OSGi

Related concepts:
Caching objects with no relationships involved (ObjectMap API)
Java API overview
Java API overview
Related information:
API documentation
Getting started tutorial lesson 2.1: Creating a client application
API documentation

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 overview
Before using WebSphere eXtreme Scale in a production environment, consider the following issues to optimize your deployment.
- 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.
- Interoperability with other products
You can integrate WebSphere eXtreme Scale with other products, such as WebSphere Application Server and WebSphere Application Server Community Edition.
- Planning for configuration
Before configuring the hardware or software, understand the following considerations.
- Planning for installation
Before you install the product, you must consider software and hardware requirements and Java environment settings.
- 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.
- Planning to develop WebSphere eXtreme Scale applications
Set up your development environment and learn where to find details about available programming interfaces.

Planning overview

Before using WebSphere® eXtreme Scale in a production environment, consider the following issues to optimize your deployment.

Caching topology considerations

Each type of cache topology has advantages and disadvantages. The caching topology you implement depends on the requirements of your environment and application. For more information about the different caching topologies, see Planning the topology.

Data capacity considerations

The following list includes items to consider:

- **Number of systems and processors**: How many physical machines and processors are needed in the environment?
- **Number of servers**: How many eXtreme Scale servers to host eXtreme Scale maps?
- **Number of partitions**: The amount of data stored in the maps is one factor in determining the number of partitions needed.
- **Number of replicas**: How many replicas are required for each primary in the domain?
- **Synchronous or asynchronous replication**: Is the data vital so that synchronous replication is required? Or is performance a higher priority, making asynchronous replication the correct choice?
- **Heap sizes**: How much data will be stored on each server?

For a detailed discussion of each of these considerations, see Planning environment capacity..

Installation considerations

You can install WebSphere eXtreme Scale in a stand-alone environment, or you can integrate the installation with WebSphere Application Server. To ensure that you are able to seamlessly upgrade your servers in the future, you must plan your environment accordingly. For the best performance, catalog servers should run on different machines than the container servers. If you must run your catalog servers and container servers on the same machine, then use separate installations of WebSphere eXtreme Scale for the catalog and container servers. By using two installations, you can upgrade the installation that is running the catalog server first. See Updating eXtreme Scale servers.

Related concepts:
Planning the topology
Interoperability with other products
Planning environment capacity
Related tasks:
Planning for configuration
Planning for installation
Planning to develop 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.
- 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).
- Embedded cache
  WebSphere eXtreme Scale grids can run within existing processes as embedded eXtreme Scale servers or you can manage them as external processes.
- 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 configure a database.
- 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.

- 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.

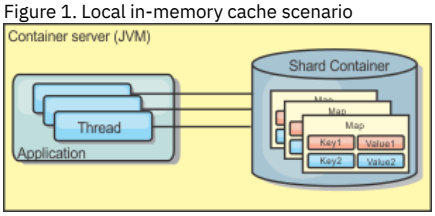
Related concepts:
Planning overview
Interoperability with other products
Planning environment capacity
Caching architecture: Maps, containers, clients, and catalogs
Related tasks:
Planning for configuration
Planning for installation
Planning to develop WebSphere eXtreme Scale applications

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 1. 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

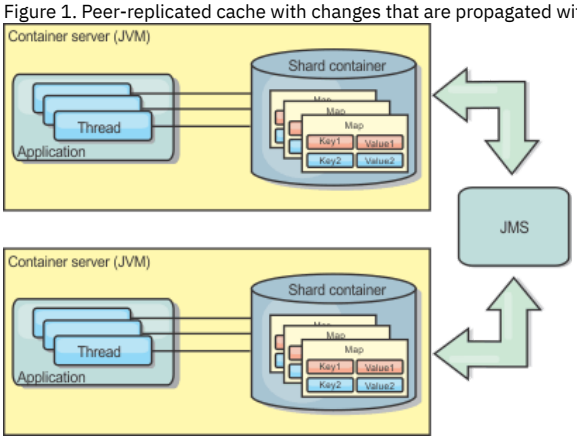
The local, in-memory cache deployment topology should only be used when the amount of data to be cached is small (can fit into a single Java virtual machine) and is relatively stable. Stale data must be tolerated with this approach. Using evictors to keep most frequently or recently used data in the cache can help keep the cache size low and increase relevance of the data.

Related concepts:
Peer-replicated local cache
Embedded cache
Distributed cache
Database integration: Write-behind, in-line, and side caching
Planning multiple data center topologies

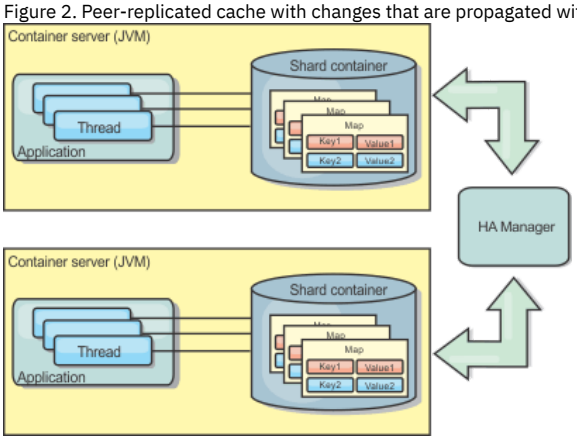
Peer-replicated local cache

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 1. 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 2. 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.

Related concepts:
Local in-memory cache

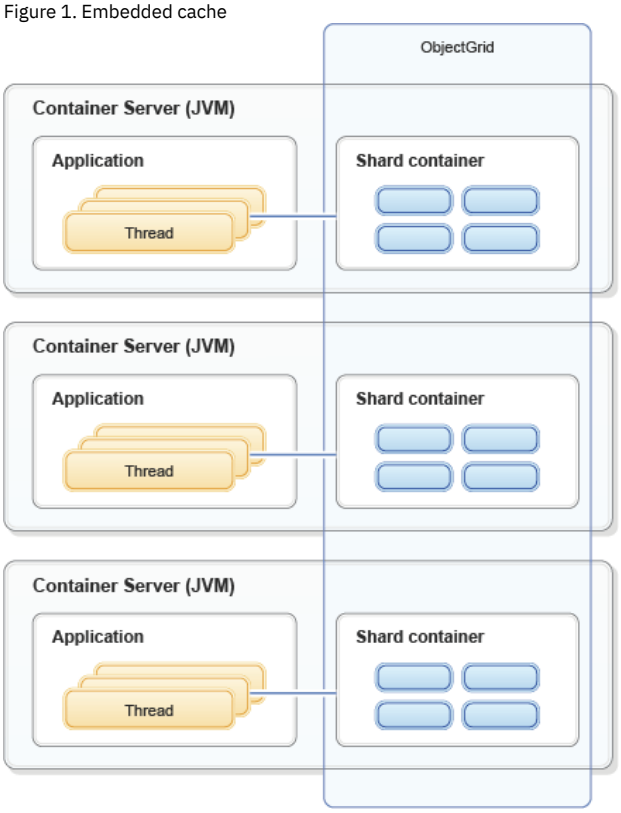
Embedded cache
Distributed cache
Database integration: Write-behind, in-line, and side caching
Planning multiple data center topologies

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 1. 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 Configuring Java Message Service (JMS)-based client synchronization.

Related concepts:
Local in-memory cache
Peer-replicated local cache
Distributed cache
Database integration: Write-behind, in-line, and side caching
Planning multiple data center topologies

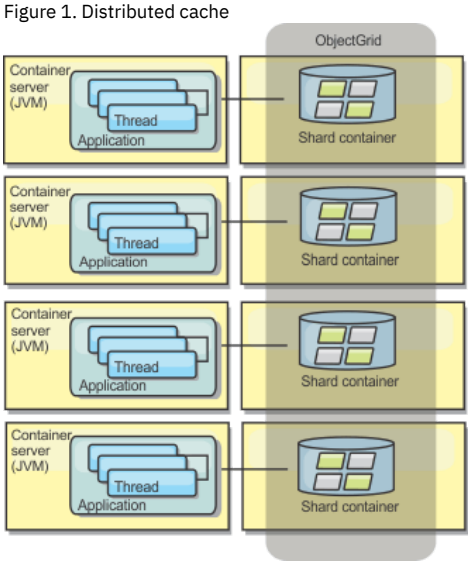
Distributed cache

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 configure a database.

Coherency of the cache

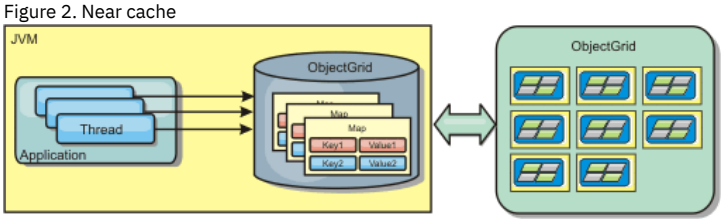
The cache is coherent because all of the clients see the same data in the cache. Each piece of data is stored on exactly one server in the cache, preventing wasteful copies of records that could potentially contain different versions of the data. A coherent cache can also hold more data as more servers 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 1. 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 2. 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.

Related concepts:
Local in-memory cache
Peer-replicated local cache
Embedded cache
Database integration: Write-behind, in-line, and side caching
Planning multiple data center topologies
Related tasks:
Configuring the near cache
Related reference:
ObjectGrid descriptor XML file

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 1. ObjectGrid as a database buffer

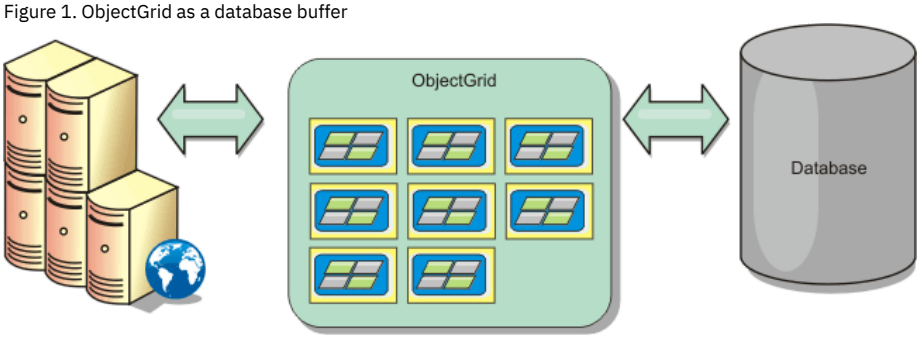
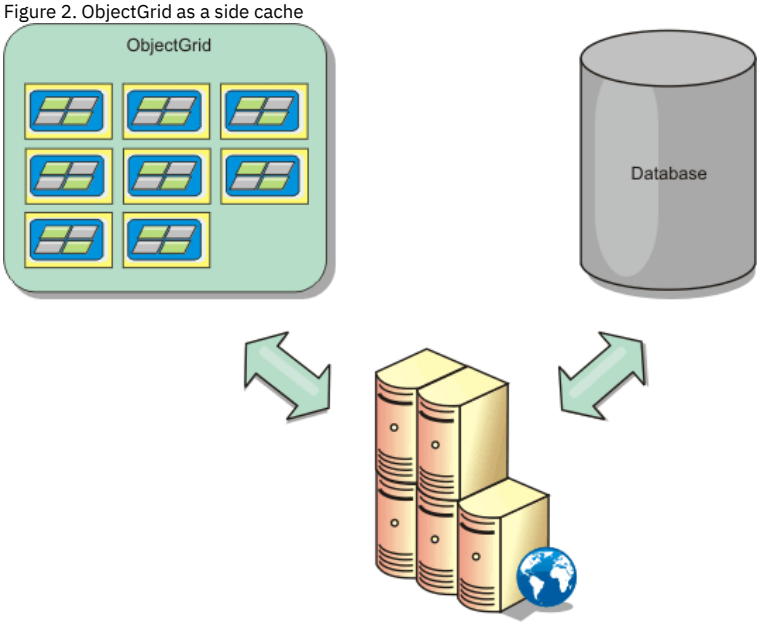


Figure 2. ObjectGrid as a side cache



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

- Write-behind caching
  You can use write-behind caching to reduce the overhead that occurs when updating a database you are using as a back end.
- Loaders
  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.
- Data preloading and warm-up
  In many scenarios that incorporate the use of a loader, you can prepare your data grid by preloading it with data.
- 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.
- Data invalidation
  To remove stale cache data, you can use invalidation mechanisms.
- Indexing
  Use the MapIndexPlugin plug-in to build an index or several indexes on a BackingMap to support non-key data access.
- JPA Loaders
  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.

Related concepts:
Local in-memory cache
Peer-replicated local cache
Embedded cache
Distributed cache
Planning multiple data center topologies
Loader considerations in a multi-master topology
Programming for JPA integration
Plug-ins for communicating with databases

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

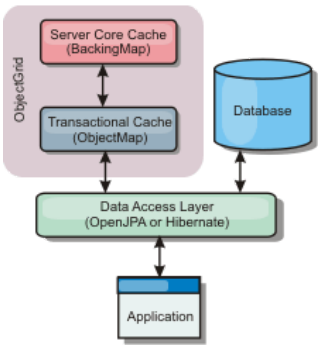
When WebSphere® eXtreme Scale is used as a side cache, the back end is used with the data grid.

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 1. 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.

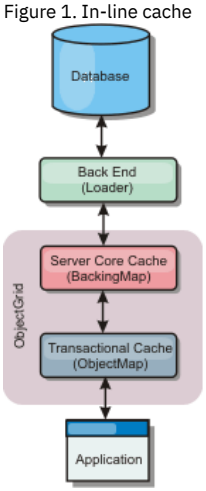
In-line cache

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 1. In-line cache



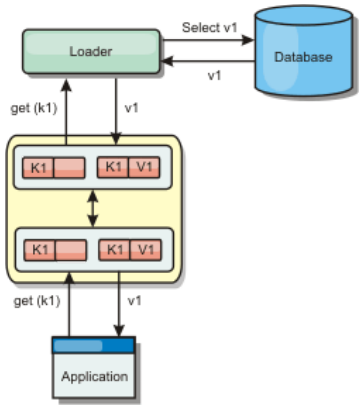
The in-line caching option simplifies data access because it allows applications to access the eXtreme Scale APIs directly. WebSphere eXtreme Scale supports several in-line caching scenarios, as follows.

- 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.

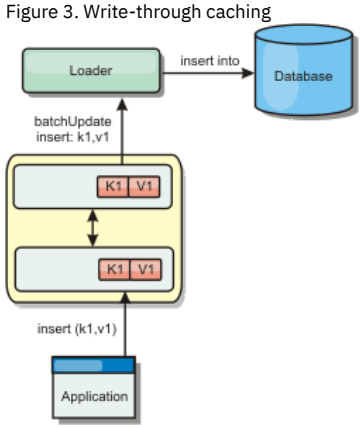
Figure 2. Read-through caching



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.

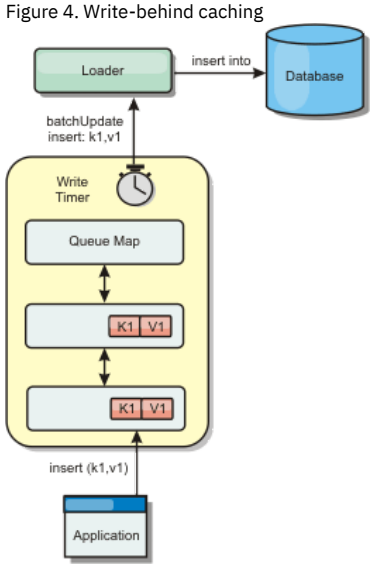
Figure 3. Write-through caching



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 4. Write-behind caching



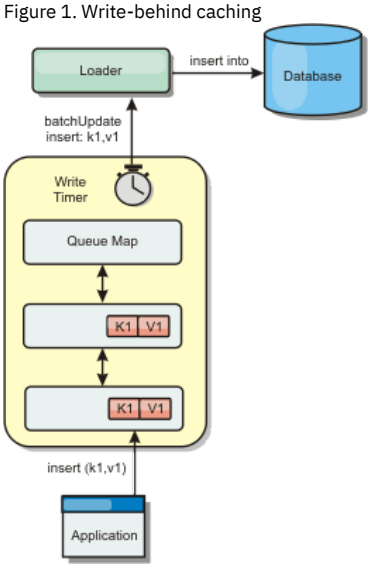
Write-behind caching

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 1. 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.

Related concepts:
Write-behind loader application design considerations
Handling failed write-behind updates
Related reference:
Example: Writing a write-behind dumper class
Example: Writing a write-behind dumper class

Loaders

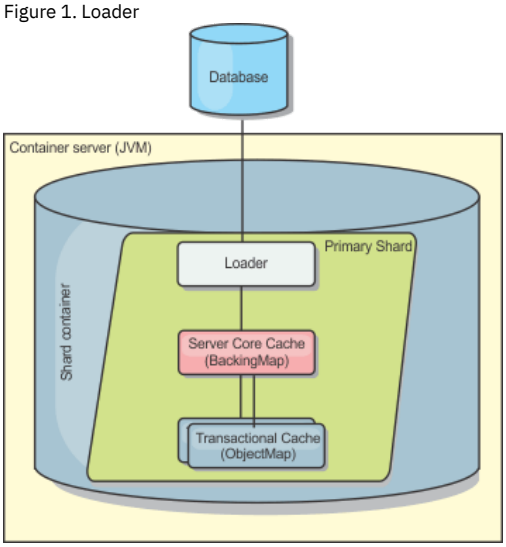
With a Loader plug-in, a data grid map can behave as a memory cache for data that is typically kept in a persistent store on either the same system or another system. Typically, a database or file system is used as the persistent store. A remote Java™ virtual machine (JVM) can also be used as the source 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. The Loader logic provides a read-through capability for the cache, which means that data is populated into the cache on demand. Since the entire data set does not need to be loaded upon startup, the cache can be populated lazily. 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 1. Loader



In order to avoid database locking on the row that requires updating, the loader can also perform optimistic transaction locking. In this scenario, no locking is required on a row. The update is overqualified to ensure that only rows that are in the same state as those originally read are changed. By storing a version attribute in the cache value, the loader can also 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 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.

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.

Related concepts:
Plug-ins for communicating with databases
Writing a loader
JPAEntityLoader plug-in
Using a loader with entity maps and tuples
Writing a loader with a replica preload controller
Related tasks:
Monitoring eXtreme Scale information in DB2
Related reference:
JPA loader programming considerations

Data preloading and warm-up

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

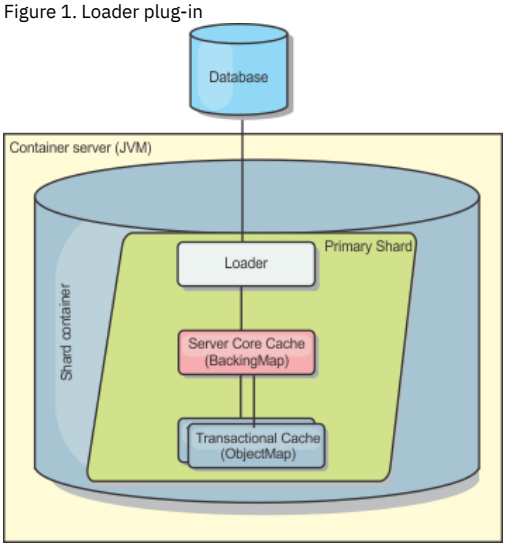
When used as a complete cache, the data grid must hold all of the data and must be loaded before any clients can connect to it. When you are using a sparse cache, you can warm up the cache with data so that clients can have immediate access to data when they connect.

Two approaches exist for preloading data into the data grid: Loader plug-in or client loader.

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 runs 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 completes.

Figure 1. Loader plug-in

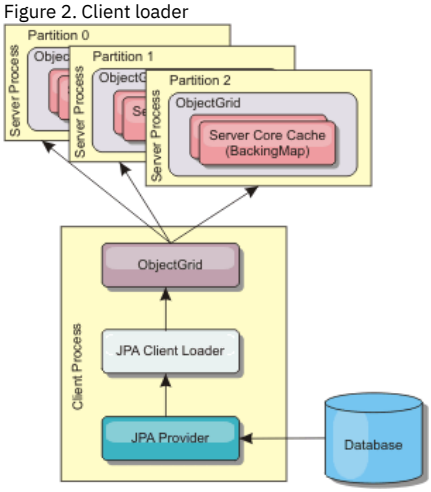


For more information about the Loader plug-in, see Plug-ins for communicating with databases.

Client loader

A client loader is a pattern for using one or more clients to load the data 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 preload mode, so that clients are not able to access the data grid while it is preloading 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.

Figure 2. Client loader



Database synchronization techniques

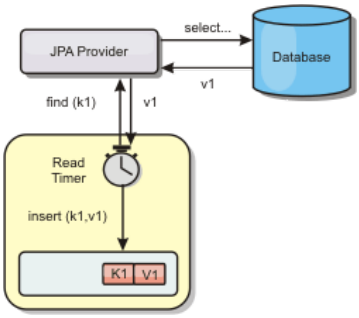
When WebSphere® eXtreme Scale is used as a cache, applications must be written to tolerate stale data if the database can be updated independently from an eXtreme Scale transaction. To serve as a synchronized in-memory database processing space, eXtreme Scale provides several ways of keeping the cache 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 1. 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. See Plug-ins for evicting cache objects for further details.

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.

8.5+

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.

Related concepts:
JMS event listener
Related tasks:
8.5+ Querying and invalidating data
Configuring the dynamic cache provider for WebSphere eXtreme Scale
Related reference:
ObjectGridEventListener plug-in
Introduction to ObjectMap
Related information:
ObjectMap.invalidate method
EntityManager.invalidate method
ObjectGridEventListener interface

Indexing

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.

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, 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.

The HashIndex class is the built-in index plug-in implementation that can support both of the built-in application index interfaces: MapIndex and MapRangeIndex. You also can create your own indexes. You can add HashIndex as either a static or dynamic index into the BackingMap, obtain either MapIndex or MapRangeIndex index proxy object, and use the index proxy object to find cached objects.

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.

Related concepts:
Plug-ins for indexing data
Plug-ins for custom indexing of cache objects
Using a composite index
Tuning query performance
Related tasks:
Configuring the HashIndex plug-in
Accessing data with indexes (Index API)
Related reference:
HashIndex plug-in attributes

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.
- Configuration considerations for multi-master topologies
  Consider the following issues when you are deciding whether and how to use multi-master replication topologies.
- 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.
- Design considerations for multi-master replication
  When implementing multi-master replication, you must consider aspects in your design such as: arbitration, linking, and performance.

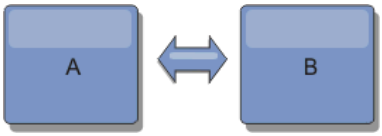
Related concepts:
Local in-memory cache
Peer-replicated local cache
Embedded cache
Distributed cache
Database integration: Write-behind, in-line, and side caching
Related tasks:
Configuring multiple data center topologies
Developing custom arbiters for multi-master replication
Troubleshooting multiple data center configurations
Administering with the xscmd utility
Related information:
dW Improve response time and data availability with WebSphere eXtreme Scale multi-master capability

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

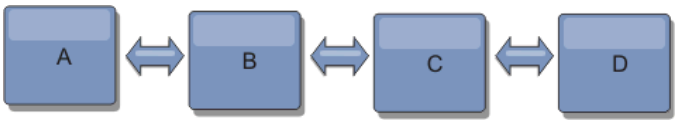
A replication data grid infrastructure is a connected graph of catalog service domains with bidirectional links among them. With a link, two catalog service domains can communicate data changes. For example, the simplest topology is a pair of catalog service domains with a single link between them. The 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.



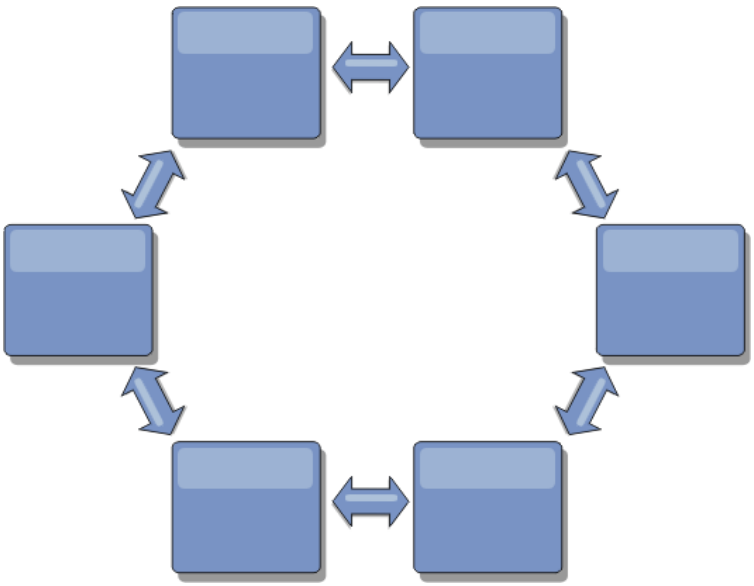
Notice that if catalog service domain C fails, the following actions would occur:

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.

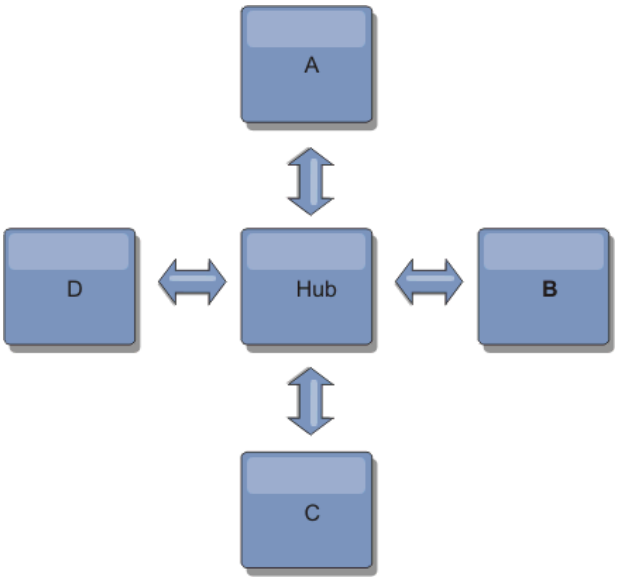


You can also deploy a more sophisticated ring topology, with a root catalog service domain at the center of the ring. The root catalog service domain functions as the central point of reconciliation. The other catalog service domains act as remote points of reconciliation for changes occurring in the 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.



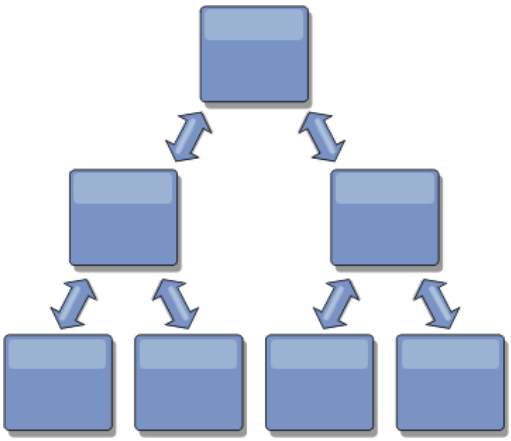
You can also use a strategy with fully replicated clients, a topology variation which uses a pair of servers that are running as a hub. Every client creates a self-contained single container data grid with a catalog in the client JVM. A client uses its data grid to connect to the hub catalog. This connection 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.

Related tasks:
Configuring multiple data center topologies
Developing custom arbiters for multi-master replication

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 data grid that is configured to use the `FIXED_PARTITIONS` placement strategy. `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.

Related concepts:
8.5+ Installing fix packs using IBM Installation Manager
Related tasks:
Configuring multiple data center topologies
Developing custom arbiters for multi-master replication
Retrieving eXtreme Scale environment information with the xscmd utility
Updating eXtreme Scale servers
Migrating to WebSphere eXtreme ScaleVersion 8.5
Starting and stopping stand-alone servers
Starting and stopping servers in a WebSphere Application Server environment

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 is 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.

Related concepts:
Programming for JPA integration
Database integration: Write-behind, in-line, and side caching
Plug-ins for communicating with databases
Related tasks:
Configuring multiple data center topologies
Developing custom arbiters for multi-master replication

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 1. 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.

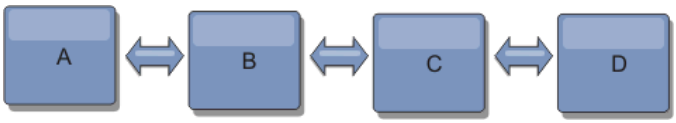
- **Change latency**
  Change latency is determined by the number of intermediate catalog service domains a change must go through before arriving at a specific catalog service domain.

  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.

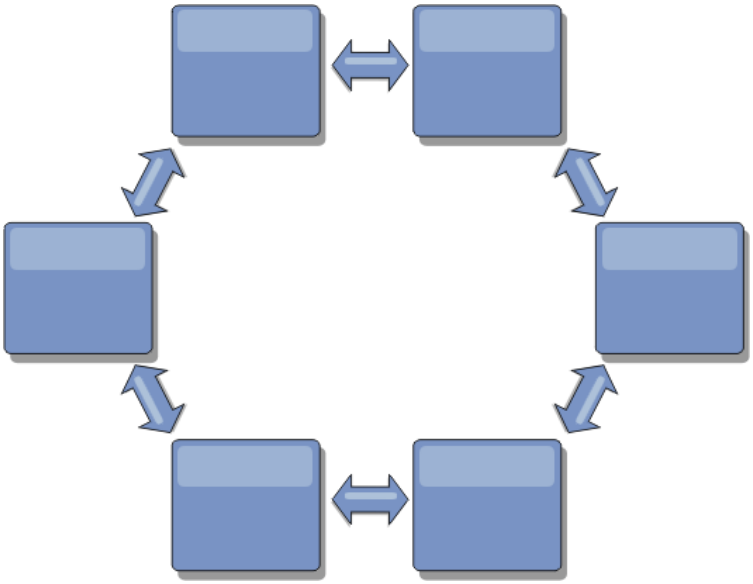
  If you have only one link between a given pair of catalog service domains, a link failure disallows propagation of changes. Similarly, changes are not propagated between catalog service domains if any of the intermediate domains experiences link failure. Your topology could have a single link from one 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:

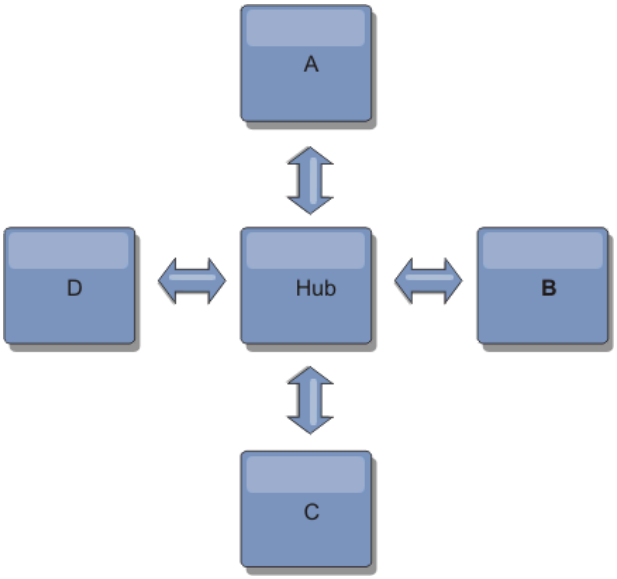
  If any of these conditions hold, Domain D does not see any changes from A:
  - Domain A is up and B is down
  - Domains A and B are up and C is down
  - The link between A and B is down
  - The link between B and C is down
  - The link between C and D is down

  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 drown 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.

Related tasks:
Configuring multiple data center topologies
Developing custom arbiters for multi-master replication

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.

Related concepts:
Planning overview
Planning the topology
Planning environment capacity
Monitoring with vendor tools
Installation topologies
Tuning garbage collection with WebSphere Real Time
Related tasks:
Planning for configuration
Planning for installation
Planning to develop WebSphere eXtreme Scale applications
Configuring the HTTP session manager for various application servers
Configuring HTTP session manager with WebSphere Portal
Configuring the HTTP session manager with WebSphere Application Server
Configuring WebSphere eXtreme Scale with WebSphere Application Server
Related information:
Configure WebSphere Commerce to use WebSphere eXtreme Scale for dynamic cache to improve performance and scale
WebSphere Business Process Management and Connectivity integration
Using WebSphere eXtreme Scale to enhance WebSphere Portal and IBM Web Content Manager performance

Planning for configuration

Before configuring the hardware or software, understand the following considerations.

- Planning for network ports
  WebSphere eXtreme Scale servers require several ports to operate.
- Planning to use IBM eXtremeMemory
  By configuring eXtremeMemory, you can store objects in native memory instead of on the Java™ heap. When you configure eXtremeMemory, you can either allow the default amount of memory to be used, or you can calculate the amount of memory that you want to dedicate to eXtremeMemory.
- Security overview
  WebSphere eXtreme Scale can secure data access, including allowing for integration with external security providers.

Related concepts:
Planning overview
Planning the topology
Interoperability with other products
Planning environment capacity
Related tasks:
Planning for installation
Planning to develop WebSphere eXtreme Scale applications

Planning for network ports

WebSphere® eXtreme Scale servers require several ports to operate.

Important: Avoid hard coding port numbers from the ephemeral range of your operating system. If you set a port that belongs in the ephemeral range, port conflicts can occur.

Catalog service domain

A catalog service domain requires the following ports to be defined:

peerPort

Specifies the port for the high availability (HA) manager to communicate between peer catalog servers over a TCP stack. In WebSphere Application Server, this setting is inherited by the high availability manager port configuration.

clientPort
: Specifies the port that peer catalog servers use to access each other's service data. While the value defined for peerPort is used for heartbeat communication between peers that are in the same domain, the clientPort is the port over which actual data gets exchanged. In WebSphere Application Server, this port is set through the catalog service domain configuration.

listenerPort (catalog server)
: Specifies the port number to which the Object Request Broker transport protocol binds for communication.
Default: `2809`

Note: When a data grid server is run inside and the ORB transport protocol is being used, another port ORB_LISTENER_ADDRESS must also be opened. The BOOTSTRAP_ADDRESS port forwards requests to this port.

JMXConnectorPort
: Defines the Secure Sockets Layer (SSL) port to which the Java™ Management Extensions (JMX) service binds. Only required if an SSL transport protocol is needed for JMX data.

SSLPort (optional)
: For secure transport of grid data, the SSL port is used only when the ORB transport protocol is used. If an SSL port is not configured an ephemeral port is chosen at startup, and this can vary each time the catalog server is restarted. When security is enabled, you must use the following argument on the **startOgServer** script to configure the Secure Socket Layer (SSL) port: `-jvmArgs -Dcom.ibm.CSI.SSLPort=<sslPort>`.

Container servers

The WebSphere eXtreme Scale container servers also require several ports to operate. By default, an eXtreme Scale container server generates its HA manager port and listener port automatically. For an environment that has a firewall, it is advantageous for you to plan and control ports. For container servers to start with specific ports, you can use the following options in the **startOgServer** command.

haManagerPort
: Specifies the port that is used by the high availability (HA) manager for heartbeat communication between peer container servers. The haManagerPort port is only used for peer-to-peer communication between container servers that are in same domain. If the haManagerPort property is not defined, then an ephemeral port is used. In WebSphere Application Server, this setting is inherited by the high availability manager port configuration.
Default: `A dynamic port is chosen`.

listenerPort (container server)
: Specifies the port number to which the ORB transport protocol binds for communication.
Default: An ephemeral port is chosen.

Note: When a data grid server is run inside WebSphere Application Server and the ORB transport protocol is being used, another port ORB_LISTENER_ADDRESS must also be opened. The BOOTSTRAP_ADDRESS port forwards requests to this port.

JMXConnectorPort
: Defines the Secure Sockets Layer (SSL) port to which the Java Management Extensions (JMX) service binds. Only required if an SSL transport protocol is needed for JMX data.

JMXServicePort
: Required only for WebSphere eXtreme Scale in a stand-alone environment. Specifies the port number on which the MBean server listens for communication with Java Management Extensions (JMX).
Default: `1099`

SSLPort (optional)
: For secure transport of grid data, the SSL port is used only when the ORB transport protocol is used. If an SSL port is not configured an ephemeral port is chosen at startup, and this can vary each time the container server is restarted. When security is enabled, you must use the following argument on the **startOgServer** script to configure the Secure Socket Layer (SSL) port: `-jvmArgs -Dcom.ibm.CSI.SSLPort=<sslPort>`.

Proper planning of port control is essential when hundreds of Java virtual machines are started in a server. If a port conflict exists, container servers do not start.

Clients

WebSphere eXtreme Scale clients can receive callbacks from servers when you are using the DataGrid API or other multi-partition operations. Use the listenerPort property in the client properties file to specify the port on which the client listens for callbacks from the server.

listenerPort (client)
: Specifies the port number to which the ORB transport protocol binds for communication. This setting configures the client to communicate with the catalog and container service. If a listener is not configured with the ORB transport protocol, an ephemeral port is chosen at startup. This port can vary each time the client application is started.
Default: An ephemeral port is chosen.

Note: When a data grid client is run inside WebSphere Application Server and the ORB transport protocol is being used, another port ORB_LISTENER_ADDRESS must also be opened. The BOOTSTRAP_ADDRESS port forwards requests to this port.

SSLPort (optional)
: For secure transport of grid data, the SSL port is used only when the ORB transport protocol is used. When the ORB transport protocol is used, SSL is an optional configuration. When SSL is enabled with the ORB transport protocol, both sides can initiate traffic. If an SSL port is not configured an ephemeral port is chosen at startup, and this can vary each time the client is restarted. When security is enabled, you must use the following system property when starting the client process: `-Dcom.ibm.CSI.SSLPort=<sslPort>`.

Ports in WebSphere Application Server

- The listenerPort value is inherited from the BOOTSTRAP_ADDRESS value for each WebSphere Application Server application server.

- The listenerPort value is inherited. The value is different depending on the type of transport you are using:
  - If you are using the ORB transport, the BOOTSTRAP_ADDRESS and the ORB_LISTENER_ADDRESS values for each WebSphere Application Server application server are used.
- The haManagerPort and peerPort values are inherited from the DCS_UNICAST_ADDRESS value for each WebSphere Application Server application server.
- The JMXServicePort and JMXConnectorPort values are inherited from the BOOTSTRAP_ADDRESS value for each WebSphere Application Server application server.
- The SSLPort value is inherited from the CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS value for each WebSphere Application Server application server.

You can define a catalog service domain in the administrative console. For more information, see Creating catalog service domains in WebSphere Application Server.

You can view the ports for a particular server by clicking one of the following paths in the administrative console:

- WebSphere Application Server Network Deployment Version 6.1: Servers > Application Servers > *server_name* > Ports > *end_point_name*.
- WebSphere Application Server Network Deployment Version 7.0 and later: Servers > Server Types > WebSphere Application Servers > *server_name* > Ports > *port_name*.

Related tasks:
Configuring ports
Configuring ports in stand-alone mode
Configuring ports in a WebSphere Application Server environment
Administering with the xscmd utility
Related reference:
xscmd utility reference
Related information:
Port number settings in WebSphere Application Server versions

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 configureeXtreme 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.
Related tasks:
Tutorial: Configuring Java SE security
Related information:
Introduction: Integrate WebSphere eXtreme Scale security with WebSphere Application Server using the WebSphere Application Server Authentication plug-ins
WebSphere Application Server: Securing applications and their environment

Planning for installation

Before you install the product, you must consider software and hardware requirements and Java environment settings.

- 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.
- Java SE considerations
  WebSphere eXtreme Scale requires Java™ SE 5, Java SE 6, or Java SE 7. In general, newer versions of Java SE have better functionality and performance.
- Java EE considerations
  As you prepare to integrate WebSphere eXtreme Scale in a Java Platform, Enterprise Edition environment, consider certain items, such as versions, configuration options, requirements and limitations, and application deployment and management.
- Directory conventions
  The following directory conventions are used throughout the documentation to 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.

Related concepts:
Planning overview
Planning the topology
Interoperability with other products
Planning environment capacity
Related tasks:
Planning for configuration
Planning to develop WebSphere eXtreme Scale applications

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

8.5+ 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.

8.5+

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 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

8.5

- WebSphere Application Server Version 6.1.0.41 or later
- WebSphere Application Server Version 7.0.0.19 or later
- WebSphere Application Server Version 8.0.0.2 or later
- WebSphere Application Server Version 8.5.0.1 or later

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

Java requirements

8.5 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 5 or later.

Java SE considerations

WebSphere® eXtreme Scale requires Java™ SE 5, Java SE 6, or Java SE 7. In general, newer versions of Java SE have better functionality and performance.

Supported versions

You can use WebSphere eXtreme Scale with Java SE 5, Java SE 6, and Java SE 7. The version that you use must be currently supported by the Java Runtime Environment (JRE) vendor. If you want to use Secure Sockets Layer (SSL), you must use an IBM® Runtime Environment.
IBM Runtime Environment, Java Technology Edition Version 5, Version 6 , and Version 7 are supported for general use with the product. Version 6 Service Release 9 Fix Pack 2 is a fully supported JRE that is installed as a part of the stand-alone WebSphere eXtreme Scale and WebSphere eXtreme Scale Client installations in the *wxs_install_root*/java directory and is available to be used by both clients and servers. If you are installing WebSphere eXtreme Scale within WebSphere Application Server, you can use the JRE that is included in the WebSphere Application Server installation. For the web console, you must use IBM Runtime Environment, Java Technology Edition Version 6 Service Release 7 and later service releases only.

WebSphere eXtreme Scale takes advantage of Java Development Kit (JDK) Version 5 , Version 6 , and Version 7 functionality as it becomes available. Generally, newer versions of the Java Development Kit (JDK) and Java SE have better performance and functionality.

For more information, see Supported software.

WebSphere eXtreme Scale features that are dependent on Java SE

Table 1. Features that require Java SE 5, Java SE 6, and Java SE 7 . WebSphere eXtreme Scale uses functionality that is introduced in Java SE 5 or Java SE 6 to provide the following product features.

Feature	Supported in Java SE 5 and later service releases	Supported in Java SE Version 6 , Version 7 and later service releases
EntityManager API annotations (Optional: You can also use XML files)	X	X
Java Persistence API (JPA): JPA loader, JPA client loader, and JPA time-based updater	X	X
Memory-based eviction (uses MemoryPoolMXBean)	X	X
Instrumentation agents: • wxssizeagent.jar: Increases the accuracy of the used bytes map metrics. • ogagent.jar: Increases the performance of field-access entities.	X	X
Web console for monitoring		X

Upgrading the JDK in WebSphere eXtreme Scale

Common questions about the upgrade process for releases of WebSphere eXtreme Scale in both stand-alone and WebSphere Application Server environments follow:

- How do I upgrade the JDK that is included in WebSphere eXtreme Scale Version 7.1.0 for a stand-alone environment?
  WebSphere eXtreme Scale Version 7.1.0 included a JRE which is automatically installed to the *wxs_install_root*/java directory. WebSphere eXtreme Scale Version 7.1.0 GA shipped with IBM Java Runtime Environment (JRE) Version 1.6 SR4. To update the JRE shipped with WebSphere eXtreme Scale 7.1.0 requires the application of WebSphere eXtreme Scale 7.1.0 Refresh Pack 0000001 followed by iFIX IFPM56253. For more information, see http://www-304.ibm.com/support/docview.wss?uid=swg1PM56253

  Note: There is currently no interim fix available to upgrade a JRE that was included with WebSphere eXtreme Scale Version 7.0.x.x.
  CAUTION:
  Errors might occur in stand-alone configurations of WebSphere eXtreme Scale that are using the Java 6 runtime and SSL Transport Layer security.
- How do I upgrade the JDK that is included with WebSphere eXtreme Scale for WebSphere Application Server?
  You need to use the JDK upgrade process that is made available by WebSphere Application Server. For more information, see http://www-304.ibm.com/support/docview.wss?uid=swg21427178.

- Which version of the JDK should I use when using WebSphere eXtreme Scale in a WebSphere Application Server environment?
  You can use any level of JDK that is supported by WebSphere Application Server, for the supported version of WebSphere Application Server.

Related concepts:
Tuning Java virtual machines
Java EE considerations
Tuning garbage collection with WebSphere Real Time
Related reference:
startOgServer script
Related information:
Tuning the IBM virtual machine for Java

Java EE considerations

As you prepare to integrate WebSphere® eXtreme Scale in a Java™ Platform, Enterprise Edition environment, consider certain items, such as versions, configuration options, requirements and limitations, and application deployment and management.

Running eXtreme Scale applications in a Java EE environment

A Java EE application can connect to a remote eXtreme Scale application. Additionally, the WebSphere Application Server environment supports starting an eXtreme Scale server when an application starts in the application server.

If you use an XML file to create an ObjectGrid instance, and the XML file is in the module of the enterprise archive (EAR) file, access the file by using the getClass().getClassLoader().getResource("META-INF/objGrid.xml") method to obtain a URL object to use to create an ObjectGrid instance. Substitute the name of the XML file that you are using in the method call.

You can use startup beans for an application to bootstrap an ObjectGrid instance when the application starts, and to destroy the instance when the application stops. A startup bean is a stateless session bean with a com.ibm.websphere.startupservice.AppStartUpHome remote location and a com.ibm.websphere.startupservice.AppStartUp remote interface. The remote interface has two methods: the start method and the stop method. Use the start method to bootstrap the instance, and use the stop method to destroy the instance. The application uses the ObjectGridManager.getObjectGrid method to maintain a reference to the instance. See Interacting with an ObjectGrid using the ObjectGridManager interface for more information.

Using class loaders

When application modules that use different class loaders share a single ObjectGrid instance in a Java EE application, verify the objects that are stored in eXtreme Scale and the plug-ins for the product are in a common loader in the application.

Managing the life cycle of ObjectGrid instances in a servlet

To manage the life cycle of an ObjectGrid instance in a servlet, you can use the init method to create the instance and the destroy method to remove the instance. If the instance is cached, it is retrieved and manipulated in the servlet code. See Interacting with an ObjectGrid using the ObjectGridManager interface for more information.

Related concepts:
Tuning Java virtual machines
Java SE considerations
Tuning garbage collection with WebSphere Real Time
Related reference:
startOgServer script
Related information:
Tuning the IBM virtual machine for Java

Directory conventions

The following directory conventions are used throughout the documentation to 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

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*

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.

- Sizing memory and partition count calculation
  You can calculate the amount of memory and partitions needed for your specific configuration.
- Sizing CPU per partition for transactions
  Although a major functionality of eXtreme Scale is its ability for elastic scaling, it is also important to consider sizing and to adjust the ideal number of CPUs to scale up.
- Sizing CPUs for parallel transactions
  Single-partition transactions have throughput scaling linearly as the data grid grows. Parallel transactions are different from single-partition transactions because they touch a set of the servers (this can be all of the servers).
- Dynamic cache capacity planning
  The Dynamic Cache API is available to Java™ EE applications that are deployed in WebSphere Application Server. You can use the dynamic cache to cache business data, generated HTML, or to synchronize the cached data in the cell by using the data replication service (DRS).

Related concepts:
Planning overview
Planning the topology
Interoperability with other products
Related tasks:
Planning for configuration
Planning for installation
Planning to develop WebSphere eXtreme Scale applications

Sizing memory and partition count calculation

You can calculate the amount of memory and partitions needed for your specific configuration.

Attention: This topic applies when you are **not** using the COPY_TO_BYTES copy mode. If you are using the COPY_TO_BYTES mode, then the memory size is much less and the calculation procedure is different. For more information about this mode, see Tuning the copy mode.
WebSphere eXtreme Scale stores data within the address space of Java™ virtual machines (JVM). Each JVM provides processor space for servicing create, retrieve, update, and delete calls for the data that is stored in the JVM. In addition, each JVM provides memory space for data entries and replicas. Java objects vary in size, therefore you must make a measurement to make an estimate of how much memory you need.
To size the memory that you need, load your application data into a single JVM. When the heap usage reaches 60%, note the number of objects that are used. This number is the maximum recommended object count for each of your Java virtual machines. To get the most accurate sizing, use realistic data and include any defined indexes in your sizing because indexes also consume memory. The best way to size memory use is to run garbage collection **verbosegc** output because this output gives you the numbers after garbage collection. You can query the heap usage at any given point through MBeans or programmatically, but those queries give you only a current snapshot of the heap. This snapshot might include uncollected garbage, so using that method is not an accurate indication of the consumed memory.

Scaling up the configuration

Number of shards per partition (numShardsPerPartition value)
To calculate the number of shards per partition, or the numShardsPerPartition value, add `1` for the primary shard plus the total number of replica shards you want. For more information about partitioning, see Partitioning.

```
numShardsPerPartition = 1 + total_number_of_replicas
```

Number of Java virtual machines (minNumJVMs value)

To scale up your configuration, first decide on the maximum number of objects that need to be stored in total. To determine the number of Java virtual machines you need, use the following formula:

```
minNumJVMS=(numShardsPerPartition * numObjs) / numObjsPerJVM
```

Round this value up to the nearest integer value.
Number of shards (numShards value)

At the final growth size, use 10 shards for each JVM. As described before, each JVM has one primary shard and (N-1) shards for the replicas, or in this case, nine replicas. Because you already have a number of Java virtual machines to store the data, you can multiply the number of Java virtual machines by 10 to determine the number of shards:

```
numShards = minNumJVMs * 10 shards/JVM
```

Number of partitions If a partition has one primary shard and one replica shard, then the partition has two shards (primary and replica). The number of partitions is the shard count divided by 2, rounded up to the nearest prime number. If the partition has a primary and two replicas, then the number of partitions is the shard count divided by 3, rounded up to the nearest prime number.

```
numPartitions = numShards / numShardsPerPartition
```

Example of scaling

In this example, the number of entries begins at 250 million. Each year, the number of entries grows about 14%. After seven years, the total number of entries is 500 million, so you must plan your capacity accordingly. For high availability, a single replica is needed. With a replica, the number of entries doubles, or 1,000,000,000 entries. As a test, 2 million entries can be stored in each JVM. Using the calculations in this scenario the following configuration is needed:

- 500 Java virtual machines to store the final number of entries.
- 5000 shards, calculated by multiplying 500 Java virtual machines by 10.
- 2500 partitions, or 2503 as the next highest prime number, calculated by taking the 5000 shards, divided by two for primary and replica shards.

Starting configuration

Based on the previous calculations, start with 250 Java virtual machines and grow toward 500 Java virtual machines over five years. With this configuration, you can manage incremental growth until you arrive at the final number of entries.
In this configuration, about 200,000 entries are stored per partition (500 million entries divided by 2503 partitions). Set the numberOfBuckets parameter on the map that holds the entries to the closest higher prime number, in this example 70887, which keeps the ratio around three.

When the maximum number of Java virtual machines is reached

When you reach your maximum number of 500 Java virtual machines, you can still grow your data grid. As the number of Java virtual machines grows beyond 500, the shard count begins to drop below 10 for each JVM, which is below the recommended number. The shards start to become larger, which can cause problems. Repeat the sizing process considering future growth again, and reset the partition count. This practice requires a full data grid restart, or an outage of your data grid.

Number of servers

Attention: Do not use paging on a server under any circumstances.
A single JVM uses more memory than the heap size. For example, 1 GB of heap for a JVM actually uses 1.4 GB of real memory. Determine the available free RAM on the server. Divide the amount of RAM by the memory per JVM to get the maximum number of Java virtual machines on the server.
Related concepts:
Sizing CPU per partition for transactions
Sizing CPUs for parallel transactions
Dynamic cache capacity planning

Sizing CPU per partition for transactions

Although a major functionality of eXtreme Scale is its ability for elastic scaling, it is also important to consider sizing and to adjust the ideal number of CPUs to scale up.

Processor costs include:

- Cost of servicing create, retrieve, update, and delete operations from clients
- Cost of replication from other Java™ virtual machines
- Cost of invalidation
- Cost of eviction policy
- Cost of garbage collection
- Cost of application logic
- Cost of serialization

Java virtual machines per server

Use two servers and start the maximum JVM count per server. Use the calculated partition counts from the previous section. Then, preload the Java virtual machines with enough data to fit on these two computers only. Use a separate server as a client. Run a realistic transaction simulation against this data grid of two servers.
To calculate the baseline, try to saturate the processor usage. If you cannot saturate the processor, then it is likely that the network is saturated. If the network is saturated, add more network cards and round robin the Java virtual machines over the multiple network cards.

Run the computers at 60% processor usage, and measure the create, retrieve, update, and delete transaction rate. This measurement provides the throughput on two servers. This number doubles with four servers, doubles again at 8 servers, and so on. This scaling assumes that the network capacity and client capacity is also able to scale.

As a result, eXtreme Scale response time should be stable as the number of servers is scaled up. The transaction throughput should scale linearly as computers are added to the data grid.
Related concepts:
Sizing memory and partition count calculation
Sizing CPUs for parallel transactions
Dynamic cache capacity planning

Sizing CPUs for parallel transactions

Single-partition transactions have throughput scaling linearly as the data grid grows. Parallel transactions are different from single-partition transactions because they touch a set of the servers (this can be all of the servers).

If a transaction touches all of the servers, then the throughput is limited to the throughput of the client that initiates the transaction or the slowest server touched. Larger data grids spread the data out more and provide more processor space, memory, network, and so on. However, the client must wait for the slowest server to respond, and the client must consume the results of the transaction.

When a transaction touches a subset of the servers, M out of N servers get a request. The throughput is then N divided by M times faster than the throughput of the slowest server. For example, if you have 20 servers and a transaction that touches 5 servers, then the throughput is 4 times the throughput of the slowest server in the data grid.

When a parallel transaction completes, the results are sent to the client thread that started the transaction. This client must then aggregate the results single threaded. This aggregation time increases as the number of servers touched for the transaction grows. However, this time depends on the application because it is possible that each server returns a smaller result as the data grid grows.

Typically, parallel transactions touch all of the servers in the data grid because partitions are uniformly distributed over the grid. In this case, throughput is limited to the first case.

Summary

With this sizing, you have three metrics, as follows.

- Number of partitions.
- Number of servers that are needed for the memory that is required.
- Number of servers that are needed for the required throughput.

If you need 10 servers for memory requirements, but you are getting only 50% of the needed throughput because of the processor saturation, then you need twice as many servers.
For the highest stability, you should run your servers at 60% processor loading and JVM heaps at 60% heap loading. Spikes can then drive the processor usage to 80–90%, but do not regularly run your servers higher than these levels.

Related concepts:
Sizing memory and partition count calculation
Sizing CPU per partition for transactions
Dynamic cache capacity planning

Dynamic cache capacity planning

The Dynamic Cache API is available to Java™ EE applications that are deployed in WebSphere® Application Server. You can use the dynamic cache to cache business data, generated HTML, or to synchronize the cached data in the cell by using the data replication service (DRS).

Overview

All dynamic cache instances created with the WebSphere eXtreme Scale dynamic cache provider are highly available by default. The level and memory cost of high availability depends on the topology used.

When using the embedded topology, the cache size is limited to the amount of free memory in a single server process, and each server process stores a full copy of the cache. As long as a single server process continues to run, the cache survives. The cache data will only be lost if all servers that access the cache are shut down.

For caching using the embedded partitioned topology, the cache size is limited to an aggregate of the free space available in all server processes. By default, the eXtreme Scale dynamic cache provider uses 1 replica for every primary shard, so each piece of cached data is stored twice.

Use the following formula A to determine the capacity of an embedded partitioned cache.

Formula A

F * C / (1 + R) = M

Where:

- F = Free memory per container process
- C = number of containers
- R = number of replicas
- M = Total size of the cache

For a WebSphere Application Server Network Deployment data grid that has 256 MB of available space in each process, with 4 server processes total, a cache instance across all of those servers could store up to 512 megabytes of data. In this mode, the cache can survive one server crashing without losing data. Also, up to two servers could be shut down sequentially without losing any data. So, for the previous example, the formula is as follows:

256mb * 4 containers/ (1 primary + 1 replica) = 512mb.

Caches using the remote topology have similar sizing characteristics as caches using embedded partitioned, but they are limited by the amount of available space in all eXtreme Scale container processes.

In remote topologies, it is possible to increase the number of replicas to provide a higher level of availability at the cost of additional memory overhead. In most dynamic cache applications this should be unnecessary, but you can edit the dynacache-remote-deployment.xml file to increase the number of replicas.

Use the following formulas, B and C, to determine the effect of adding more replicas on the high availability of the cache.

Formula B

N = Minimum(T -1, R)

Where:

- N = the number of processes that can crash simultaneously
- T = the total number of containers
- R = the total number of replicas

Formula C

Ceiling(T/ (1+N)) = m

Where:

- T = Total number containers
- N = Total number of replicas
- m = minimum number of containers needed to support the cache data.

For performance tuning with the dynamic cache provider, see Tuning the dynamic cache provider.

Cache sizing

Before an application using the WebSphere eXtreme Scale dynamic cache provider can be deployed, the general principals described in the previous section should be combined with the environmental data for the production systems. The first figure to establish is the total number of container processes and the amount of available memory in each process to hold cache data. When using the embedded topology, the cache containers will be co-located inside of the WebSphere Application server processes, so there is one container for each server that is sharing the cache. Determining the memory overhead of the application without caching enabled and the WebSphere Application Server is the best way to figure out how much space is available in the process. This can be done by analyzing verbose garbage collection data. When using the remote topology, this information can be found by looking at the verbose garbage collection output of a newly started standalone container that has not yet been populated with cache data. The last thing to keep in mind when figuring out how much space per process is available for cache data, is to reserve some heap space for garbage collection. The overhead of the container, WebSphere Application Server or stand-alone, plus the size reserved for the cache should not be more than 70% of the total heap.

After this information is collected, the values can be plugged into formula A, described previously, to determine the maximum size for the partitioned cache. Once the maximum size is known, the next step is to determine the total number of cache entries that can be supported, which requires determining the average size per cache entry. The simple way of doing this is to add 10% to the size of the customer object. See theTuning guide for dynamic cache and data replication service for more in depth information on sizing cache entries when using dynamic cache.

When compression is enabled it affects the size of the customer object, not the overhead of the caching system. Use the following formula to determine the size of a cached object when using compression:

S = O * C + O * 0.10

Where:

- S = Average size of cached object
- O = Average size of un-compressed customer object
- C = Compression ratio expressed as a fraction.

So, a 2 to 1 compression ratio is 1/2 = 0.50. Smaller is better for this value. If the object being stored is a normal POJO mostly full of primitive types, then assume a compression ratio of 0.60 to 0.70. If the object cached is a Servlet, JSP, or WebServices object, the optimal method for determining the compression ratio is to compress a representative sample with a ZIP compression utility. If this is not possible, then a compression ratio of 0.2 to 0.35 is common for this type of data. Next, use this information to determine the total number of cache entries that can be supported. Use the following D formula:

Formula D

T = S / A

Where:

- T= Total number of cache entries
- S = Total size available for cache data as computed using formula A
- A = Average size of each cache entry

Finally, you must set the cache size on the dynamic cache instance to enforce this limit. The WebSphere eXtreme Scale dynamic cache provider differs from the default dynamic cache provider in this regard. Use the following formula to determine the value to set for the cache size on the dynamic cache instance. Use the following E formula:
Formula E

Cs = Ts / Np

Where:

- Ts = Total target size for the cache
- Cs = Cache Size setting to set on the dynamic cache instance
- Np = number of partitions. The default is 47.

Set the size of the dynamic cache instance to a value calculated by formula E on each server that shares the cache instance.
Related concepts:
Sizing memory and partition count calculation
Sizing CPU per partition for transactions
Sizing CPUs for parallel transactions
Dynamic cache provider overview
Related tasks:
Configuring the dynamic cache provider for WebSphere eXtreme Scale

Planning to develop WebSphere eXtreme Scale applications

Set up your development environment and learn where to find details about available programming interfaces.

- Planning to develop Java applications
  Before you develop Java applications, you should be familiar with the available APIs, plug-ins, and any considerations that are required.

Related concepts:
Planning overview
Planning the topology
Interoperability with other products
Planning environment capacity
Related tasks:
Planning for configuration
Planning for installation

Planning to develop Java applications

Before you develop Java applications, you should be familiar with the available APIs, plug-ins, and any considerations that are required.

- Java API overview
  WebSphere® eXtreme Scale provides several features that are accessed programmatically using the Java™ programming language through application programming interfaces (APIs) and system programming interfaces.
- Java plug-ins overview
  A WebSphere eXtreme Scale plug-in is a component that provides a certain type of function to the pluggable components that include ObjectGrid and BackingMap. WebSphere eXtreme Scale provides several plug points to allow applications and cache providers to integrate with various data stores, alternative client APIs and to improve overall performance of the cache. The product ships with several default, prebuilt plug-ins, but you can also build custom plug-ins with the application.
- REST data services overview
  The WebSphere eXtreme Scale REST data service is a Java HTTP service that is compatible with Microsoft WCF Data Services (formally 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.
- Spring framework overview
  Spring is a framework for developing Java applications. WebSphere eXtreme Scale provides support to allow Spring to manage transactions and configure the clients and servers comprising your deployed in-memory data grid.

- Java class loader and classpath considerations
  Because WebSphere eXtreme Scale stores Java objects in the cache by default, you must define classes on the classpath wherever the data is accessed.
- Relationship management
  Object-oriented languages such as Java, and relational databases support relationships or associations. Relationships decrease the amount of storage through the use of object references or foreign keys.
- Cache key considerations
  WebSphere eXtreme Scale uses hash maps to store data in the grid, where a Java object is used for the key.
- Data for different time zones
  When inserting data with calendar, java.util.Date, and timestamp attributes into an ObjectGrid, you must ensure these date time attributes are created based on same time zone, especially when deployed into multiple servers in various time zones. Using the same time zone based date time objects can ensure the application is time-zone safe and data can be queried by calendar, java.util.Date and timestamp predicates.

Java API overview

WebSphere® eXtreme Scale provides several features that are accessed programmatically using the Java™ programming language through application programming interfaces (APIs) and system programming interfaces.

WebSphere eXtreme Scale APIs

When you are using eXtreme Scale APIs, you must distinguish between transactional and non-transactional operations. A transactional operation is an operation that is performed within a transaction. ObjectMap, EntityManager, Query, and DataGrid API are transactional APIs that are contained inside the Session that is a transactional container. Non-transactional operations have nothing to do with a transaction, such as configuration operations.

The ObjectGrid, BackingMap, and plug-in APIs are non-transactional. The ObjectGrid, BackingMap, and other configuration APIs are categorized as ObjectGrid Core API. Plug-ins are for customizing the cache to achieve the functions that you want, and are categorized as the System Programming API. A plug-in in eXtreme Scale is a component that provides a certain type of function to the pluggable eXtreme Scale components that include ObjectGrid and BackingMap. A feature represents a specific function or characteristic of an eXtreme Scale component, including ObjectGrid, Session, BackingMap, ObjectMap, and so on. Typically, features are configurable with configuration APIs. Plug-ins can be built-in, but might require that you develop your own plug-ins in some situations.

You can normally configure the ObjectGrid and BackingMap to meet your application requirements. When the application has special requirements, consider using specialized plug-ins. WebSphere eXtreme Scale might have built-in plug-ins that meet your requirements. For example, if you need a peer-to-peer replication model between two local ObjectGrid instances or two distributed eXtreme Scale grids, the built-in JMSObjectGridEventListener is available. If none of the built-in plug-ins can solve your business problems, refer to the System Programming API to provide your own plug-ins.

ObjectMap is a simple map-based API. If the cached objects are simple and no relationship is involved, the ObjectMap API is ideal for your application. If object relationships are involved, use the EntityManager API, which supports graph-like relationships.

Query is a powerful mechanism for finding data in the ObjectGrid. Both Session and EntityManager provide the traditional query capability.

The DataGrid API is a powerful computing capability in a distributed eXtreme Scale environment that involves many machines, replicas, and partitions. Applications can run business logic in parallel in all of the nodes in the distributed eXtreme Scale environment. The application can obtain the DataGrid API through the ObjectMap API.

The WebSphere eXtreme Scale REST data service is a Java HTTP service that is compatible with Microsoft WCF Data Services (formally ADO.NET Data Services) and implements the Open Data Protocol (OData). The REST data service allows any HTTP client to access an eXtreme Scale grid. It is compatible with the WCF Data Services support that is 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. For more details, refer to the eXtreme Scale REST data service user guide.

Related tasks:
Getting started with developing applications
Accessing API documentation
Setting up the development environment
Setting up a stand-alone development environment in Eclipse
Running a WebSphere eXtreme Scale application that uses an application server other than WebSphere Application Server in Eclipse
Running an integrated client or server application with WebSphere Application Server in Rational Application Developer
Getting started with developing applications
Accessing API documentation
Setting up the development environment
Setting up a stand-alone development environment in Eclipse
Running a WebSphere eXtreme Scale application that uses an application server other than WebSphere Application Server in Eclipse
Running an integrated client or server application with WebSphere Application Server in Rational Application Developer
Getting started with developing applications
Related information:
API documentation
API documentation

Java plug-ins overview

A WebSphere® eXtreme Scale plug-in is a component that provides a certain type of function to the pluggable components that include ObjectGrid and BackingMap. WebSphere eXtreme Scale provides several plug points to allow applications and cache providers to integrate with various data stores, alternative

client APIs and to improve overall performance of the cache. The product ships with several default, prebuilt plug-ins, but you can also build custom plug-ins with the application.

All plug-ins are concrete classes that implement one or more eXtreme Scale plug-in interfaces. These classes are then instantiated and invoked by the ObjectGrid at appropriate times. The ObjectGrid and BackingMaps each allow custom plug-ins to be registered.

ObjectGrid plug-ins

The following plug-ins are available for an ObjectGrid instance. If the plug-in is server side only, the plug-ins are removed on the client ObjectGrid and BackingMap instances. The ObjectGrid and BackingMap instances are only on the server.

- **TransactionCallback**: A TransactionCallback plug-in provides transaction life cycle events. If the TransactionCallback plug-in is the built-in JPATxCallback (com.ibm.websphere.objectgrid.jpa.JPATxCallback) class implementation, then the plug-in is server side only. However, the subclasses of the JPATxCallback class are not server side only.
- **ObjectGridEventListener**: An ObjectGridEventListener plug-in provides ObjectGrid life cycle events for the ObjectGrid, shards, and transactions.
- **ObjectGridLifecycleListener**: An ObjectGridLifecycleListener plug-in provides ObjectGrid life cycle events for the ObjectGrid instance. The ObjectGridLifecycleListener plug-in can be used as an optional mixin interface for all other ObjectGrid plug-ins.
- **ObjectGridPlugin**: An ObjectGridPlugin is an optional mix-in interface that provides extended life cycle management events for all other ObjectGrid plug-ins.
- **SubjectSource, ObjectGridAuthorization, SubjectValidation**: eXtreme Scale provides several security endpoints to allow custom authentication mechanisms to be integrated with eXtreme Scale. (Server side only)

Common ObjectGrid plug-in requirements

The ObjectGrid instantiates and initializes plug-in instances using JavaBeans conventions. All of the previous plug-in implementations have the following requirements:

- The plug-in class must be a top-level public class.
- The plug-in class must provide a public, no-argument constructor.
- The plug-in class must be available in the class path for both servers and clients (as appropriate).
- Attributes must be set using the JavaBeans style property methods.
- Plug-ins, unless specifically noted, are registered before ObjectGrid initializes and cannot be changed after the ObjectGrid is initialized.

BackingMap plug-ins

The following plug-ins are available for a BackingMap:

- **Evictor**: An evictor plug-in is a default mechanism that is provided for evicting cache entries and for creating custom evictors. The built in time-to-live evictor uses a time-based algorithm to decide when an entry in BackingMap must be evicted. Some applications might need to use a different algorithm for deciding when a cache entry needs to be evicted. The Evictor plug-in makes a custom designed Evictor available to the BackingMap to use. The Evictor plug-in is in addition to the built in time-to-live evictor. You can use the provided custom Evictor plug-in that implements well-known algorithms such as "least recently used" or "least frequently used". Applications can either plug-in one of the provided Evictor plug-ins or it can provide its own Evictor plug-in. For more information, see Plug-ins for evicting cache objects.
- **ObjectTransformer**: An ObjectTransformer plug-in allows you to serialize, deserialize, and copy objects in the cache. 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. For more information, see ObjectTransformer plug-in.
- **OptimisticCallback**: An OptimisticCallback plug-in allows you to customize versioning and comparison operations of cache objects when you are using the optimistic lock strategy. The OptimisticCallback plug-in has been replaced by the ValueDataSerializer.Versionable interface, which you can implement when you use the DataSerializer plug-in with the COPY_TO_BYTES copy mode or when you use the @Version annotation with the EntityManager API. For more information, see Plug-ins for versioning and comparing cache objects.
- **MapEventListener**: A MapEventListener plug-in provides callback notifications and significant cache state changes that occur for a BackingMap. An application might want to know about BackingMap events such as a map entry eviction or a preload of a BackingMap completion. A BackingMap calls methods on the MapEventListener plug-in to notify an application of BackingMap events. An application can receive notification of various BackingMap events by using the setMapEventListener method to provide one or more custom designed MapEventListener plug-ins to the BackingMap. The application can modify the listed MapEventListener objects by using the addMapEventListener method or the removeMapEventListener method. For more information, see MapEventListener plug-in.
- **BackingMapLifecycleListener**: A BackingMapLifecycleListener plug-in provides BackingMap life cycle events for the BackingMap instance. The BackingMapLifecycleListener plug-in can be used as an optional mix-in interface for all other BackingMap plug-ins.
- **BackingMapPlugin**: A BackingMapPlugin is an optional mix-in interface that provides extended life cycle management events for all other BackingMap plug-ins.
- **Indexing**: Use the indexing feature, which is represented by the MapIndexplug-in plug-in, to build an index or several indexes on a BackingMap map to support non-key data access.
- **Loader**: A Loader plug-in on an ObjectGrid map acts as a memory cache for data that is typically kept in a persistent store on either the same system or some other system. (Server side only) For example, a Java™ database connectivity (JDBC) Loader can be used to move data in and out of a BackingMap and one or more relational tables of a relational database. A relational database does not need to be used as the persistent store for a BackingMap. The Loader can also be used to moved data between a BackingMap and a file, between a BackingMap and a Hibernate map, between a BackingMap and a Java 2 Platform, Enterprise Edition (JEE) entity bean, between a BackingMap and another application server, and so on. The application must provide a custom-designed Loader plug-in to move data between the BackingMap and the persistent store for every technology that is used. If a Loader is not provided, the BackingMap becomes a simple in-memory cache. For more information, see Plug-ins for communicating with databases.
- **MapSerializerPlugin**: A MapSerializerPlugin allows you to serialize and inflate Java objects and non-Java data in the cache. It is used with the DataSerializer mix-in interfaces, allowing robust and flexible options for high-performance applications.

Related concepts:
Serialization using Java
Serialization overview

Serialization using the DataSerializer plug-ins
ObjectTransformer plug-in
Samples
Plug-ins for serializing cached objects
Serializer programming overview
IBM eXtremeMemory
System APIs and plug-ins
Related tasks:
Avoiding object inflation when updating and retrieving cache data
Planning to use IBM eXtremeMemory
Running eXtreme Scale containers with non-dynamic plug-ins in an OSGi environment
Related information:
Oracle Java Serialization API

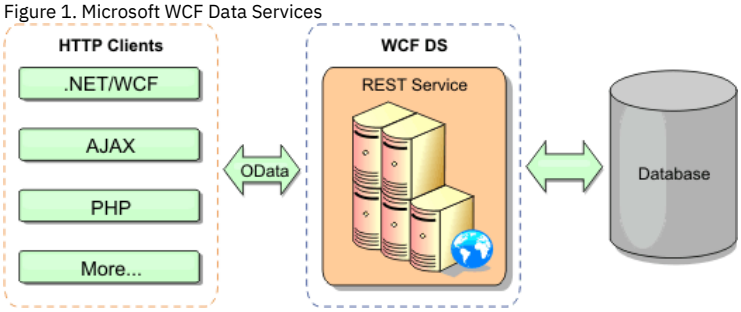
REST data services overview

The WebSphere® eXtreme Scale REST data service is a Java™ HTTP service that is compatible with Microsoft WCF Data Services (formally 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

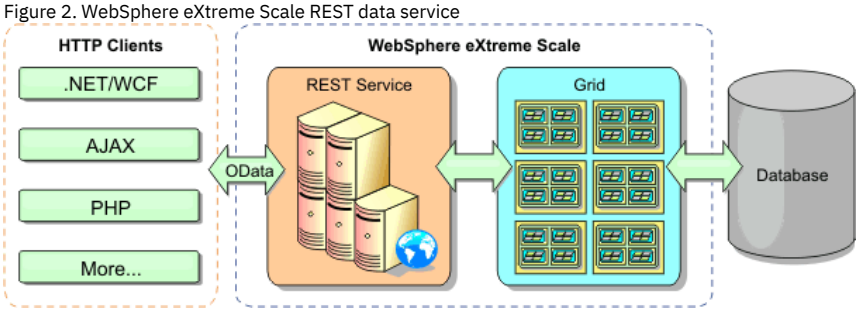
The REST data service allows any HTTP client to access a data grid. The REST data service is compatible with the WCF Data Services support supplied with the Microsoft .NET Framework 3.5 SP1. RESTful applications can be developed with the rich tooling provided by Microsoft Visual Studio 2008 SP1. The figure provides an overview of how WCF Data Services interacts with clients and databases.

Figure 1. 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 2. 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.
- 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 Java EE 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.

Related concepts:
Operations with the REST data service
Related tasks:
Accessing data with the REST data service
Configuring REST data services
Related reference:
Optimistic concurrency in the REST data service
Request protocols for the REST data service
Retrieve requests with the REST data service
Retrieving non-entities with REST data services
Insert requests with REST data services
Update requests with REST data services
Delete requests with REST data services

Spring framework overview

Spring is a framework for developing Java™ applications. WebSphere® eXtreme Scale provides support to allow Spring to manage transactions and configure the clients and servers comprising your deployed in-memory data grid.

8.5+

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. For more information, see Configuring a Spring cache provider.

Spring managed native transactions

Spring provides container-managed transactions that are similar to a Java Platform, Enterprise Edition application server. However, the Spring mechanism can use different implementations. WebSphere eXtreme Scale provides transaction manager integration which allows Spring to manage the ObjectGrid transaction life cycles. For more information, see Managing transactions with Spring.

Spring managed extension beans and namespace support

Also, eXtreme Scale integrates with Spring to allow Spring-style beans defined for extension points or plug-ins. This feature provides more sophisticated configurations and more flexibility for configuring the extension points.

In addition to Spring managed extension beans, eXtreme Scale provides a Spring namespace called "objectgrid". Beans and built-in implementations are pre-defined in this namespace, which makes it easier for users to configure eXtreme Scale. See Spring extension beans and namespace support for more details on these topics and a sample of how to start an eXtreme Scale container server using Spring configurations.

Shard scope support

With the traditional style Spring configuration, an ObjectGrid bean can either be a singleton type or prototype type. ObjectGrid also supports a new scope called the "shard" scope. If a bean is defined as shard scope, then only one bean is created per shard. All requests for beans with an ID or IDs matching that bean definition in the same shard results in that one specific bean instance being returned by the Spring container.

The following example shows that a com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl bean is defined with scope set to shard. Therefore, only one instance of the JPAPropFactoryImpl class is created per shard.

```
<bean id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxssprgfrmwk_jpaPropFactory"
class="com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl" scope="shard" />
```

Spring Web Flow

Spring Web Flow stores its session state in an HTTP session by default. If a web application uses eXtreme Scale for session management, then Spring automatically stores state with eXtreme Scale. Also, fault tolerance is enabled in the same manner as the session.

For more information, see HTTP session management.

Packaging

The eXtreme Scale Spring extensions are in the ogspring.jar file. This Java archive (JAR) file must be on the class path for Spring support to work. If a Java EE application that is running in a WebSphere Extended Deployment augmented WebSphere Application Server Network Deployment, put the spring.jar file and its associated files in the enterprise archive (EAR) modules. You must also place the ogspring.jar file in the same location.

Related tasks:
Configuring a Spring cache provider
Developing applications with the Spring framework
Starting a container server with Spring
Managing transactions with Spring
Related reference:
ObjectGrid descriptor XML file
Deployment policy descriptor XML file
Spring managed extension beans
Spring descriptor XML file
Spring objectgrid.xsd file

Java class loader and classpath considerations

Because WebSphere® eXtreme Scale stores Java™ objects in the cache by default, you must define classes on the classpath wherever the data is accessed.

Specifically, WebSphere eXtreme Scale client and container processes must include the classes or JAR files in the classpath when starting the process. When you are designing an application for use with eXtreme Scale, separate out any business logic from the persistent data objects.

See Class loading in the WebSphere Application Server information center for more information.

For considerations within a Spring Framework setting, see the packaging section in the Spring framework overview.

For settings related to using the WebSphere eXtreme Scale instrumentation agent, see Entity performance instrumentation agent.

For details on adding your classes or JAR files to the stand-alone container server classpath, see startOgServer script .

Relationship management

Object-oriented languages such as Java™, and relational databases support relationships or associations. Relationships decrease the amount of storage through the use of object references or foreign keys.

When you are using relationships in a data grid, the data must be organized in a constrained tree. One root type must exist in the tree and all children must be associated to only one root. For example: Department can have many Employees and an Employee can have many Projects. But a Project cannot have many Employees that belong to different departments. Once a root is defined, all access to that root object and its descendants are managed through the root. WebSphere® eXtreme Scale uses the hash code of the root object's key to choose a partition. For example:

```
partition = (hashCode MOD numPartitions).
```

When all of the data for a relationship is tied to a single object instance, the entire tree can be collocated in a single partition and can be accessed very efficiently using one transaction. If the data spans multiple relationships, then multiple partitions must be involved which involves additional remote calls, which can lead to performance bottlenecks.

Reference data

Some relationships include look-up or reference data such as: `CountryName`. For look-up or reference data, the data must exist in every partition. The data can be accessed by any root key and the same result is returned. Reference data such as this should only be used in cases where the data is fairly static. Updating this data can be expensive because the data needs to be updated in every partition. The DataGrid API is a common technique to keeping reference data up-to-date.

Costs and benefits of normalization

Normalizing the data using relationships can help reduce the amount of memory used by the data grid since duplication of data is decreased. However, in general, the more relational data that is added, the less it will scale out. When data is grouped together, it becomes more expensive to maintain the relationships

and to keep the sizes manageable. Since the grid partitions data based on the key of the root of the tree, the size of the tree isn't taken into account. Therefore, if you have a lot of relationships for one tree instance, the data grid may become unbalanced, causing one partition to hold more data than the others.

When the data is denormalized or flattened, the data that would normally be shared between two objects is instead duplicated and each table can be partitioned independently, providing a much more balanced data grid. Although this increases the amount of memory used, it allows the application to scale since a single row of data can be accessed that has all of the necessary data. This is ideal for read-mostly grids since maintaining the data becomes more expensive.

For more information, see Classifying XTP systems and scaling.

Managing relationships using the data access APIs

The ObjectMap API is the fastest, most flexible and granular of the data access APIs, providing a transactional, session-based approach at accessing data in the grid of maps. The ObjectMap API allows clients to use common CRUD (create, read, update and delete) operations to manage key-value pairs of objects in the distributed data grid.

When using the ObjectMap API, object relationships must be expressed by embedding the foreign key for all relationships in the parent object.

An example follows.

```
public class Department {
        Collection<String> employeeIds;
}
```

The EntityManager API simplifies relationship management by extracting the persistent data from the objects including the foreign keys. When the object is later retrieved from the data grid, the relationship graph is rebuilt, as in the following example.

```
@Entity
public class Department {
        Collection<String> employees;
}
```

The EntityManager API is very similar to other Java object persistence technologies such as JPA and Hibernate in that it synchronizes a graph of managed Java object instances with the persistent store. In this case, the persistent store is an eXtreme Scale data grid, where each entity is represented as a map and the map contains the entity data rather than the object instances.

Related concepts:
Entity manager in a distributed environment
Interacting with EntityManager
EntityManager fetch plan support
Entity query queues
Related reference:
Defining an entity schema
EntityTransaction interface

Cache key considerations

WebSphere® eXtreme Scale uses hash maps to store data in the grid, where a Java™ object is used for the key.

Guidelines

When choosing a key, consider the following requirements:

- Keys can never change. If a portion of the key needs to change, then the cache entry should be removed and reinserted.
- Keys should be small. Since keys are used in every data access operation, it's a good idea to keep the key small so that it can be serialized efficiently and use less memory.
- Implement a good hash and equals algorithm. The hashCode and equals(Object o) methods must always be overridden for each key object.
- Cache the key's hashCode. If possible, cache the hash code in the key object instance to speed up hashCode() calculations. Since the key is immutable, the hashCode should be cacheable.
- Avoid duplicating the key in the value. When using the ObjectMap API, it is convenient to store the key inside the value object. When this is done, the key data is duplicated in memory.

Data for different time zones

When inserting data with calendar, java.util.Date, and timestamp attributes into an ObjectGrid, you must ensure these date time attributes are created based on same time zone, especially when deployed into multiple servers in various time zones. Using the same time zone based date time objects can ensure the application is time-zone safe and data can be queried by calendar, java.util.Date and timestamp predicates.

Without explicitly specifying a time zone when creating date time objects, Java™ uses the local time zone and may cause inconsistent date time values in clients and servers.

Consider an example in a distributed deployment in which client1 is in time zone [GMT-0] and client2 is in [GMT-6] and both want to create a java.util.Date object with value '1999-12-31 06:00:00'. Then client1 will create java.util.Date object with value '1999-12-31 06:00:00 [GMT-0]' and client2 will create java.util.Date

object with value '1999-12-31 06:00:00 [GMT-6]'. Both java.util.Date objects are not equal because the time zone is different. A similar problem occurs when preloading data into partitions residing in servers in different time zones if local time zone is used to create date time objects.

To avoid the described problem, the application can choose a time zone such as [GMT-0] as the base time zone for creating calendar, java.util.Date, and timestamp objects.

Related concepts:
Querying data in multiple time zones
Using the ObjectQuery API
EntityManager Query API
Reference for eXtreme Scale queries
Querying data in multiple time zones

Installing

WebSphere® eXtreme Scale is an in-memory data grid that you can use to dynamically partition, replicate, and manage application data and business logic across multiple servers. After determining the purposes and requirements of your deployment, install eXtreme Scale on your system.

Before you begin

- Before you begin the installation, you should have an understanding of WebSphere eXtreme Scale caching architectures, cache and database integration, serialization, scalability and availability. See Product overview for more information.
- Plan your WebSphere eXtreme Scale deployment. For more information about the different caching topologies, sizing information, and more, see Planning.
- Verify that your environment meets the prerequisites to install eXtreme Scale. See Hardware and software requirements for more information.
- For more information on environments and other requirements, see Planning for installation.
- If you are installing an upgrade on a previous version of WebSphere eXtreme Scale, follow the steps described in Updating eXtreme Scale servers.

- 8.5+ Installation overview
  You can install WebSphere eXtreme Scale in a stand-alone or WebSphere Application Server environment. To install WebSphere eXtreme Scale , you must first install the IBM Installation Manager and obtain product files. After installing the Installation Manager and setting up access to the appropriate product repositories, you can then choose your installation and environment types. You must install a server and one or more clients. You can also install WebSphere eXtreme Scale as an in-memory data grid for use with both .NET and Java applications with extreme transaction processing (XTP) capabilities.
- 8.5+ Planning for installation
  Before you install the product, you must consider your environment.
- 8.5+ IBM Installation Manager and WebSphere eXtreme Scale product offerings
  WebSphere eXtreme Scale product offerings are available in product repositories. To access these repositories, you must first install IBM® Installation Manager.
- Installing the Eclipse Equinox OSGi framework with Eclipse Gemini for clients and servers
  If you want to deploy WebSphere eXtreme Scale in the OSGi framework, then you must set up the Eclipse Equinox Environment.
- Installing the REST data service
  This topic describes how to install the WebSphere eXtreme Scale REST data service into a Web server.
- 8.5+ Installing WebSphere eXtreme Scale or WebSphere eXtreme Scale Client with WebSphere Application Server
  You can install WebSphere eXtreme Scale or WebSphere eXtreme Scale Client in an environment in which WebSphere Application Server or WebSphere Application Server Network Deployment is installed. You can use the existing features of WebSphere Application Server or WebSphere Application Server Network Deployment to enhance your eXtreme Scale applications.
- 8.5+ Installing the Liberty profile
  You install the Liberty profile application-serving environment by using the Installation Manager, or by running a Java™ archive (JAR) file.
- 8.5+ Uninstalling the product using IBM Installation Manager
  Use IBM Installation Manager to uninstall WebSphere eXtreme Scale product offerings.
- z/OS Customizing WebSphere eXtreme Scale for z/OS
  Using the WebSphere Customization Toolbox, you can generate and run customized jobs to customize WebSphere eXtreme Scale for z/OS®.
- 8.5+ Creating and augmenting profiles for WebSphere eXtreme Scale
  After you install the product, create unique types of profiles and augment existing profiles for WebSphere eXtreme Scale.
- 8.5+ Taking the first steps after installation
  After complete and verify the installation, you can begin to use WebSphere eXtreme Scale to create your data grid.
- 8.5+ Troubleshooting the product installation
  IBM Installation Manager is a common installer for many IBM software products that you use to install this version of WebSphere eXtreme Scale.

Installation overview

You can install WebSphere® eXtreme Scale in a stand-alone or WebSphere Application Server environment. To install WebSphere eXtreme Scale , you must first install the IBM Installation Manager and obtain product files. After installing the Installation Manager and setting up access to the appropriate product repositories, you can then choose your installation and environment types. You must install a server and one or more clients. You can also install WebSphere eXtreme Scale as an in-memory data grid for use with both .NET and Java applications with extreme transaction processing (XTP) capabilities.

WebSphere eXtreme Scale environment types

- **WebSphere Application Server environment:**

By installing WebSphere eXtreme Scale on the nodes in your WebSphere Application Server environment, you can automatically start catalog servers and container servers in the same cell as your deployment manager and other application servers.

- **Stand-alone environment:**
  In a stand-alone installation, you install WebSphere eXtreme Scale in an environment that does not have WebSphere Application Server. With a stand-alone environment, you manually configure and start the catalog server and container server processes.

WebSphere eXtreme Scale installation types

If you have servers that are running client applications that access the data grid, you can use a client-only installation. Or you can choose a full (client and server) installation on nodes that run catalog servers or container servers.

- **Full (Client and Server) installation:**
  - When you are installing on WebSphere Application Server, you can choose to install the client only or both the client and server.
  - When you are installing in a stand-alone environment, you can choose to install the client-only or both the client and server.
- **Client installation:**
  You can use the client-only installation on nodes that are running the client applications.

IBM Installation Manager

Installation Manager is a single installation program that can use remote or local software flat-file repositories to install, modify, or update new WebSphere eXtreme Scale products. It determines and shows available packages—including products, fix packs, interim fixes, and so on—checks prerequisites and interdependencies, and installs the selected packages. You also use Installation Manager to easily uninstall the packages that it installed.

Overview of IBM® Installation Manager: IBM Installation Manager is a general-purpose software installation and update tool that runs on a range of computer systems. Installation Manager can be invoked through a graphical user interface (GUI) or a command-line interface. You can also create response files in XML and use them to direct the performance of Installation Manager tasks in silent mode.

For more information on using Installation Manager, read the IBM Installation Manager Version 1.5 Information Center.

Packages and package groups: Each software product that can be installed with Installation Manager is referred to as a package. An installed package has a product level and an installation location. A package group consists of all of the products that are installed at a single location.

Installation Manager modes: IBM Installation Manager can be installed in one of the following three modes:

- In admin mode, the Installation Manager is installed from an administrator or a root ID and can be invoked by any administrator or root user.
- In nonAdmin mode (also called user mode), the Installation Manager can be invoked only by the user that installed it.
- UNIX Linux In group mode, the Installation Manager can be invoked by any user ID that is connected to the default group of the user that installed it.
  This does not mean that two people can use the single instance of IBM Installation Manager at the same time.

How many Installation Managers do you need: You only need to run Installation Manager on those systems on which you install or update product code. You normally need only one Installation Manager on a system because one Installation Manager can keep track of any number of product installations.

Installing Installation Manager: When the installation kit is available on your system, you can install Installation Manager. Installation Manager consists of a set of binaries that are copied from the installation kit and a set of runtime data that describe the products that have been installed by this particular Installation Manager. Before installing Installation Manager, you must decide in which mode the Installation Manager will run as well as where the binaries and runtime data —called agent data or appdata—will reside. Then, you issue an Installation Manager installation command from the appropriate user ID to install Installation Manager.

Accessing product repositories: All software materials that will be installed with IBM Installation Manager are stored in flat-file repositories. Each repository contains program objects and metadata for one or more packages—that is, software products at a particular level. Repositories can also contain product maintenance, such as fix packs and interim fixes. Whenever you install a new product, you can choose from any of the available product levels in any accessible repository.

Installing the product: After you have installed Installation Manager and have access to all necessary product repositories, you can use the Installation Manager GUI, command-line commands, or response files to perform the actual product installations. When you install a product, you provide the package name, optionally the product level to be installed, the product location, and any other optional properties. For example, some products have optional features that you can select at installation time or a list of optional supported language packs from which you can select.

Working with installed products: You can use Installation Manager commands to list installed products and product levels. You can also obtain this information for installed copies of WebSphere eXtreme Scale products by issuing the **versionInfo** command from the product file system. You can use Installation Manager commands or response files to install a new product level, roll back to a previous level, or modify the product by adding or removing optional features or language packs.

Using the IBM Packaging Utility: With the Packaging Utility, you can create and manage packages for installation repositories. You can copy multiple packages into one repository or copy multiple disks for one product into a repository. You can copy packages from Passport Advantage® into a repository for example. For more information on the Packaging Utility, go to the IBM Installation Manager Version 1.5 Information Center.

Restrictions:

- Windows If a non-administrator installs WebSphere eXtreme Scale Version 8.5 on a Windows Vista, Windows 7, or Windows Server 2008 operating system into the Program Files or Program Files (x86) directory with User Account Control (UAC) enabled, WebSphere eXtreme Scale will not function correctly.
  UAC is an access-control mechanism that allows non-administrative users to install a software product into the Program Files or Program Files (x86) directory; but it then prohibits any write access to that directory after the installation has completed. WebSphere eXtreme Scale requires write access in the *app_server_root* directory in order to function correctly.

To resolve this issue, perform one of the following actions:
    - Install WebSphere eXtreme Scale into a directory other than Program Files or Program Files (x86).
      For example:

      ```
      C:\IBM\WebSphere\AppServer
      ```

    - Disable UAC.
- When you install a product using Installation Manager with local repositories, the installation takes a significantly longer amount of time if you use a compressed repository file directly without extracting it.
  Before you install a product using local repositories, extract the compressed repository file to a location on your local system before using Installation Manager to access it.

- Installation Manager console mode, which is included in Installation Manager Version 1.4.3 and later, does not work with WebSphere eXtreme Scale Version 8.5 offerings on systems other than z/OS®.

Important: Do not transfer the content of a repository in non-binary mode and do not convert any content on extraction.
Tip: Although almost all of the instructions in this section of the information center will work with earlier versions of IBM Installation Manager, the information here is optimized for users who have installed or upgraded to Installation Manager Version 1.5 or later.
8.5+

Available product offerings in Installation Manager

After you have installed Installation Manager and have access to all necessary product repositories, the Installation Manager lists the product offerings that are available to you. When you point to our product repository, you should see the following product offerings:

- WebSphere eXtreme Scale in a stand-alone environment
- WebSphere eXtreme Scale Client in a stand-alone environment
- WebSphere eXtreme Scale for WebSphere Application Server Version 6.1
- WebSphere eXtreme Scale for WebSphere Application Server Version 7.0
- WebSphere eXtreme Scale for WebSphere Application Server Version 8.0
- WebSphere eXtreme Scale Client forWebSphere Application Server Version 6.1
- WebSphere eXtreme Scale Client for WebSphere Application Server Version 7.0
- WebSphere eXtreme Scale Client for WebSphere Application Server Version 8.0

Important: If you want to install WebSphere Application Server and WebSphere eXtreme Scale at the same time, then you need to point to both product repositories. These are installed as two separate products, but can be installed at the same time in Installation Manager.
Related tasks:
Installing IBM Installation Manager using the GUI
Installing the product with the GUI
Uninstalling the product using the GUI
Installing IBM Installation Manager using the command line
Installing the product using the command line
Uninstalling the product using the command line
Installing IBM Installation Manager using response files
Installing the product using a response file
Uninstalling the product using response files
Installing fix packs using the GUI
Uninstalling fix packs using the GUI
Installing fix packs using the command line
Uninstalling fix packs using the command line
Installing fix packs using a response file
Uninstalling fix packs using response files

Planning for installation

Before you install the product, you must consider your environment.

- 8.5+ Installation topologies
  With WebSphere® eXtreme Scale, you can create many installation topologies that include stand-alone servers, WebSphere Application Server, or both.
- 8.5+ 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.
- 8.5+ Offering IDs for WebSphere eXtreme Scale product offerings
  When installing product updates or rolling back fixes with Installation Manager and from the command line, you are required to specify the offering ID. Use the following tables to identify the product offerings and required and optionally installable features.
- 8.5+ Java SE considerations
  WebSphere eXtreme Scale requires Java™ SE 5, Java SE 6, or Java SE 7. In general, newer versions of Java SE have better functionality and performance.
- 8.5+ Java EE considerations
  As you prepare to integrate WebSphere eXtreme Scale in a Java Platform, Enterprise Edition environment, consider certain items, such as versions, configuration options, requirements and limitations, and application deployment and management.
- 8.5+ Directory conventions
  The following directory conventions are used throughout the documentation to 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.

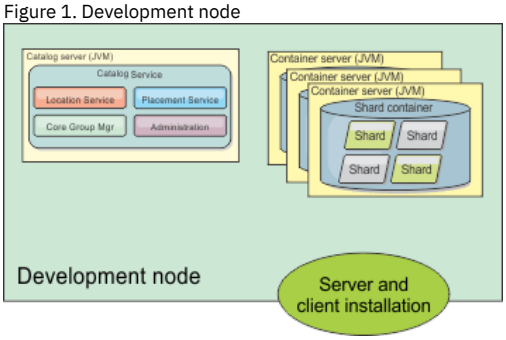
- 8.5+ Runtime files for WebSphere eXtreme Scale integrated with WebSphere Application Server
  Java archive (JAR) files are included in the installation. You can see the JAR files that are included and the location to which they are installed.
- 8.5+ Runtime files for WebSphere eXtreme Scale stand-alone installation
  Java archive (JAR) files are included in the installation. You can see the JAR files that are included and the location to which they are installed.

Installation topologies

With WebSphere® eXtreme Scale, you can create many installation topologies that include stand-alone servers, WebSphere Application Server, or both.

Development node

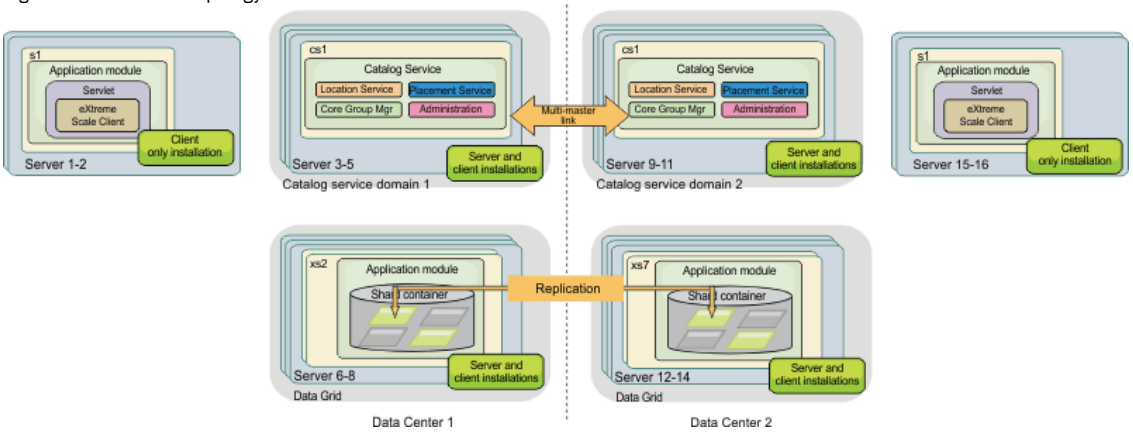
The simplest installation scenario is creating a development node. In this scenario, you install the client and server installation of WebSphere eXtreme Scale one time on the node where you want to develop your application.
Figure 1. Development node



After you complete the installation on your development node, you can configure your development environment and begin writing your applications.

Stand-alone topology

A stand-alone topology consists of servers that are not running on WebSphere Application Server. You can create many different stand-alone topologies, but the following topology is included as an example. In this topology, two data centers are present. In each data center, WebSphere eXtreme Scale full installations (client and server) and client-only installations are installed on the physical servers. The client-only installations are on the nodes that are running the web applications that are using the data grid. These nodes do not run any catalog or container servers, so the server installation is not required. A multi-master link connects the two catalog service domains in the configuration. The multi-master link enables replication between the shards in the container servers in the different data centers.
Figure 2. Stand-alone topology with two data centers

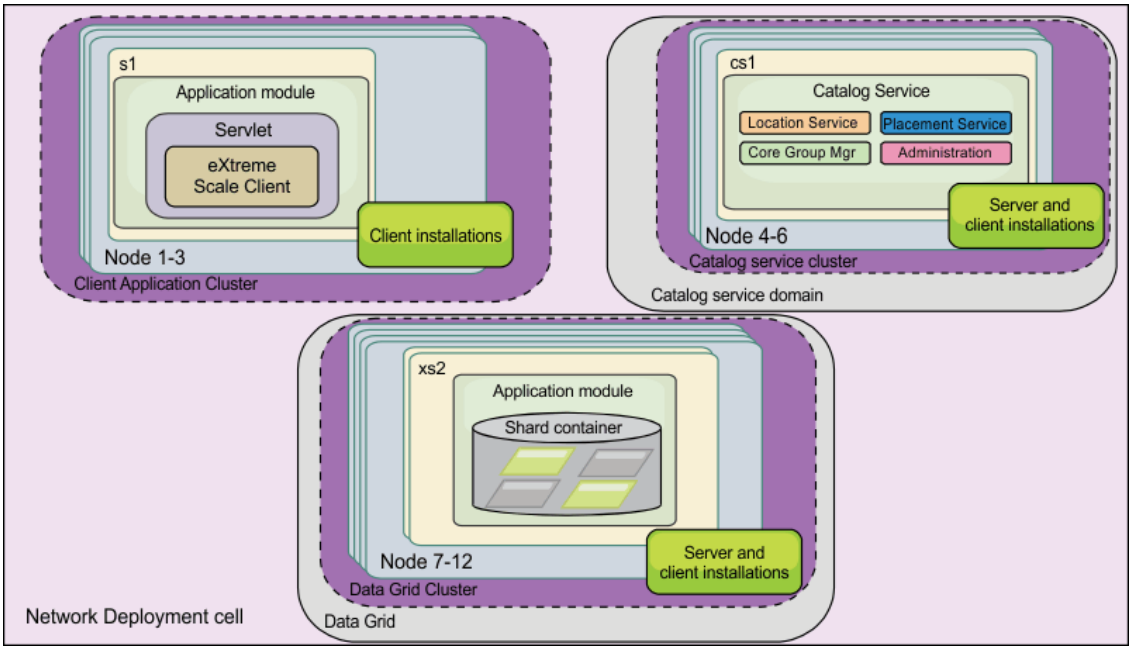


Advantages to using a stand-alone topology:

- Flexible integration options that can be embedded with vendor frameworks and libraries.
- Smaller footprint than a WebSphere Application Server topology.
- Fewer licensing requirements than a WebSphere Application Server topology.
- Expanded Java Runtime Environment (JRE) options.

WebSphere Application Server topology

You can also create an installation that runs entirely in a WebSphere Application Server cell. The clients, catalog servers, and container servers each have an associated cluster. The nodes that run the application have the client-only installation. The other nodes have the client and server installation.
Figure 3. WebSphere Application Server topology example



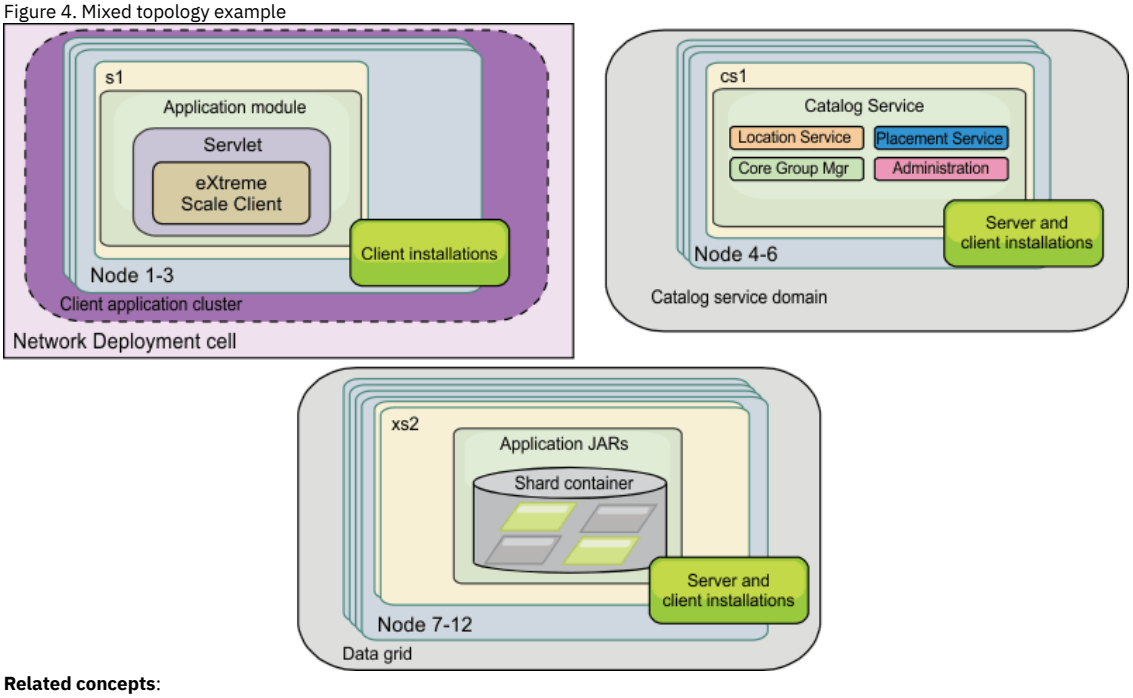
Advantages of using a WebSphere Application Server topology.

- Centralized and consistent administration and configuration.
- Security integration.
- Java EE application integration.
- Performance monitoring infrastructure (PMI) integration.
- Integration with the following WebSphere Application Server components: OpenJPA L2 cache, dynamic cache, and HTTP session persistence.

Mixed topology

You can create a mixed topology that contains both WebSphere Application Server and stand-alone servers. In the following example, the client applications are running in the WebSphere Application Server cell, while the catalog servers and container servers are running in stand-alone mode.

Figure 4. Mixed topology example



Related concepts:
Interoperability with other products
Monitoring with vendor tools
Tuning garbage collection with WebSphere Real Time

Related tasks:
Configuring the HTTP session manager for various application servers
Configuring HTTP session manager with WebSphere Portal
Configuring the HTTP session manager with WebSphere Application Server
Configuring WebSphere eXtreme Scale with WebSphere Application Server
Configuring catalog servers and catalog service domains
Configuring the quorum mechanism

Tuning the heartbeat interval setting for failover detection
Configuring the catalog service in WebSphere Application Server
Creating catalog service domains in WebSphere Application Server
Configuring catalog and container servers
Starting and stopping stand-alone servers
Using the embedded server API to start and stop servers
Configuring WebSphere Application Server applications to automatically start container servers
Configuring container servers in WebSphere Application Server
Controlling placement

Related reference:
Server properties file
startOgServer script
Catalog service domain administrative tasks
ObjectGrid descriptor XML file
Deployment policy descriptor XML file

Related information:
Configure WebSphere Commerce to use WebSphere eXtreme Scale for dynamic cache to improve performance and scale
WebSphere Business Process Management and Connectivity integration

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

8.5+ 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.

8.5+

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 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

8.5

- WebSphere Application Server Version 6.1.0.41 or later
- WebSphere Application Server Version 7.0.0.19 or later
- WebSphere Application Server Version 8.0.0.2 or later
- WebSphere Application Server Version 8.5.0.1 or later

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

Java requirements

8.5 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 5 or later.

Offering IDs for WebSphere eXtreme Scale product offerings

When installing product updates or rolling back fixes with Installation Manager and from the command line, you are required to specify the offering ID. Use the following tables to identify the product offerings and required and optionally installable features.

Table 1. Offering IDs for WebSphere eXtreme Scale product offerings

Product name	Offering ID
WebSphere® eXtreme Scale in a stand-alone environment	`com.ibm.websphere.WXS.v85`
WebSphere eXtreme Scale Client in a stand-alone environment	`com.ibm.websphere.WXSCLIENT.v85`
WebSphere eXtreme Scale for WebSphere Application Server Version 6	`com.ibm.websphere.WXS.was6.v85` Note:
WebSphere eXtreme Scale for WebSphere Application Server Version 7	`com.ibm.websphere.WXS.was7.v85`
WebSphere eXtreme Scale for WebSphere Application Server Version 8	`com.ibm.websphere.WXS.was8.v85`
WebSphere eXtreme Scale Client for WebSphere Application Server Version 6	`com.ibm.websphere.WXSCLIENT.was6.v85` Note:
WebSphere eXtreme Scale Client for WebSphere Application Server Version 7	`com.ibm.websphere.WXSCLIENT.was7.v85`
WebSphere eXtreme Scale Client for WebSphere Application Server Version 8	`com.ibm.websphere.WXSCLIENT.was8.v85`

Table 2. Installable features for WebSphere eXtreme Scale; offering ID `com.ibm.websphere.WXS.v85`

Feature name	Feature ID
WebSphere eXtreme Scale	`xs.core.feature`
WebSphere eXtreme Scale Java Runtime Environment	`xs.jre.feature`
WebSphere eXtreme Scale server in a stand-alone environment (Optional)	`xs.server.standalone.feature`
WebSphere eXtreme Scale Client in a stand-alone environment	`xs.client.standalone.feature`
WebSphere eXtreme Scale monitoring console (Optional)	`xs.console.feature`
WebSphere eXtreme Scale Samples (Optional)	`xs.samples.feature`

Table 3. Installable features for WebSphere eXtreme Scale Client; offering ID `com.ibm.websphere.WXSCLIENT.v85`

Feature name	Feature ID
WebSphere eXtreme Scale Client	`xs.client.core.feature`
WebSphere eXtreme Scale Java Runtime Environment	`xs.jre.feature`
WebSphere eXtreme Scale Client in a stand-alone environment	`xs.client.standalone.feature`
WebSphere eXtreme Scale monitoring console (Optional)	`xs.console.feature`
WebSphere eXtreme Scale Samples (Optional)	`xs.samples.feature`

Table 4. Installable features for WebSphere eXtreme Scale for WebSphere Application Server Version 7; offering ID `com.ibm.websphere.WXS.was7.v85`

Feature name	Feature ID
WebSphere eXtreme Scale	`xs.core.feature`
WebSphere eXtreme Scale server	`xs.server.feature`
WebSphere eXtreme Scale Client	`xs.client.feature`
WebSphere eXtreme Scale monitoring console (Optional)	`xs.console.feature`
WebSphere eXtreme Scale Samples (Optional)	`xs.samples.feature`

Table 5. Installable features for WebSphere eXtreme Scale Client for WebSphere Application Server Version 7; offering ID `com.ibm.websphere.WXSCLIENT.was7.v85`

Feature name	Feature ID
WebSphere eXtreme Scale Client	`xs.client.core.feature`
WebSphere eXtreme Scale Client in a stand-alone environment	`xs.client.feature`
WebSphere eXtreme Scale monitoring console (Optional)	`xs.console.feature`
WebSphere eXtreme Scale Samples (Optional)	`xs.samples.feature`

Table 6. Installable features for WebSphere eXtreme Scale for WebSphere Application Server Version 8; offering ID `com.ibm.websphere.WXS.was8.v85`

Feature name	Feature ID

Feature name	Feature ID
WebSphere eXtreme Scale for WebSphere Application Server Version 8.5.5	`xs.core.feature` This feature is required on WebSphere Application Server V8, WebSphere Application Server V8.5 (with or without the WebSphere Application Server Liberty profile), WebSphere Application Server V8.5.5 and the Liberty profile for WebSphere Application Server V8.5.5
WebSphere eXtreme Scale server for WebSphere Application Server (Optional)	`xs.server.feature` You can install this feature with WebSphere Application Server versions 8, 8.5 and 8.5.5. You cannot install this feature with the Liberty profile for WebSphere Application Server V8.5.5 and higher.
WebSphere eXtreme Scale Client for WebSphere Application Server	`xs.client.feature` This feature is required for WebSphere Application Server versions 8, 8.5 and 8.5.5. You cannot install this feature with the Liberty profile.
WebSphere eXtreme Scale monitoring console (Optional)	`xs.console.feature` This feature can be installed on WebSphere Application Server versions 8, 8.5 and 8.5.5. You cannot install this feature with the Liberty profile.
WebSphere eXtreme Scale Samples (Optional)	`xs.samples.feature` This feature can be installed on WebSphere Application Server versions 8, 8.5 and 8.5.5. You cannot install this feature with the Liberty profile.
WebSphere eXtreme Scale Client for WebSphere Application Server Version 8.5 (Optional)	`xs.liberty.feature` You can install this feature with WebSphere Application Server V8.5 with the Liberty profile installed.
WebSphere eXtreme Scale for WebSphere Application Server (Optional; Required for the Liberty profile for WebSphere Application Server Version 8.5.5)	`xs.liberty.standalone.feature` You cannot install this feature with WebSphere Application Server. This feature is the Liberty profile for WebSphere Application Server V8.5.5, and it is intended to be installed by itself.

Table 7. Installable features for WebSphere eXtreme Scale Client for WebSphere Application Server Version 8; offering ID `com.ibm.websphere.WXSCLIENT.was8.v85`

Feature name	Feature ID
WebSphere eXtreme Scale for WebSphere Application Server Version 8.5.5	`xs.client.core.feature` This feature is required on WebSphere Application Server V8, WebSphere Application Server V8.5 (with or without the Liberty profile), WebSphere Application Server V8.5.5 and the Liberty profile for WebSphere Application Server V8.5.5
WebSphere eXtreme Scale Client for WebSphere Application Server	`xs.client.feature` You can install this feature with WebSphere Application Server versions 8, 8.5 and 8.5.5. You cannot install this feature with the Liberty profile.
WebSphere eXtreme Scale monitoring console (Optional)	`xs.console.feature` This feature can be installed on WebSphere Application Server versions 8, 8.5 and 8.5.5. You cannot install this feature with the Liberty profile.
WebSphere eXtreme Scale Samples (Optional)	`xs.samples.feature` This feature can be installed on WebSphere Application Server versions 8, 8.5 and 8.5.5. You cannot install this feature with the Liberty profile.
WebSphere eXtreme Scale for WebSphere Application Server Version 8.5 (Optional)	`xs.liberty.feature` You can install this feature with WebSphere Application Server V8.5 with the Liberty profile installed.
WebSphere eXtreme Scale for WebSphere Application Server (Optional; Required for the Liberty profile for WebSphere Application Server Version 8.5.5)	`xs.liberty.standalone.feature` You cannot install this feature with WebSphere Application Server. This feature is the Liberty profile for WebSphere Application Server V8.5.5, and it is intended to be installed by itself.

Java SE considerations

WebSphere® eXtreme Scale requires Java™ SE 5, Java SE 6, or Java SE 7. In general, newer versions of Java SE have better functionality and performance.

Supported versions

You can use WebSphere eXtreme Scale with Java SE 5, Java SE 6, and Java SE 7. The version that you use must be currently supported by the Java Runtime Environment (JRE) vendor. If you want to use Secure Sockets Layer (SSL), you must use an IBM® Runtime Environment.
IBM Runtime Environment, Java Technology Edition Version 5, Version 6 , and Version 7 are supported for general use with the product. Version 6 Service Release 9 Fix Pack 2 is a fully supported JRE that is installed as a part of the stand-alone WebSphere eXtreme Scale and WebSphere eXtreme Scale Client installations in the *wxs_install_root*/java directory and is available to be used by both clients and servers. If you are installing WebSphere eXtreme Scale within WebSphere Application Server, you can use the JRE that is included in the WebSphere Application Server installation. For the web console, you must use IBM Runtime Environment, Java Technology Edition Version 6 Service Release 7 and later service releases only.

WebSphere eXtreme Scale takes advantage of Java Development Kit (JDK) Version 5 , Version 6 , and Version 7 functionality as it becomes available. Generally, newer versions of the Java Development Kit (JDK) and Java SE have better performance and functionality.

For more information, see Supported software.

WebSphere eXtreme Scale features that are dependent on Java SE

Table 1. Features that require Java SE 5, Java SE 6, and Java SE 7 . WebSphere eXtreme Scale uses functionality that is introduced in Java SE 5 or Java SE 6 to provide the following product features.

Feature	Supported in Java SE 5 and later service releases	Supported in Java SE Version 6 , Version 7 and later service releases
EntityManager API annotations (Optional: You can also use XML files)	X	X
Java Persistence API (JPA): JPA loader, JPA client loader, and JPA time-based updater	X	X
Memory-based eviction (uses MemoryPoolMXBean)	X	X
Instrumentation agents: • wxssizeagent.jar: Increases the accuracy of the used bytes map metrics. • ogagent.jar: Increases the performance of field-access entities.	X	X
Web console for monitoring		X

Upgrading the JDK in WebSphere eXtreme Scale

Common questions about the upgrade process for releases of WebSphere eXtreme Scale in both stand-alone and WebSphere Application Server environments follow:

- How do I upgrade the JDK that is included in WebSphere eXtreme Scale Version 7.1.0 for a stand-alone environment?
  WebSphere eXtreme Scale Version 7.1.0 included a JRE which is automatically installed to the *wxs_install_root*/java directory. WebSphere eXtreme Scale Version 7.1.0 GA shipped with IBM Java Runtime Environment (JRE) Version 1.6 SR4. To update the JRE shipped with WebSphere eXtreme Scale 7.1.0 requires the application of WebSphere eXtreme Scale 7.1.0 Refresh Pack 0000001 followed by iFIX IFPM56253. For more information, see http://www-304.ibm.com/support/docview.wss?uid=swg1PM56253

  Note: There is currently no interim fix available to upgrade a JRE that was included with WebSphere eXtreme Scale Version 7.0.x.x.
  CAUTION:
  Errors might occur in stand-alone configurations of WebSphere eXtreme Scale that are using the Java 6 runtime and SSL Transport Layer security.
- How do I upgrade the JDK that is included with WebSphere eXtreme Scale for WebSphere Application Server?
  You need to use the JDK upgrade process that is made available by WebSphere Application Server. For more information, see http://www-304.ibm.com/support/docview.wss?uid=swg21427178.
- Which version of the JDK should I use when using WebSphere eXtreme Scale in a WebSphere Application Server environment?
  You can use any level of JDK that is supported by WebSphere Application Server, for the supported version of WebSphere Application Server.

Related concepts:
Tuning Java virtual machines
Java EE considerations
Tuning garbage collection with WebSphere Real Time
Related reference:
startOgServer script
Related information:
Tuning the IBM virtual machine for Java

Java EE considerations

As you prepare to integrate WebSphere® eXtreme Scale in a Java™ Platform, Enterprise Edition environment, consider certain items, such as versions, configuration options, requirements and limitations, and application deployment and management.

Running eXtreme Scale applications in a Java EE environment

A Java EE application can connect to a remote eXtreme Scale application. Additionally, the WebSphere Application Server environment supports starting an eXtreme Scale server when an application starts in the application server.

If you use an XML file to create an ObjectGrid instance, and the XML file is in the module of the enterprise archive (EAR) file, access the file by using the getClass().getClassLoader().getResource("META-INF/objGrid.xml") method to obtain a URL object to use to create an ObjectGrid instance. Substitute the name of the XML file that you are using in the method call.

You can use startup beans for an application to bootstrap an ObjectGrid instance when the application starts, and to destroy the instance when the application stops. A startup bean is a stateless session bean with a com.ibm.websphere.startupservice.AppStartUpHome remote location and a com.ibm.websphere.startupservice.AppStartUp remote interface. The remote interface has two methods: the start method and the stop method. Use the start method to bootstrap the instance, and use the stop method to destroy the instance. The application uses the ObjectGridManager.getObjectGrid method to maintain a reference to the instance. See Interacting with an ObjectGrid using the ObjectGridManager interface for more information.

Using class loaders

When application modules that use different class loaders share a single ObjectGrid instance in a Java EE application, verify the objects that are stored in eXtreme Scale and the plug-ins for the product are in a common loader in the application.

Managing the life cycle of ObjectGrid instances in a servlet

To manage the life cycle of an ObjectGrid instance in a servlet, you can use the init method to create the instance and the destroy method to remove the instance. If the instance is cached, it is retrieved and manipulated in the servlet code. See Interacting with an ObjectGrid using the ObjectGridManager interface for more information.

Related concepts:
Tuning Java virtual machines
Java SE considerations
Tuning garbage collection with WebSphere Real Time
Related reference:
startOgServer script
Related information:
Tuning the IBM virtual machine for Java

Directory conventions

The following directory conventions are used throughout the documentation to 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

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*

Runtime files for WebSphere eXtreme Scale integrated with WebSphere Application Server

Java™ archive (JAR) files are included in the installation. You can see the JAR files that are included and the location to which they are installed.

Table 1. Runtime files for WebSphere eXtreme Scale. The following table lists the Java archive (JAR) files that are included in the installation. The installation location is relative to the *wxs_home* directory that you choose during the installation.

File name	Environment	Installation location	Description
wxsdynacache.jar	Client and server	lib	The wxsdynacache.jar file contains the necessary classes to use with the dynamic cache provider.
wsobjectgrid.jar	Local and client	lib	The wsobjectgrid.jar contains the eXtreme Scale local, client, and server run times.
ogagent.jar	Local, client, and server	lib	The ogagent.jar file contains the runtime classes that are required to run the Java instrumentation agent that is used with the EntityManager API.
sessionobjectgrid.jar	Client and server	lib	The sessionobjectgrid.jar file contains the eXtreme Scale HTTP session management runtime.
wsogclient.jar	Local and client	lib	The wsogclient.jar file installed when you use an environment that contains WebSphere® Application Server Version 6.0.2 and later. This file contains only the local and client runtime environments.
wxssizeagent.jar	Local, client, and server	lib	The wxssizeagent.jar file is used to provide more accurate cache entry sizing information when using Java runtime environment (JRE) Version 1.5 or later.
oghibernate-cache.jar	Client and server	optionalLibraries/ObjectGrid	The oghibernate-cache.jar file contains the eXtreme Scale level 2 cache plug-in for JBoss Hibernate.

File name	Environment	Installation location	Description
ogspring.jar	Local, client, and server	optionalLibraries/ObjectGrid	The ogspring.jar file contains support classes for the SpringSource Spring framework integration.
xsadmin.jar	Utility	optionalLibraries/ObjectGrid	The xsadmin.jar file contains the eXtreme Scale administration sample utility.
ibmcfw.jar ibmext.jar ibmorb.jar ibmorbapi.jar	Client and server	optionalLibraries/ObjectGrid/endorsed	This set of files includes the Object Request Broker (ORB) runtime that is used for running applications in Java SE processes.
wxshyperic.jar	Utility	optionalLibraries/ObjectGrid/hyperic/lib	The WebSphere eXtreme Scale server detection plug-in for the SpringSource Hyperic monitoring agent.
restservice.ear	Client	optionalLibraries/ObjectGrid/restservice/lib	The restservice.ear file contains the eXtreme Scale REST data service application enterprise archive for WebSphere Application Server environments.
restservice.war	Client	optionalLibraries/ObjectGrid/restservice/lib	The restservice.war file contains the eXtreme Scale REST data service Web archive for application servers acquired from another vendor.
splicerlistener.jar	Utility	optionalLibraries/ObjectGrid/session/lib	The splicerlistener.jar file contains the splicer utility for the eXtreme Scale HTTP session manager filter.
splicer.jar	Utility	optionalLibraries/ObjectGrid/legacy/session/lib	The splicer.jar contains the Version 7.0 splicer utility for the eXtreme Scale HTTP session manager filter.

Table 2. Runtime files for WebSphere eXtreme Scale Client. The following table lists the Java archive (JAR) files that are included in the installation. The installation location is relative to the *wxs_home* directory that you choose during the installation.

File name	Environment	Installation location	Description
wxsdynacache.jar	Client and server	lib	The wxsdynacache.jar file contains the necessary classes to use with the dynamic cache provider.
ogagent.jar	Local, client, and server	lib	The ogagent.jar file contains the runtime classes that are required to run the Java instrumentation agent that is used with the EntityManager API.
sessionobjectgrid.jar	Client and server	lib	The sessionobjectgrid.jar file contains the eXtreme Scale HTTP session management runtime.
wsogclient.jar	Local and client	lib	The wsogclient.jar file installed when you use an environment that contains WebSphere Application Server Version 6.0.2 and later. This file contains only the local and client runtime environments.
wxssizeagent.jar	Local, client, and server	lib	The wxssizeagent.jar file is used to provide more accurate cache entry sizing information when using Java runtime environment (JRE) Version 1.5 or later.
oghibernate-cache.jar	Client and server	optionalLibraries/ObjectGrid	The oghibernate-cache.jar file contains the eXtreme Scale level 2 cache plug-in for JBoss Hibernate.
ogspring.jar	Local, client, and server	optionalLibraries/ObjectGrid	The ogspring.jar file contains support classes for the SpringSource Spring framework integration.
xsadmin.jar	Utility	optionalLibraries/ObjectGrid	The xsadmin.jar file contains the eXtreme Scale administration sample utility.
ibmcfw.jar ibmext.jar ibmorb.jar ibmorbapi.jar	Client and server	optionalLibraries/ObjectGrid/endorsed	This set of files includes the Object Request Broker (ORB) runtime that is used for running applications in Java SE processes.
wxshyperic.jar	Utility	optionalLibraries/ObjectGrid/hyperic/lib	The WebSphere eXtreme Scale server detection plug-in for the SpringSource Hyperic monitoring agent.
restservice.ear	Client	optionalLibraries/ObjectGrid/restservice/lib	The restservice.ear file contains the eXtreme Scale REST data service application enterprise archive for WebSphere Application Server environments.
restservice.war	Client	optionalLibraries/ObjectGrid/restservice/lib	The restservice.war file contains the eXtreme Scale REST data service Web archive for application servers acquired from another vendor.
splicerlistener.jar	Utility	optionalLibraries/ObjectGrid/session/lib	The splicerlistener.jar file contains the splicer utility for the eXtreme Scale HTTP session manager filter.

File name	Environment	Installation location	Description
splicer.jar	Utility	optionalLibraries/ObjectGrid/legacy/session/lib	The splicer.jar contains the Version 7.0 splicer utility for the eXtreme Scale HTTP session manager filter.

Runtime files for WebSphere eXtreme Scale stand-alone installation

Java™ archive (JAR) files are included in the installation. You can see the JAR files that are included and the location to which they are installed.

Table 1. Runtime files for WebSphere eXtreme Scale full installation. WebSphere® eXtreme Scale relies on ObjectGrid processes and related APIs. The following table lists the JAR files that are included in the installation. The installation location is relative to the *wxs_home* directory that you choose during the installation.

File name	Environment	Installation location	Description
wxsdynacache.jar	Client and server	dynacache/lib	The wxsdynacache.jar file contains the necessary classes to use with the dynamic cache provider. The file is automatically included in the server runtime environment when you use the supplied scripts.
wxshyperic.jar	Utility	hyperic/lib	The WebSphere eXtreme Scale server detection plug-in for the SpringSource Hyperic monitoring agent.
objectgrid.jar	Local, client, and server	lib	The objectgrid.jar file is an OSGi bundle that is used by the server runtime environment of Java SE 1.5 and later. The file is automatically included in the server runtime environment when you use the supplied scripts.
ogagent.jar	Local, client, and server	lib	The ogagent.jar file contains the runtime classes that are required to run the Java instrumentation agent that is used with the EntityManager API.
ogclient.jar	Local and client	lib	The ogclient.jar file is an OSGi bundle that contains only the local and client runtime environments. You can use this file with Java SE 1.5 and later.
ogspring.jar	Local, client, and server	lib	The ogspring.jar file contains support classes for the SpringSource Spring framework integration.
wsogclient.jar	Local and client	lib	The wsogclient.jar file installed when you use an environment that contains WebSphere Application Server Version 6.1 and later. This file contains only the local and client runtime environments.
wxssizeagent.jar	Local, client, and server	lib	The wxssizeagent.jar file is used to provide more accurate cache entry sizing information when using Java runtime environment (JRE) Version 1.5 and later.
ibmcfw.jar ibmorb.jar ibmorbapi.jar	Client and server	lib/endorsed	This set of files includes the Object Request Broker (ORB) runtime that is used for running applications in Java SE processes.
restservice.ear	Client	restservice/lib	The restservice.ear file contains the eXtreme Scale REST data service application enterprise archive for WebSphere Application Server environments.
restservice.war	Client	restservice/lib	The restservice.war file contains the eXtreme Scale REST data service Web archive for application servers acquired from another vendor.
xsadmin.jar	Utility	samples	The xsadmin.jar file contains the eXtreme Scale administration sample utility.
sessionobjectgrid.jar	Client and server	session/lib	The sessionobjectgrid.jar file contains the eXtreme Scale HTTP session management runtime.
splicerlistener.jar	Utility	session/lib	The splicerlistener.jar file contains the splicer utility for the eXtreme Scale Version 7.1 and later HTTP session listener.
xsgbean.jar	Server	wasce/lib	The xsgbean.jar file contains the GBean for embedding eXtreme Scale servers in WebSphere Application Server Community Edition application servers.
splicer.jar	Utility	legacy/session/lib	The splicer utility for the WebSphere eXtreme Scale Version 7.0 HTTP session manager filter.
wxsra.rar	Client and server	session/lib	The wxsra.rar contains the eXtreme Scale resource adapter to connect to the grid using a connection factory.

Table 2. Runtime files for WebSphere eXtreme Scale Client. WebSphere eXtreme Scale Client relies on ObjectGrid processes and related APIs. The following table lists the JAR files that are included in the installation. The installation location is relative to the *wxs_home* directory that you choose during the installation.

File name	Environment	Installation location	Description
wxsdynacache.jar	Client and server	dynacache/lib	The wxsdynacache.jar file contains the necessary classes to use with the dynamic cache provider. The file is automatically included in the server runtime environment when you use the supplied scripts.
wxshyperic.jar	Utility	hyperic/lib	The WebSphere eXtreme Scale server detection plug-in for the SpringSource Hyperic monitoring agent.
ogagent.jar	Local, client, and server	lib	The ogagent.jar file contains the runtime classes that are required to run the Java instrumentation agent that is used with the EntityManager API.
ogclient.jar	Local and client	lib	The ogclient.jar file is an OSGi bundle that contains only the local and client runtime environments. You can use this file with Java SE 1.5 and later.
ogspring.jar	Local, client, and server	lib	The ogspring.jar file contains support classes for the SpringSource Spring framework integration.
wsogclient.jar	Local and client	lib	The wsogclient.jar file installed when you use an environment that containsWebSphere Application Server Version 6.1 and later. This file contains only the local and client runtime environments.
wxssizeagent.jar	Local, client, and server	lib	The wxssizeagent.jar file is used to provide more accurate cache entry sizing information when using Java runtime environment (JRE) Version 1.5 and later.
ibmcfw.jar ibmorb.jar ibmorbapi.jar	Client and server	lib/endorsed	This set of files includes the Object Request Broker (ORB) runtime that is used for running applications in Java SE processes.
restservice.ear	Client	restservice/lib	The restservice.ear file contains the eXtreme Scale REST data service application enterprise archive for WebSphere Application Server environments.
restservice.war	Client	restservice/lib	The restservice.war file contains the eXtreme Scale REST data service Web archive for application servers acquired from another vendor.
xsadmin.jar	Utility	samples	The xsadmin.jar file contains the eXtreme Scale administration sample utility.
sessionobjectgrid.jar	Client and server	session/lib	The sessionobjectgrid.jar file contains the eXtreme Scale HTTP session management runtime.
splicerlistener.jar	Utility	session/lib	The splicerlistener.jar file contains the splicer utility for the eXtreme Scale Version 7.1 and later HTTP session listener.
splicer.jar	Utility	legacy/session/lib	The splicer utility for the WebSphere eXtreme Scale Version 7.0 HTTP session manager filter.
wxsra.rar	Client and server	session/lib	The wxsra.rar contains the eXtreme Scale resource adapter to connect to the grid using a connection factory.

IBM Installation Manager and WebSphere eXtreme Scale product offerings

WebSphere® eXtreme Scale product offerings are available in product repositories. To access these repositories, you must first install IBM® Installation Manager.

You can install Installation Manager with files available on the product media, or with a file obtained from the Passport Advantage® site. You can also download a file from the IBM Installation Manager download website. A file is a compressed file that contains installation images.

Installation Manager gives you access to the necessary product repositories. You must access these repositories to install the WebSphere eXtreme Scale product offerings.

There are two options to access product repositories.
Option 1: Access product repositories on the physical media, and use local installation

1. Install Installation Manager on your system.
2. With Installation Manager, Install the product offering from the product repositories on the media.

Option 2: Download product repositories from Passport Advantage, and use local installation

1. Download the repositories from the Passport Advantage site.
   Note: See see Supported software for a list of the IBM WebSphere eXtreme Scale installation images downloadable from the IBM Passport Advantage Online website and other information.
2. Install Installation Manager on your system.
3. Install the product from downloaded product repositories with Installation Manager.

- 8.5+ Installing IBM Installation Manager using the GUI
  To access the necessary product repositories so that you can install WebSphere eXtreme Scale product offerings, you must install IBM Installation

Manager. You can install Installation Manager using a GUI.

- 8.5+ Installing IBM Installation Manager using the command line
  In order to access the necessary product repositories so that you can install WebSphere eXtreme Scale product offerings, you must install IBM Installation Manager. You can install Installation Manager from the command line.
- 8.5+ Installing IBM Installation Manager using response files
  In order to access the necessary product repositories so that you can install WebSphere eXtreme Scale product offerings, you must install IBM Installation Manager. You can install Installation Manager using response files.

Related tasks:
Installing IBM Installation Manager using the GUI
Installing the product with the GUI
Uninstalling the product using the GUI
Installing IBM Installation Manager using the command line
Installing the product using the command line
Uninstalling the product using the command line
Installing IBM Installation Manager using response files
Installing the product using a response file
Uninstalling the product using response files
Installing fix packs using the GUI
Uninstalling fix packs using the GUI
Installing fix packs using the command line
Uninstalling fix packs using the command line
Installing fix packs using a response file
Uninstalling fix packs using response files

Installing IBM Installation Manager using the GUI

To access the necessary product repositories so that you can install WebSphere® eXtreme Scale product offerings, you must install IBM® Installation Manager. You can install Installation Manager using a GUI.

Before you begin

You must install IBM Installation Manager and have access to necessary repositories. For more information, see IBM Installation Manager and WebSphere eXtreme Scale product offerings.

Procedure

1. From the location that contains the Installation Manager installation files, run one of the following commands:

   Administrative installation:
   - Windows `install.exe`
   - UNIX Linux `./install`

   Non-administrative installation:
   - Windows `userinst.exe`
   - UNIX Linux `./userinst`

   For more information about administrative and non-administrative installations, see Install as an administrator, nonadministrator, or group

   Group-mode installation:
   UNIX Linux `./groupinst`

   Notes on group mode:
   - With group mode, multiple users can use a single instance of IBM Installation Manager to manage software packages.
     Group mode does not enable two people to use a single instance of IBM Installation Manager at the same time.
   - Windows Group mode is not available on Windows operating systems.
   - If you do not install Installation Manager with group mode, you cannot use group mode to manage any of the products that you install later using this Installation Manager instance.
   - Change the installation location for the current user from the default location to a location that is accessible by all users in the group.
   - Set up your groups, permissions, and environment variables as described in the Group mode road maps in the IBM Installation Manager Version 1.5 Information Center before installing in group mode.
   - For more information about using group mode, read the Group mode road maps in the IBM Installation Manager Version 1.5 Information Center.

2. Verify that the Installation Manager package is selected, and click Next.
3. Accept the terms in the license agreements, and click Next.
4. Click Next.
5. Review the summary information, and click Install. If the installation is successful, the program displays a message indicating that installation is successful. If the installation is not successful, click View Log File to troubleshoot the problem.
6. Add the product repository to your Installation Manager preferences.
   a. Start Installation Manager.
   b. In the top menu, click File > Preferences.
   c. Select Repositories.
   d. Click **Add Repository**.
   e. Enter the path to the `repository.config` file in the location that contains the repository files, for example:

- Windows `C:\repositories\product_name\local-repositories`
- UNIX Linux `/var/repositories/product_name/local-repositories`

f. Click OK.

7. Clear any locations listed in the Repositories window that you are not using.
8. Click Apply.
9. Click OK.
10. Click File > Exit to close Installation Manager.

What to do next

After you successfully install Installation Manager and set up the repository, you can continue to install any WebSphere eXtreme Scale stand-alone or WebSphere eXtreme Scale for WebSphere Application Server for product offering. For more information, see Installing the product with the GUI

- 8.5+ Installing the product with the GUI
  Use the Installation Manager GUI to install WebSphere eXtreme Scale product offerings.

Related concepts:
Installation overview
IBM Installation Manager and WebSphere eXtreme Scale product offerings
Related tasks:
Installing the product with the GUI
Uninstalling the product using the GUI

Installing the product with the GUI

Use the Installation Manager GUI to install WebSphere® eXtreme Scale product offerings.

Before you begin

- You must install the necessary product files for the Installation Manager and have access to necessary repositories. For more information, see IBM Installation Manager and WebSphere eXtreme Scale product offerings.

Procedure

1. Start Installation Manager.
   UNIX Linux Tip: You can start Installation Manager in group mode with the **./IBMIM** command.
   - Group mode allows multiple users to use a single instance of IBM Installation Manager to manage software packages.
   - For more information about using group mode, read the Group mode roadmaps in the IBM Installation Manager Version 1.5 Information Center.
2. Click Install.
   Note: If you are prompted to authenticate, use the IBM ID and password that you registered with on the program website.
   Installation Manager searches its defined repositories for available packages.
3. Select one of the following product offerings and the appropriate version.
   - WebSphere eXtreme Scale in a stand-alone environment
   - WebSphere eXtreme Scale Client in a stand-alone environment
   - WebSphere eXtreme Scale for WebSphere Application Server Version 6.1
   - WebSphere eXtreme Scale for WebSphere Application Server Version 7
   - WebSphere eXtreme Scale for WebSphere Application Server Version 8
   - WebSphere eXtreme Scale Client for WebSphere Application Server Version 6.1
   - WebSphere eXtreme Scale Client for WebSphere Application Server Version 7
   - WebSphere eXtreme Scale Client for WebSphere Application Server Version 8

   If you already have the product that is installed on your system, a message indicates that the product is already installed. To create another installation of the product in another location, click Continue.

   Tip: If **Search service repositories during installation and updates** option is selected on the Installation Manager Repository preference page and you are connected to the Internet, you can click **Check for Other Versions and Extensions**. By doing so, you can search for updates in the default update repositories for the selected packages. In this case, you do not need to add the specific service-repository URL to the Installation Manager Repository preference page.
   a. Select the fixes to install.
      Any suggested fixes are selected by default.

      If there are suggested fixes, you can select the option to show only suggested fixes and hide non-recommended fixes.

   b. Click Next.

   Note: Installation Manager might prompt you to update to the latest level of Installation Manager when it connects to the repository. If you are prompted, update to the newer version before you continue. Read the IBM Installation Manager Version 1.5 Information Center for information about automatic updates.
4. Accept the terms in the license agreements, and click Next.
5. Specify the installation root directory for the product.
   The panel also specifies the shared resources directory and disk-space information.
   Note: The first time that you install a package with Installation Manager, specify the shared resources directory. The shared resources directory is where installation artifacts are located that can be used by one or more package groups. Use your largest drive for this installation. You cannot change the

directory location until after you uninstall all packages.
Restrictions:

- Deleting the default target location and leaving an installation-directory field empty prevents you from continuing.
- Do not use symbolic links as the destination directory.
  Symbolic links are not supported.
- Do not use a semicolon in the directory name.
  If the target directory includes a semicolon, the WebSphere eXtreme Scale does not install as expected.

  Windows A semicolon is the character that is used to construct the class path on Windows systems.
- Windows The maximum path length on the Windows Server 2008, Windows Vista, and Windows 7 operating systems is 60 characters.

6. Click Next.
7. Select the features that you want to install.
   Depending on which product offering you selected, you can choose from the following features:
   - Client
     Available as a required feature if you install either WebSphere eXtreme Scale in a stand-alone environment or WebSphere eXtreme Scale for WebSphere Application Server product offerings. The client must be installed for these product offerings.
   - Server
     Available if you to install either WebSphere eXtreme Scale in a stand-alone environment or WebSphere eXtreme Scale for WebSphere Application Server. You can choose not to install the server for these product offerings.
   - Console
     Available for all WebSphere eXtreme Scale product offerings. You can choose to install the monitoring console. With the web console, you can chart current and historical statistics. This console provides some charts for high-level overviews, and has a custom reports page that you can use to build charts from the available statistics. You can use the charting capabilities in the monitoring console of WebSphere eXtreme Scale to view the overall performance of the data grids in your environment.
   - Samples
     Available for all WebSphere eXtreme Scale product offerings.
8. Click Next.
9. Review the summary information, and click Install.
   - If the installation is successful, the program gives you a message to indicate that installation is successful.
     Note: The program might also specify important post-installation instructions as well.
   - If the installation is not successful, click View Log File to troubleshoot the problem.
10. Select which tool you want to start when this installation is finished.
    - Select Profile Management Tool to create a profile if you want to create a new application server profile with settings appropriate for a production environment.
    - Select Profile Management Tool to create an application server profile for a development environment if you want to create an application server profile with settings appropriate for a development environment.
      Note: The development settings are appropriate for a development environment where frequent application updates are done and system resources are at a minimum. Do not use the development settings for production servers.
    - Select None if you do not want to create a new profile when this installation is finished.

    Restriction: The option to start the Profile Management tool is only available when a version of WebSphere Application Server containing the Profile Management tool is installed.
11. Click Finish.
12. Click File > Exit to close Installation Manager.

Related concepts:
Installation overview
IBM Installation Manager and WebSphere eXtreme Scale product offerings
Related tasks:
Installing IBM Installation Manager using the GUI
Uninstalling the product using the GUI

Installing IBM Installation Manager using the command line

In order to access the necessary product repositories so that you can install WebSphere® eXtreme Scale product offerings, you must install IBM® Installation Manager. You can install Installation Manager from the command line.

Before you begin

You must install the necessary product files for the Installation Manager and have access to necessary repositories. For more information, see IBM Installation Manager and WebSphere eXtreme Scale product offerings.

Procedure

1. Change to the location containing the Installation Manager installation files, and run one of the following commands:

   Administrative installation:
   - Windows `installc.exe -acceptLicense -log` *`log_file_path_and_name`*
   - UNIX Linux `./installc -acceptLicense -log` *`log_file_path_and_name`*

Non-administrative installation:

- Windows `userinstc.exe -acceptLicense -log log_file_path_and_name`
- UNIX Linux `./userinstc -acceptLicense -log log_file_path_and_name`

Group-mode installation:

UNIX Linux `./groupinstc -acceptLicense -dataLocation application_data_location -log log_file_path_and_name -installationDirectory Installation_Manager_home`

Notes on group mode:

- Group mode allows multiple users to use a single instance of IBM Installation Manager to manage software packages.
- Windows Group mode is not available on Windows operating systems.
- If you do not install Installation Manager using group mode, you will not be able to use group mode to manage any of the products that you install later using this Installation Manager.
- Make sure that you change the installation location from the default location in the current user's home directory to a location that is accessible by all users in the group.
- Set up your groups, permissions, and environment variables as described in the Group mode road maps in the IBM Installation Manager Version 1.5 Information Center before installing in group mode.
- For more information about using group mode, read the Group mode road maps in the IBM Installation Manager Version 1.5 Information Center.

2. Optional: If the repository requires a user name and password, create a keyring file to access this repository.
   For more information about creating a keyring file for Installation Manager, read the IBM Installation Manager Version 1.5 Information Center.

   Tip: When creating a keyring file, append /repository.config at the end of the repository URL location if the **imutilsc** command is unable to find the URL that is specified.

What to do next

After successfully installing Installation Manager and setting up the repository, you can continue to install any WebSphere eXtreme Scale stand-alone or WebSphere eXtreme Scale for WebSphere Application Server product offering. For more information, see Installing the product with the GUI.

- 8.5+ Installing the product using the command line
  Use the Installation Manager from the command line to install WebSphere eXtreme Scale product offerings.

Related concepts:
Installation overview
IBM Installation Manager and WebSphere eXtreme Scale product offerings
Related tasks:
Installing the product using the command line
Uninstalling the product using the command line

Installing the product using the command line

Use the Installation Manager from the command line to install WebSphere® eXtreme Scale product offerings.

Before you begin

You must install the necessary product files for the Installation Manager and have access to necessary repositories. For more information, seeIBM Installation Manager and WebSphere eXtreme Scale product offerings.

Procedure

1. Log on to your system.
2. Change to the eclipse/tools subdirectory in the directory where you installed Installation Manager.
3. Verify that the product repository is available.
   Windows

   ```
   imcl.exe listAvailablePackages -repositories source_repository
   ```

   UNIX Linux

   ```
   ./imcl listAvailablePackages -repositories source_repository
   ```

   You should see one or more levels of the offering.
4. Use the **imcl** command to install the product.
   Windows

   ```
   imcl install offeringID_version,optionalFeatureID
     -repositories source_repository
     -installationDirectory installation_directory
     -sharedResourcesDirectory shared_directory
     -accessRights access_mode
     -preferences preference_key=value
     -properties property_key=value
     -keyring keyring_file -password password
     -acceptLicense
   ```

   For example: `imcl install com.ibm.websphere.WXS.v86_8.6.0.20110503_0200,xs.console.feature`

UNIX Linux

```
./imcl install offeringID_version,optionalFeatureID
  -repositories source_repository
  -installationDirectory installation_directory
  -sharedResourcesDirectory shared_directory
  -accessRights access_mode
  -preferences preference_key=value
  -properties property_key=value
  -keyring keyring_file -password password
  -acceptLicense
```

Tips:

- The *offeringID* is the offering ID that is listed in Offering IDs for WebSphere eXtreme Scale product offerings.
- The *version*, which optionally can be attached to the offering ID with an underscore, is a specific version of the offering to install (`8.5.0.20110503_0200` for example).
  - If *version* is **not** specified, the latest version of the offering and **all** interim fixes for that version are installed.
  - If *version* is specified, the specified version of the offering and **no** interim fixes for that version are installed.

  The version can be found attached to the end of the offering ID with an underscore when you run the following command against the repository:

  ```
  imcl listAvailablePackages -repositories source_repository
  ```

- You can also specify `none`, `recommended` or `all` with the `-installFixes` argument to indicate which interim fixes you want installed with the offering.
  - If the offering version is **not** specified, the `-installFixes` option defaults to `all`.
  - If the offering version is specified, the `-installFixes` option defaults to `none`.
- You can add a list of features that are separated by commas. An example follows:

  ```
  imcl -acceptLicense install com.ibm.websphere.WXS.v85,xs.console.feature,xs.samples.feature
  ```

  ```
  imcl -acceptLicense install com.ibm.websphere.WXS.v86,xs.console.feature,xs.samples.feature
  ```

  - `xs.client.standalone.feature` Available as a required feature if you install WebSphere eXtreme Scale in a stand-alone environment.
  - `xs.server.standalone.feature` Depending on which product offering you want to install, you can choose to install the server. The server is a selectable feature in the following product offerings:
    - WebSphere eXtreme Scale in a stand-alone environment
  - `xs.console.feature` Available for all product offerings. You can choose to install the monitoring console. With the web console, you can chart current and historical statistics. This console provides some preconfigured charts for high-level overviews, and has a custom reports page that you can use to build charts from the available statistics. You can use the charting capabilities in the monitoring console of WebSphere eXtreme Scale to view the overall performance of the data grids in your environment
  - `xs.samples.feature` Available for all product offerings. You can choose to install samples.

  Notes:
- If you previously specified the mode in which to install Installation Manager, the -accessRights parameter is not required
- If you experience issues later, Installation Manager can save earlier versions of a package to roll back to. When Installation Manager rolls back a package to a previous version, the current version of the files is uninstalled and the earlier versions are reinstalled. If you choose not to save the files for rollback, you can prevent the files from being saved by specifying a preference:

  ```
  -preference com.ibm.cic.common.core.preferences.preserveDownloadedArtifacts=False
  ```

  For more information about setting your Installation Manager preferences, see the IBM Installation Manager Version 1.5 Information Center.

  Tip: Even if you choose not to save files for rollback, you can still access product files for rollback from the repository.
- The program might write important post-installation instructions to standard output.

For more information about using the **imcl** command to install the product, see the IBM Installation Manager Version 1.5 Information Center.

Related concepts:
Installation overview
IBM Installation Manager and WebSphere eXtreme Scale product offerings
Related tasks:
Installing IBM Installation Manager using the command line
Uninstalling the product using the command line

Installing IBM Installation Manager using response files

In order to access the necessary product repositories so that you can install WebSphere® eXtreme Scale product offerings, you must install IBM® Installation Manager. You can install Installation Manager using response files.

Before you begin

You must install the necessary product files for the Installation Manager and have access to necessary repositories. For more information, see IBM Installation Manager and WebSphere eXtreme Scale product offerings.

Procedure

Change to the location containing the Installation Manager installation files, and run one of the following commands to install Installation Manager.

Administrative installation:

- Windows `installc.exe -acceptLicense -log log_file_path_and_name`
- UNIX Linux `./installc -acceptLicense -log log_file_path_and_name`

Non-administrative installation:

- Windows `userinstc.exe -acceptLicense -log log_file_path_and_name`
- UNIX Linux `./userinstc -acceptLicense -log log_file_path_and_name`

Group-mode installation:

UNIX Linux `./groupinstc -acceptLicense -dataLocation application_data_location -log log_file_path_and_name -installationDirectory Installation_Manager_home`

Notes on group mode:

- Group mode allows multiple users to use a single instance of IBM Installation Manager to manage software packages. Group mode does not mean that two people can use the single instance of IBM Installation Manager at the same time.
- Windows Group mode is not available on Windows operating systems.
- If you do not install Installation Manager using group mode, you will not be able to use group mode to manage any of the products that you install later using this Installation Manager.
- Make sure that you change the installation location from the default location in the current user's home directory to a location that is accessible by all users in the group.
- Set up your groups, permissions, and environment variables as described in the Group mode road maps in the IBM Installation Manager Version 1.5 Information Center before installing in group mode.
- For more information about using group mode, read the Group mode road maps in the IBM Installation Manager Version 1.5 Information Center.

What to do next

After successfully installing Installation Manager and setting up the repository, you can continue to install any WebSphere eXtreme Scale stand-alone or WebSphere eXtreme Scale for WebSphere Application Server for product offering. For more information, see Installing the product with the GUI.

- 8.5+ Installing the product using a response file
  Use the Installation Manager with a response file to install WebSphere eXtreme Scale product offerings.

Related concepts:
Installation overview
IBM Installation Manager and WebSphere eXtreme Scale product offerings
Related tasks:
Installing the product using a response file
Uninstalling the product using response files

Installing the product using a response file

Use the Installation Manager with a response file to install WebSphere® eXtreme Scale product offerings.

Before you begin

You must install the necessary product files for the Installation Manager and have access to necessary repositories. For more information, seeIBM Installation Manager and WebSphere eXtreme Scale product offerings.

About this task

Using Installation Manager, you can record a response file using the GUI.

Procedure

1. From a command line, change to the Eclipse subdirectory in the directory where you installed Installation Manager.
2. Start Installation Manager from the command line using the -record option.
   For example:
   - Windows **Administrator or non-administrator:**

     ```
     IBMIM.exe -skipInstall "C:\temp\imRegistry"
       -record C:\temp\install_response_file.xml
     ```

   - UNIX Linux **Administrator:**

     ```
     ./IBMIM -skipInstall /var/temp/imRegistry
       -record /var/temp/install_response_file.xml
     ```

   - UNIX Linux **Non-administrator:**

     ```
     ./IBMIM -skipInstall user_home/var/temp/imRegistry
       -record user_home/var/temp/install_response_file.xml
     ```

   Tip: When you record a new response file, you can specify the -skipInstall parameter. Using this parameter has the following benefits:
   - No files are installed, and this speeds up the recording.

- If you use a temporary data location with the -skipInstall parameter, Installation Manager writes the installation registry to the specified data location while recording. When you start Installation Manager again without the -skipInstall parameter, you then can use your response file to install against the real installation registry.
  The -skipInstall operation should not be used on the actual agent data location used by Installation Manager. This operation is unsupported. Use a clean writable location, and reuse that location for future recording sessions.

For more information, read the IBM Installation Manager Version 1.5 Information Center.

3. Add the appropriate repositories to your Installation Manager preferences.
   a. In the top menu, click File > Preferences
   b. Select Repositories
   c. Perform the following actions for each repository:
      i. Click **Add Repository**.
      ii.
      iii. Enter the path to the `repository.config` file in the remote web-based repository or the local directory into which you unpacked the repository files.
      For example:
      - Remote repositories:

        ```
        https://downloads.mycorp.com:8080/WXS_85_repository
        ```

      - Local repositories:
        - Windows `C:\repositories\wxs85\local-repositories`
        - UNIX Linux `/var/repositories/wxs85/local-repositories`

      iv. Click OK.
      v. Click Apply.
      vi. Click OK.
   d. Click **Install**.
   Note: If you are prompted to authenticate, use the IBM ID and password that you registered with on the program website.
   Installation Manager searches its defined repositories for available packages.

4. Select one the following product offerings and the appropriate version:
   - WebSphere eXtreme Scale in a stand-alone environment
   - WebSphere eXtreme Scale Client in a stand-alone environment
   - WebSphere eXtreme Scale for WebSphere Application Server Version 6.1
   - WebSphere eXtreme Scale for WebSphere Application Server Version 7
   - WebSphere eXtreme Scale for WebSphere Application Server Version 8
   - WebSphere eXtreme Scale Client for WebSphere Application Server Version 6.1
   - WebSphere eXtreme Scale Client for WebSphere Application Server Version 7
   - WebSphere eXtreme Scale Client for WebSphere Application Server Version 8

   If you already have the product installed on your system, a message indicates that the product is already installed. To create another installation of the product in another location, click Continue.

   Tip: If **Search service repositories during installation and updates** option is selected on the Installation Manager Repository preference page and you are connected to the Internet, you can click **Check for Other Versions and Extensions**. By doing so, you can search for updates in the default update repositories for the selected packages. In this case, you do not need to add the specific service-repository URL to the Installation Manager Repository preference page.
5. Select the fixes to install.
   Any recommended fixes are selected by default.

   If there are recommended fixes, you can select the option to show only recommended fixes and hide non-recommended fixes.

6. Click Next.
7. Accept the terms in the license agreements, and click Next.
8. Specify the installation root directory for the product.
   The panel also specifies the shared resources directory and disk-space information.
   Note: The first time that you install a package using Installation Manager, specify the shared resources directory. The shared resources directory is where installation artifacts are located that can be used by one or more package groups. Use your largest drive for this installation. You cannot change the directory location until after you uninstall all packages.
   Restrictions:
   - Deleting the default target location and leaving an installation-directory field empty prevents you from continuing.
   - Do not use symbolic links as the destination directory.
     Symbolic links are not supported.

   - Windows The maximum path length on the Windows Server 2008, Windows Vista, and Windows 7 operating systems is 60 characters.
9. Click Next.
10. Select the features that you want to install.
    - Client
      Available as a required feature if you install either WebSphere eXtreme Scale in a stand-alone environment or WebSphere eXtreme Scale for WebSphere Application Server product offerings. The client must be installed for these product offerings.

    - Server
      Available if you to install either WebSphere eXtreme Scale in a stand-alone environment or WebSphere eXtreme Scale for WebSphere Application Server. You can choose not to install the server for these product offerings.

    - Console
      Available for all WebSphere eXtreme Scale product offerings. You can choose to install the monitoring console. With the web console, you can chart current and historical statistics. This console provides some charts for high-level overviews, and has a custom reports page that you can use to

build charts from the available statistics. You can use the charting capabilities in the monitoring console of WebSphere eXtreme Scale to view the overall performance of the data grids in your environment.

- Samples
  Available for all WebSphere eXtreme Scale product offerings.

11. Click Next.
12. Review the summary information, and click **Install**.
    - If the installation is successful, the program gives you a message to indicate that installation is successful.
      Note: The program might also specify important post-installation instructions as well.
    - If the installation is not successful, click View Log File to troubleshoot the problem.
13. Click **Finish**.
14. Click **File > Exit** to close Installation Manager.

- 8.5+ Creating a keyring
  After using the Installation Manager to record a response file to install WebSphere eXtreme Scale product offerings, you can choose to create a keyring file. If you are using a remote repository that requires authentication, then you can create a keyring for installation.

Related concepts:
Installation overview
IBM Installation Manager and WebSphere eXtreme Scale product offerings
Related tasks:
Installing IBM Installation Manager using response files
Uninstalling the product using response files

Creating a keyring

After using the Installation Manager to record a response file to install WebSphere® eXtreme Scale product offerings, you can choose to create a keyring file. If you are using a remote repository that requires authentication, then you can create a keyring for installation.

Before you begin

You must record a response file. For more information, see Installing the product using a response file.

Procedure

1. From a command line, change to the Eclipse subdirectory in the directory where you installed Installation Manager.
2. Start Installation Manager from the command line using the -record option.
   For example:
   - Windows **Administrator or non-administrator:**

     ```
     IBMIM.exe -skipInstall "C:\temp\imRegistry"
       -keyring C:\IM\im.keyring
       -record C:\temp\keyring_response_file.xml
     ```

   - UNIX Linux **Administrator:**

     ```
     ./IBMIM -skipInstall /var/temp/imRegistry
       -keyring /var/IM/im.keyring
       -record /var/temp/keyring_response_file.xml
     ```

   - UNIX Linux **Non-administrator:**

     ```
     ./IBMIM -skipInstall user_home/var/temp/imRegistry
       -keyring user_home/var/IM/im.keyring
       -record user_home/var/temp/keyring_response_file.xml
     ```

3. When a window opens that requests your credentials for the authenticated remote repository, enter the correct credentials and **save** them.
4. Click **File > Exit** to close Installation Manager.
   For more information, read the IBM® Installation Manager Version 1.5 Information Center.

Installing the Eclipse Equinox OSGi framework with Eclipse Gemini for clients and servers

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, \
   gemini-blueprint-core-1.0.0.jar@1:start, \
   gemini-blueprint-extender-1.0.0.jar@1:start, \
   gemini-blueprint-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
  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.

Previous topic: OSGi framework overview
Next topic: Running eXtreme Scale containers with non-dynamic plug-ins in an OSGi environment
Related concepts:
OSGi framework overview
Related tasks:
Programming to use the OSGi framework
Updating OSGi services for eXtreme Scale plug-ins with xscmd

Related reference:
Server properties file
Related information:
Introduction: Starting and configuring the eXtreme Scale server and container to run plug-ins in the OSGi framework

Installing the REST data service

This topic describes how to install the WebSphere® eXtreme Scale REST data service into a Web server.

Before you begin

Software requirements
The WebSphere eXtreme Scale REST data service is a Java™ Web application that can be deployed to any application server that supports Java servlet specification, Version 2.3 and a Java runtime environment, Version 5 or later.

The following software is required:

- Java Standard Edition 5 or later
- Web servlet container, Version 2.3 or later, which includes one of the following:
  - WebSphere Application Server Version 6.1.0.25 or later
  - WebSphere Application Server Version 7.0.0.5 or later
  - WebSphere Community Edition Version 2.1.1.3 or later
  - Apache Tomcat Version 5.5 or later

  WebSphere eXtreme Scale, Version 7.1 or later, including the trial.

About this task

The WebSphere eXtreme Scale REST data service includes a single wxsrestservice.war file. The wxsrestservice.war file includes a single servlet that acts as a gateway between your WCF Data Services client applications or any other HTTP REST client and a data grid.

The REST data service includes a sample that allows you to quickly create a data grid and interact with it using an eXtreme Scale client or the REST data service. See Configuring REST data services for details on using the sample.

When WebSphere eXtreme Scale 7.1 is installed or the eXtreme Scale Version 7.1 trial is extracted, the following directories and files are included:

- restservice_home/lib
  The lib directory contains these files:
  - wxsrestservice.ear – The REST data service enterprise application archive for use with WebSphere Application Server and WebSphere Application Server CE.
  - wxsrestservice.war – The REST data service web module for use with Apache Tomcat.

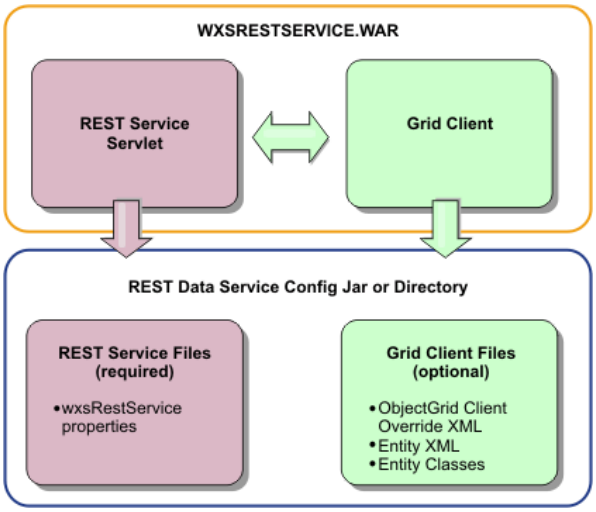
  The wxsrestservice.ear file includes the wxsrestservice.war file and are both tightly coupled with the WebSphere WebSphere eXtreme Scale runtime. If WebSphere eXtreme Scale is upgraded to a new version or a fix pack applied, the wxsrestservice.war file or wxsrestservice.ear file will need to be manually upgraded to the version installed in this directory.

- restservice_home/gettingstarted
  The gettingstarted directory contains a simple sample that demonstrates how to use the WebSphere eXtreme Scale REST data service with a data grid.

Procedure

Package and deploy the REST data service.
The REST data service is designed as a self-contained WAR module. To configure the REST data service, you must first package the REST data service configuration and optional WebSphere eXtreme Scale configuration files into a JAR file or directory. This application packaging is then referenced by the web container server runtime. The following diagram illustrates files used by the eXtreme Scale REST data service.

Figure 1. WebSphere eXtreme Scale REST Data Service Files



The REST service configuration JAR or directory must contain the following file:

wxsRestService.properties: The wxsRestService.properties file includes the configuration options for the REST data service. This includes the catalog service endpoints, ObjectGrid names to expose, trace options and more. See REST data service properties file.

The following ObjectGrid client files are optional:

- META-INF/objectGridClient.xml: The ObjectGrid client override XML file is used to connect to the remote data grid. By default this file is not required. If this file is not present, the REST service uses the server configuration, disabling the near cache.
  The name of the file can be overridden using the objectGridClientXML REST data service configuration property. If provided, this XML file should include:
  a. Any ObjectGrids that you want to expose to the REST data service.
  b. Any reference to the entity descriptor XML file associated with each ObjectGrid configuration.
- META-INF/*entity descriptor XML files*: One or more entity descriptor XML files are required only if the client needs to override the entity definition of the client. The entity descriptor XML file must be used in conjunction with the ObjectGrid client override XML descriptor file.
- **Entity classes** Annotated entity classes or an entity descriptor XML file can be used to describe the entity metadata. The REST service only requires entity classes in the classpath if the eXtreme Scale servers are configured with entity metadata classes and a client override entity XML descriptor is not used.
  An example with the minimum required configuration file, where the entities are defined in XML on the servers:

  ```
  restserviceconfig.jar:
  wxsRestService.properties
  ```

  The property file contains:

  ```
  catalogServiceEndPoints=localhost:2809
  objectGridNames=NorthwindGrid
  ```

  An example with one entity, override XML files and entity classes:

  ```
  restserviceconfig.jar:
  wxsRestService.properties
  ```

  The property file contains:

  ```
  catalogServiceEndPoints=localhost:2809
  objectGridNames=NorthwindGrid

  com/acme/entities/Customer.class
  META-INF/objectGridClient.xml
  ```

  The client ObjectGrid descriptor XML file contains:

  ```
  <objectGrid name="CustomerGrid" entityMetadataXMLFile="emd.xml"/>
  META-INF/emd.xml
  ```

  The entity metadata descriptor XML file contains:

  ```
  <entity class-name="com.acme.entities.Customer" name="Customer"/>
  ```

- Installing eXtreme Scale bundles
  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.

Installing eXtreme Scale bundles

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.
Related concepts:
Embedded server API

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.

Installing WebSphere eXtreme Scale or WebSphere eXtreme Scale Client with WebSphere Application Server

You can install WebSphere® eXtreme Scale or WebSphere eXtreme Scale Client in an environment in which WebSphere Application Server or WebSphere Application Server Network Deployment is installed. You can use the existing features of WebSphere Application Server or WebSphere Application Server Network Deployment to enhance your eXtreme Scale applications.

Before you begin

- Verify that the target installation directory does not contain an existing installation of WebSphere eXtreme Scale or WebSphere eXtreme Scale Client.
- Stop all processes that are running in your WebSphere Application Server or WebSphere Application Server Network Deployment environment. See Command-line utilities for more information about the **stopManager**, **stopNode**, and **stopServer** commands.
  CAUTION:
  Verify that any running processes are stopped. If the running processes are not stopped, the installation proceeds, creating unpredictable results. On some platforms, the installation might be left in an undetermined state.

Important: When you install WebSphere eXtreme Scale or WebSphere eXtreme Scale Client, it should be in the same directory in which you installed WebSphere Application Server. For example, if you installed WebSphere Application Server in C:*was_root*, then you should also choose C:*was_root* as the target directory for your WebSphere eXtreme Scale or WebSphere eXtreme Scale Client installation.

About this task

Integrate eXtreme Scale with WebSphere Application Server or WebSphere Application Server Network Deployment to apply the features of eXtreme Scale to your Java™ Platform, Enterprise Edition applications. Java EE applications host data grids and access the data grids using a client connection.

You can also run the grid as a client inside a Liberty profile that serves systems. To obtain the Liberty profile, you must install WebSphere Application Server .

Procedure

- If you want to install WebSphere eXtreme Scale or WebSphere eXtreme Scale Client within a WebSphere Application Server Version 8 environment, then complete the following steps:
  1. Install IBM Installation Manager. For more information, see Installing IBM Installation Manager using the GUI.
  2. Using Installation Manager, install the appropriate eXtreme Scale product offering:
     - WebSphere eXtreme Scale for WebSphere Application Server Version 8
     - WebSphere eXtreme Scale Client for WebSphere Application Server Version 8

     For more information, see Installing the product with the GUI.
  3. Download the necessary WebSphere Application Server Version 8 repositories from the Passport Advantage® site. For more information, see How to download WebSphere Application Server - Express V8.0 from Passport Advantage.
  4. Install WebSphere Application Server Version 8 For more information, see Installing the product using distributed operating systems using the GUI.
- If you want to install WebSphere eXtreme Scale or WebSphere eXtreme Scale Client within a WebSphere Application Server Version 7 environment, then complete the following steps:
  1. Install IBM Installation Manager. For more information, see Installing IBM Installation Manager using the GUI.
  2. Install WebSphere Application Server Version 7 using InstallShield MultiPlatform (ISMP) installer. For more information, see Installing your application serving environment.
  3. After installation, you must import WebSphere Application Server Version 7 into Installation Manager to complete the installation. Importing WebSphere Application Server Version 7 into the Installation Manager allows you to manage and install fix packs for the product from one location. You must ensure you have the necessary repositories set up within Installation Manager for access to fix packs and updates. For more information on how to import an existing installation of WebSphere Application Server 7 into Installation Manager, see Importing IBM WebSphere Application Server product information into the IBM Installation Manager registry.
  4. Using Installation Manager, install the appropriate eXtreme Scale product offering:
     - WebSphere eXtreme Scale for WebSphere Application Server Version 7
     - WebSphere eXtreme Scale Client for WebSphere Application Server Version 7

     For more information, see Installing the product with the GUI.
- If you want to install WebSphere eXtreme Scale or WebSphere eXtreme Scale Client within a WebSphere Application Server Version 6 environment, then complete the following steps:
  1. Install IBM Installation Manager. For more information, see Installing IBM Installation Manager using the GUI.
  2. Using Installation Manager, install the appropriate eXtreme Scale product offering:
     - WebSphere eXtreme Scale for WebSphere Application Server Version 6.1
     - WebSphere eXtreme Scale Client for WebSphere Application Server Version 6.1

     For more information, see Installing the product with the GUI.
  3. Install WebSphere Application Server Version 6 using the ISMP installer. For more information, see Installing your application serving environment

What to do next

- Start configuring your WebSphere eXtreme Scale or WebSphere eXtreme Scale Client installation. For more information, see Taking the first steps after installation.

8.5+

Installing the Liberty profile

8.5+ You install the Liberty profile application-serving environment by using the Installation Manager, or by running a Java™ archive (JAR) file.

About this task

To install the Liberty profile, you can use IBM® Installation Manager to install WebSphere® Application Server Version 8.5 or higher with WebSphere eXtreme Scale, or you can install the Liberty profile by running a provided JAR file. If you install the WebSphere Application ServerLiberty profile using Installation Manager, then you must also install WebSphere eXtreme Scale Version 8.5 or higher using Installation Manager. Likewise, if you install the WebSphere Application ServerLiberty profile by running a JAR file, then you must also install the WebSphere eXtreme Scale Version 8.5 or higher product image by running a JAR file.
On z/OS® operating systems, the Liberty profile provides an operations environment. You can work natively with this environment using the MVS™ console. For application development, consider using the Eclipse-based developer tools on a separate distributed system, on Mac OS, or in a Linux shell on z/OS.

Procedure

- Install the WebSphere Application Server Liberty profile:
  - Install the Liberty profile by using Installation Manager. For more information about installing WebSphere Application Server Version 8.5 or higher with WebSphere eXtreme Scale, see IBM Installation Manager and WebSphere eXtreme Scale product offerings.
  - Install the Liberty profile application-serving environment by running a Java archive (JAR) file.
- Install WebSphere eXtreme Scale Version 8.5 or higher. For more information, see IBM Installation Manager and WebSphere eXtreme Scale product offerings.
- Migrate your Liberty profile environment. When you migrate from one major release of the Liberty profile to a higher major release, you must change the feature version numbers in your Liberty profile server.xml file.
  For example, in the initial version of the Liberty profile that the product supported, feature version numbers were at the `1.0` level. In WebSphere eXtreme Scale Version 8.6 and higher, feature version numbers are at the `1.1` level.

  Note: When you upgrade to WebSphere Application Server V8.5.5, the Liberty profile is removed. WebSphere eXtreme Scale supports the removal of the feature. However, if you roll back to WebSphere Application Server V8.5, the Liberty profile is added again, which causes problems in WebSphere eXtreme Scale.

- Installing the Liberty profile application-serving environment by running a JAR file
  By running the Java archive (JAR) file that contains the distribution image, you install the application-serving environment, and you are ready to create a Liberty server.

Related concepts:
Liberty profile
Related reference:
Server properties file

Installing the Liberty profile application-serving environment by running a JAR file

By running the Java archive (JAR) file that contains the distribution image, you install the application-serving environment, and you are ready to create a Liberty server.

About this task

You can install the Liberty profile application-serving environment by running a JAR file as described in this topic, or by using the Installation Manager.

When you run the JAR file for WebSphere® Application Server to install the Liberty profile, you must extract the JAR file first. Then, extract the Liberty profile JAR file for WebSphere eXtreme Scale. If you use IBM Installation Manager to install WebSphere Application Server Version 8.5 and obtain the Liberty profile, then you must use Installation Manager to also install WebSphere eXtreme Scale.

This task supports the following editions:

- WebSphere Application Server Liberty Core
- WebSphere Application Server, Base and Developer editions
- WebSphere Application Server, Network Deployment
- WebSphere Application Server for z/OS

For download information for the Liberty profile application-serving and data caching environments, see the WASdev community downloads page.

Procedure

1. Extract the WebSphere Application Server Liberty profile distribution image to your preferred directory.
   This image is packaged as a JAR file; for example, `wlp-edition-8.5.0.0.jar`. Use one of the following actions to extract this JAR file:
   - To extract the distribution image by using the wizard, run `java -jar wlp-edition-8.5.0.0.jar`.
   - To extract the distribution image by accepting the license terms and conditions silently, run `java -jar wlp-edition-8.5.0.0.jar -acceptLicense`.
   - To view all available options, run `java -jar wlp-edition-8.5.0.0.jar -help`.
   - After you run the command to extract the JAR file, follow the steps that are documented through the command-line utility to complete the installation.

   All the application server files are now stored in subdirectories of the wlp directory.

2. Optional: Set the `JAVA_HOME` property for your environment.
   The Liberty profile requires a JRE in which to run. It does not share the JDK or JRE that the WebSphere Application Server full profile uses. You can specify the JDK or JRE location using the JAVA_HOME property in the server.env file, as described in Customizing the Liberty profile environment. On Linux or UNIX systems, you can instead set JAVA_HOME in the user .bashrc file, or append the JDK or JRE path to the PATH environment variable. On Windows systems, you can instead set JAVA_HOME as a system environment variable, or append the JDK or JRE path to the PATH system variable.

   Windows For example, on Windows systems you can use the following commands to set the JAVA_HOME property, and to add the Java™ /bin directory to the path:

   ```
   set JAVA_HOME=C:\Progra~1\Java\JDK16
   set PATH=%JAVA_HOME%\bin;%PATH%
   ```

   Notes:
   - The Liberty profile runtime environment searches for the **java** command in this order: JAVA_HOME property, JRE_HOME property, and system PATH property.
   - For more information about supported Java environments, and where to get them, see Minimum supported Java levels in the Liberty profile: Runtime environment known restrictions.

3. Download WebSphere eXtreme Scale for Developers -Liberty profile. Extract the WebSphere eXtreme Scale distribution image to the directory where you extracted the `wlp-edition-8.5.0.0.jar` file.
   This image is packaged as a JAR file called `wxs-wlp_8.5.0.1.jar`. To extract the distribution image, run the JAR file; for example, run the following command, depending on your version of eXtreme Scale: 8.5

   ```
   java -jar wxs-wlp_8.5.0.1.jar -acceptLicense
   ```

   ```
   java -jar wxs-wlp_8.6.0.4.jar
   ```

Results

If you extract the `wxs-wlp_8.5.0.1.jar` file , eXtreme Scale is installed on top of the WebSphere Application Server Liberty profile when you extract both JAR files to the same directory.

Installing fix packs using IBM Installation Manager

You can use IBM® Installation Manager to update the product with the fix packs that are available for WebSphere® eXtreme Scale product offerings. Fix packs can be installed from the GUI, the command line, or using response files.

- Installing fix packs using the GUI
  You can update this product to a later version using the IBM Installation Manager wizard.
- 8.5+ Installing fix packs using the command line
  You can use the IBM Installation Manager from the command line to update the product with the fix packs that are available for WebSphere eXtreme Scale product offerings.
- Installing fix packs using a response file
  You can update this product to a later version using IBM Installation Manager with a response file.

Related concepts:
Configuration considerations for multi-master topologies
Related tasks:
Uninstalling fix packs using the GUI
Installing fix packs using the GUI
Retrieving eXtreme Scale environment information with the xscmd utility
Updating eXtreme Scale servers
Configuring multiple data center topologies
Migrating to WebSphere eXtreme ScaleVersion 8.5
Starting and stopping stand-alone servers
Starting and stopping servers in a WebSphere Application Server environment

Installing fix packs using the GUI

You can update this product to a later version using the IBM® Installation Manager wizard.

Before you begin

Contact the IBM Software Support Center for information about upgrades for WebSphere® eXtreme Scale stand-alone or WebSphere eXtreme Scale for WebSphere Application Server product offerings. The most current information is available from the IBM Software Support Center and Fix Central.

IBM Installation Manager is used to apply product maintenance to the following product offerings:

- WebSphere eXtreme Scale in a stand-alone environment
- WebSphere eXtreme Scale Client in a stand-alone environment
- WebSphere eXtreme Scale for WebSphere Application Server Version 6.1
- WebSphere eXtreme Scale for WebSphere Application Server Version 7
- WebSphere eXtreme Scale for WebSphere Application Server Version 8
- WebSphere eXtreme Scale Client forWebSphere Application Server Version 6.1
- WebSphere eXtreme Scale Client for WebSphere Application Server Version 7
- WebSphere eXtreme Scale Client for WebSphere Application Server Version 8

Make sure that the web-based or local service repository location is listed and checked or that the **Search service repositories during installation and updates** option is selected on the Repositories panel in your Installation Manager preferences. For more information on using service repositories with Installation Manager, read the IBM Installation Manager Version 1.5 Information Center.

About this task

Restriction: You cannot use the Installation Manager to upgrade an installation and add or remove the full WebSphere Application Server profile feature or the Liberty profile feature.

Procedure

1. Stop all processes that are running in your environment.
   - To stop all processes that are running in your stand-alone eXtreme Scale environment, see Stopping stand-alone servers.
   - To stop all processes that are running in your WebSphere Application Server environment, see Command-line utilities.
2. Log on to your system.
3. Stop all servers and applications on the WebSphere Application Server installation that is being updated.
4. Start Installation Manager.
5. Click Update.
   Note: If you are prompted to authenticate, use the IBM ID and password that you use to access protected IBM software websites.
6. Select the package group to update.
   Tip: If you select **Update all**, Installation Manager will search all of the added and predefined repositories for updates to all of the package groups that it has installed. Use this feature only if you have full control over which fixes are contained in the targeted repositories. If you create and point to a set of custom repositories that include only the specific fixes that you want to install, you should be able to use this feature confidently. If you enable searching service repositories or install fixes directly from other live web-based repositories, then you might not want to select this option so that you can select only the fixes that you want to install for each offering on subsequent panels.
7. Click Next.
8. Select the version to which you want to update under:
   - WebSphere eXtreme Scale in a stand-alone environment
   - WebSphere eXtreme Scale Client in a stand-alone environment
   - WebSphere eXtreme Scale for WebSphere Application Server Version 6
   - WebSphere eXtreme Scale for WebSphere Application Server Version 7
   - WebSphere eXtreme Scale for WebSphere Application Server Version 8
   - WebSphere eXtreme Scale Client forWebSphere Application Server Version 6
   - WebSphere eXtreme Scale Client for WebSphere Application Server Version 7
   - WebSphere eXtreme Scale Client for WebSphere Application Server Version 8
9. Select any fixes that you want to install.
   Any recommended fixes are selected by default.

   If there are recommended fixes, you can select the option to show only recommended fixes and hide non-recommended fixes.

10. Click Next.
11. Accept the terms in the license agreements, and click Next.
12. Select the optional features that you want in your updated installation.
13. Review the summary information, and click Update.
    - If the installation is successful, the program displays a message indicating that installation is successful.
    - If the installation is not successful, click View Log File to troubleshoot the problem.
14. Click Finish.
15. Click File > Exit to close Installation Manager.

Related concepts:
Installation overview
IBM Installation Manager and WebSphere eXtreme Scale product offerings
Installing fix packs using IBM Installation Manager
Related tasks:
Uninstalling fix packs using the GUI

Installing fix packs using the command line

You can use the IBM® Installation Manager from the command line to update the product with the fix packs that are available for WebSphere® eXtreme Scale product offerings.

Before you begin

. Contact the IBM Software Support Center for information about upgrades for WebSphere eXtreme Scale stand-alone or WebSphere eXtreme Scale for WebSphere Application Server product offerings. The most current information is available from the IBM Software Support Center and Fix Central.

IBM Installation Manager is used to apply product maintenance to the following product offerings:

- WebSphere eXtreme Scale in a stand-alone environment
- WebSphere eXtreme Scale Client in a stand-alone environment
- WebSphere eXtreme Scale for WebSphere Application Server Version 6.1
- WebSphere eXtreme Scale for WebSphere Application Server Version 7
- WebSphere eXtreme Scale for WebSphere Application Server Version 8
- WebSphere eXtreme Scale Client forWebSphere Application Server Version 6.1
- WebSphere eXtreme Scale Client for WebSphere Application Server Version 7
- WebSphere eXtreme Scale Client for WebSphere Application Server Version 8

About this task

Restriction: You cannot use the Installation Manager to upgrade an installation and add or remove the full WebSphere Application Server profile feature or the Liberty profile feature.

Procedure

1. For a list of interim fixes and fix packs that are available forWebSphere eXtreme Scale 8.5 and specific information about each fix, perform the following actions.
    a. Go to Fix Central.
    b. Select **WebSphere** as the product group.
    c. Select WebSphere eXtreme Scale as the product.
    d. Select **8.5** as the installed version.
    e. Select your operating system as the platform, and click **Continue**.
    f. Select **Browse for fixes**, and click **Continue**.
    g. Click **More Information** under each fix to view information about the fix.
    h. **Recommendation:** Make a note of the name of the fix pack that you would like to install.
2. Update WebSphere eXtreme Scale Version 8.5 with the fix pack using the following procedure.
    - Download the file that contains the fix pack from Fix Central, and use local updating.
      You can download a compressed file that contains the fix pack from Fix Central. Each compressed fix-pack file contains an Installation Manager repository for the fix pack and usually has a .zip extension. After downloading and extracting the fix-pack file, use Installation Manager to update WebSphere Application Server Version 8.x with the fix pack.
        - To download the fix pack, perform the following actions:
            - Go to Fix Central.
            - Select **WebSphere** as the product group.
            - Select **WebSphere eXtreme Scale** as the product.
            - Select **8.x** as the installed version.
            - Select your operating system as the platform, and click **Continue**.
            - Select **Browse for fixes**, and click **Continue**.
            - Select the fix pack that you want to download, and click **Continue**.
            - Select your download options, and click **Continue**.
            - Click **I agree** to agree to the terms and conditions.
            - Click **Download now** to download the fix pack.
            - Transfer the compressed file in binary format to the system on which it will be installed.
            - Extract the compressed repository files to a directory on your system.
        - To install a fix pack from a downloaded file, perform the following actions:
            - Log on to your system.
            - Stop all processes that are running in your environment. To stop all processes that are running in your WebSphere Application Server environment, see Command-line utilities.
            - Change to the *Installation_Manager_binaries*/eclipse/tools directory, where *Installation_Manager_binaries* is the installation root directory for the Installation Manager.
            - Install the fix pack.

              UNIX Linux

              ```
              ./imcl install offering_ID_offering_version,optional_feature_ID
                -installationDirectory product_installation_location
                -repositories location_of_expanded_files
                -acceptLicense
              ```

              Windows

              ```
              imcl.exe install offering_ID_offering_version,optional_feature_ID
                -installationDirectory product_installation_location
                -repositories location_of_expanded_files
                -acceptLicense
              ```

              Tips:
                - The *offering_ID* is the offering ID that is listed in Offering IDs for WebSphere eXtreme Scale product offerings.

- The *offering_version*, which optionally can be attached to the offering ID with an underscore, is a specific version of the offering to install (`8.5.0.20110503_0200` for example).
  - If *offering_version* is **not** specified, the latest version of the offering and **all** interim fixes for that version are installed.
  - If *offering_version* is specified, the specified version of the offering and **no** interim fixes for that version are installed.

  The offering version can be found attached to the end of the offering ID with an underscore when you run the following command against the repository:

  ```
  imcl listAvailablePackages -repositories source_repository
  ```

- You can also specify `none`, `recommended` or `all` with the `-installFixes` argument to indicate which interim fixes you want installed with the offering.
  - If the offering version is **not** specified, the `-installFixes` option defaults to `all`.
  - If the offering version is specified, the `-installFixes` option defaults to `none`.
- You can add a list of features that are separated by commas. If a list of features is not specified, the default features are installed.

- **Optional:** List all installed packages to verify the installation:

  UNIX Linux

  ```
  ./imcl listInstalledPackages -long
  ```

  Windows

  ```
  imcl.exe listInstalledPackages -long
  ```

Related concepts:
Installation overview
IBM Installation Manager and WebSphere eXtreme Scale product offerings
Related tasks:
Uninstalling fix packs using the command line

Installing fix packs using a response file

You can update this product to a later version using IBM® Installation Manager with a response file.

Before you begin

Tip: As an alterative to the procedure that is described in this article, Installation Manager allows you to use the **updateAll** command in a response file or on the command line to search for and update all installed packages. Use this command only if you have full control over which fixes are contained in the targeted repositories. If you create and point to a set of custom repositories that include only the specific fixes that you want to install, you should be able to use this command confidently. If you enable searching service repositories or install fixes directly from other live web-based repositories, then you might not want to select this option so that you can select only the fixes that you want to install using the -installFixes option with the **install** command on the command line or the installFixes attribute in a response file.

Procedure

1. For a list of interim fixes and fix packs that are available for WebSphere® eXtreme Scale and specific information about each fix, perform the following actions.
   a. Go to Fix Central.
   b. Select **WebSphere** as the product group.
   c. Select WebSphere eXtreme Scale as the product.
   d. Select **8.x** as the installed version.
   e. Select your operating system as the platform, and click **Continue**.
   f. Select **Browse for fixes**, and click **Continue**.
   g. Click **More Information** under each fix to view information about the fix.
   h. **Recommendation:** Make a note of the name of the fix pack that you would like to install.
2. Update WebSphere eXtreme Scale with the fix pack using the following procedure.
   - Download the file that contains the fix pack from Fix Central, and use local updating.
     You can download a compressed file that contains the fix pack from Fix Central. Each compressed fix-pack file contains an Installation Manager repository for the fix pack and usually has a .zip extension. After downloading and extracting the fix-pack file, use Installation Manager to update WebSphere eXtreme Scale with the fix pack.
     - To download the fix pack, perform the following actions:
       - Go to Fix Central.
       - Select **WebSphere** as the product group.
       - Select **WebSphere eXtreme Scale** as the product.
       - Select **8.6** as the installed version.
       - Select your operating system as the platform, and click **Continue**.
       - Select **Browse for fixes**, and click **Continue**.
       - Select the fix pack that you want to download, and click **Continue**.
       - Select your download options, and click **Continue**.
       - Click **I agree** to agree to the terms and conditions.
       - Click **Download now** to download the fix pack.
       - Transfer the compressed file in binary format to the system on which it will be installed.
       - Extract the compressed repository files to a directory on your system.
     - Perform the following actions:

- Log on to your system.
- If the repository requires a user name and password, create a keyring file to access this repository.
  For more information on creating a keyring file for Installation Manager, read the IBM Installation Manager Version 1.5 Information Center.

  Tip: When creating a keyring file, append /repository.config at the end of the repository URL location if the **imutilsc** command is unable to find the URL that is specified.
- To stop all processes that are running in your stand-alone eXtreme Scale environment, see Stopping stand-alone serversTo stop all processes that are running in your WebSphere Application Server environment, see Command-line utilities.
- Change to the *Installation_Manager_binaries*/eclipse/tools directory, where *Installation_Manager_binaries* is the installation root directory for the Installation Manager.
- Install the fix pack using a response file.
  For example:
  - Windows **Administrator or non-administrator:**

    ```
    imcl.exe -acceptLicense
      input C:\temp\update_response_file.xml
      -log C:\temp\update_log.xml
      -keyring C:\IM\im.keyring
    ```

  - UNIX Linux **Administrator:**

    ```
    ./imcl -acceptLicense
      input /var/temp/update_response_file.xml
      -log /var/temp/update_log.xml
      -keyring /var/IM/im.keyring
    ```

  - UNIX Linux **Non-administrator:**

    ```
    ./imcl -acceptLicense
      input user_home/var/temp/update_response_file.xml
      -log user_home/var/temp/update_log.xml
      -keyring user_home/var/IM/im.keyring
    ```

Related concepts:
Installation overview
IBM Installation Manager and WebSphere eXtreme Scale product offerings
Related tasks:
Uninstalling fix packs using response files

Uninstalling fix packs using IBM Installation Manager

You can use IBM® Installation Manager to rollback WebSphere® eXtreme Scale product offerings to an earlier version. You can uninstall fix packs from the GUI, the command line, or using response files.

- 8.5+ Uninstalling fix packs using the GUI
  You can roll back this product to an earlier version using the IBM Installation Manager GUI.
- 8.5+ Uninstalling fix packs using the command line
  You can roll back this product to an earlier version using IBM Installation Manager from the command line.
- 8.5+ Uninstalling fix packs using response files
  You can roll back this product to an earlier version using IBM Installation Manager with a response file.

Uninstalling fix packs using the GUI

You can roll back this product to an earlier version using the IBM® Installation Manager GUI.

Before you begin

During the rollback process, Installation Manager must access files from the earlier version of the package. By default, these files are stored on your computer when you install a package. If you change the default setting or delete the saved files, Installation Manager requires access to the repository that was used to install the earlier version.

About this task

Restriction: You cannot use the Installation Manager to roll back an installation and add or remove a feature.

Procedure

1. Stop all processes that are running in your environment.
   - To stop all processes that are running in your stand-alone eXtreme Scale environment, see Stopping stand-alone servers.
   - To stop all processes that are running in your WebSphere® Application Server environment, see Command-line utilities.
2. Start Installation Manager.

3. Click Roll Back.
4. Select the package group to roll back.
5. Click Next.
6. Select the version to which you want to roll back under.
7. Click Next.
8. Review the summary information, and click Roll Back.
   - If the rollback is successful, the program displays a message indicating that the rollback is successful.
   - If the rollback is not successful, click View Log File to troubleshoot the problem.
9. Click Finish.
10. Click File > Exit to close Installation Manager.

Related concepts:
Installation overview
IBM Installation Manager and WebSphere eXtreme Scale product offerings
Installing fix packs using IBM Installation Manager
Related tasks:
Installing fix packs using the GUI

Uninstalling fix packs using the command line

You can roll back this product to an earlier version using IBM® Installation Manager from the command line.

Before you begin

Restriction: In order to use this procedure, you must have Installation Manager Version 1.5 or later installed on your system.
During the rollback process, Installation Manager must access files from the earlier version of the package. By default, these files are stored on your computer when you install a package. If you change the default setting or delete the saved files, Installation Manager requires access to the repository that was used to install the earlier version.

About this task

Restriction: You cannot use the Installation Manager to roll back an installation and add or remove the full WebSphere® Application Server profile feature or the Liberty profile feature.

Procedure

1. Optional: If the repository requires a user name and password, create a keyring file to access this repository.
   For more information on creating a keyring file for Installation Manager, read the IBM Installation Manager Version 1.5 Information Center.

   Tip: When creating a keyring file, append /repository.config at the end of the repository URL location if the **imutilsc** command is unable to find the URL that is specified.
2. Log on to your system.
3. Stop all processes that are running in your environment.
   - To stop all processes that are running in your stand-alone eXtreme Scale environment, see Stopping stand-alone servers.
   - To stop all processes that are running in your WebSphere Application Server environment, see Command-line utilities.
4. Change to the eclipse/tools subdirectory in the directory where you installed Installation Manager.
5. Use the **imcl** command to roll back the product.

   UNIX Linux

   ```
   ./imcl rollback offering_ID_offering_version
     -repositories source_repository
     -installationDirectory installation_directory
     -preferences preference_key=value
     -properties property_key=value
     -keyring keyring_file -password password
     -acceptLicense
   ```

   Windows

   ```
   imcl.exe rollback offering_ID_offering_version
     -repositories source_repository
     -installationDirectory installation_directory
     -preferences preference_key=value
     -properties property_key=value
     -keyring keyring_file -password password
     -acceptLicense
   ```

   Tips:
   - The *offering_ID* is the offering ID that is listed in Offering IDs for WebSphere eXtreme Scale product offerings.
   - The *offering_version*, which optionally can be attached to the offering ID with an underscore, is a specific version of the offering to which to roll back (`8.5.0.20110503_0200` for example).
     - If *offering_version* is **not** specified, the installation rolls back to the previously installed version of the offering and **all** interim fixes for that version are installed.
     - If *offering_version* is specified, the installation rolls back to the specified earlier version of the offering and **no** interim fixes for that version are installed.

The offering version can be found attached to the end of the offering ID with an underscore in the Package section of the report that is generated when you run the **historyInfo** or **genHistoryReport** command from the *app_server_root*/bin directory.

For more information on using Installation Manager, read the IBM Installation Manager Version 1.5 Information Center.

6. Optional: List all installed packages to verify the roll back.
   UNIX Linux

   ```
   ./imcl listInstalledPackages -long
   ```

   Windows

   ```
   imcl.exe listInstalledPackages -long
   ```

Related concepts:
Installation overview
IBM Installation Manager and WebSphere eXtreme Scale product offerings
Related tasks:
Installing fix packs using the command line

Uninstalling fix packs using response files

You can roll back this product to an earlier version using IBM® Installation Manager with a response file.

Before you begin

During the rollback process, Installation Manager must access files from the earlier version of the package. By default, these files are stored on your computer when you install a package. If you change the default setting or delete the saved files, Installation Manager requires access to the repository that was used to install the earlier version.

About this task

Restriction: You cannot use the Installation Manager to roll back an installation and add or remove the full WebSphere® Application Server profile feature or the Liberty profile feature.

Procedure

1. Optional: If the repository requires a username and password, create a keyring file to access this repository.
   For more information on creating a keyring file for Installation Manager, read the IBM Installation Manager Version 1.5 Information Center.

   Tip: When creating a keyring file, append /repository.config at the end of the repository URL location if the **imutilsc** command is unable to find the URL that is specified.
2. Log on to your system.
3. Stop all processes that are running in your environment.
   - To stop all processes that are running in your stand-alone eXtreme Scale environment, see Stopping stand-alone servers.
   - To stop all processes that are running in your WebSphere Application Server environment, see Command-line utilities.
4. Use a response file to roll back the product.
   Change to the eclipse/tools subdirectory in the directory where you installed Installation Manager, and roll back the product.

   For example:
   - Windows **Administrator or non-administrator:**

     ```
     imcl.exe
       input C:\temp\rollback_response_file.xml
       -log C:\temp\rollback_log.xml
       -keyring C:\IM\im.keyring
     ```

   - UNIX Linux **Administrator:**

     ```
     ./imcl
       input /var/temp/rollback_response_file.xml
       -log /var/temp/rollback_log.xml
       -keyring /var/IM/im.keyring
     ```

   - UNIX Linux **Non-administrator:**

     ```
     ./imcl
       input user_home/var/temp/rollback_response_file.xml
       -log user_home/var/temp/rollback_log.xml
       -keyring user_home/var/IM/im.keyring
     ```

   Note: The program might write important post-installation instructions to standard output.
   For more information on using Installation Manager, read the IBM Installation Manager Version 1.5 Information Center.
5. Optional: List all installed packages to verify the roll back.
   UNIX Linux

   ```
   ./imcl listInstalledPackages -long
   ```

Windows

```
imcl.exe listInstalledPackages -long
```

Related concepts:
Installation overview
IBM Installation Manager and WebSphere eXtreme Scale product offerings
Related tasks:
Installing fix packs using a response file

Uninstalling the product using IBM Installation Manager

Use IBM® Installation Manager to uninstall WebSphere® eXtreme Scale product offerings.

- 8.5+ Uninstalling the product using the GUI
  You can use IBM Installation Manager GUI to uninstall the product .
- 8.5+ Uninstalling the product using the command line
  You can uninstall the product using IBM Installation Manager from the command line.
- 8.5+ Uninstalling the product using response files
  You can uninstall the product using IBM Installation Manager with response files.

Uninstalling the product using the GUI

You can use IBM® Installation Manager GUI to uninstall the product .

Before you begin

You must remove the WebSphere® eXtreme Scale augment from all WebSphere Application Server profiles before uninstalling WebSphere eXtreme Scale. You will be unable to perform the unaugment after uninstalling WebSphere eXtreme Scale. Use the manageprofiles command to unaugment existing profiles in a WebSphere eXtreme Scale environment. .

Procedure

1. Uninstall the product.
   a. Stop all processes that are running in your environment.
      - To stop all processes that are running in your stand-alone eXtreme Scale environment, see Stopping stand-alone servers.
      - To stop all processes that are running in your WebSphere Application Server environment, see Command-line utilities.
   b. Start Installation Manager.
   c. Click **Uninstall**.
   d. In the **Uninstall Packages** window, perform the following actions.
      i. Select one of the following and the appropriate version:
         - WebSphere eXtreme Scale in a stand-alone environment
         - WebSphere eXtreme Scale Client in a stand-alone environment
         - WebSphere eXtreme Scale for WebSphere Application Server Version 6
         - WebSphere eXtreme Scale for WebSphere Application Server Version 7
         - WebSphere eXtreme Scale for WebSphere Application Server Version 8
         - WebSphere eXtreme Scale Client forWebSphere Application Server Version 6
         - WebSphere eXtreme Scale Client for WebSphere Application Server Version 7
         - WebSphere eXtreme Scale Client for WebSphere Application Server Version 8
      ii. Click **Next**.
   e. If the uninstallation wizard displays a list of augmented WebSphere Application Server profiles, then you must unaugment these profiles in order to proceed with the uninstallation.
   f. Review the summary information.
   g. Click **Uninstall**.
      - If the uninstallation is successful, the program displays a message that indicates success.
      - If the uninstallation is not successful, click **View log** to troubleshoot the problem.
   h. Click **Finish**.
   i. Click **File > Exit** to close Installation Manager.
2. Optional: Uninstall IBM Installation Manager.
   Important: Before you can uninstall IBM Installation Manager, you must uninstall all of the packages that were installed by Installation Manager.
   Read the IBM Installation Manager Version 1.5 Information Center for information about performing this procedure.

Related concepts:
Installation overview
IBM Installation Manager and WebSphere eXtreme Scale product offerings
Related tasks:
Installing IBM Installation Manager using the GUI
Installing the product with the GUI

Uninstalling the product using the command line

You can uninstall the product using IBM® Installation Manager from the command line.

Before you begin

You must remove the WebSphere® eXtreme Scale augment from all WebSphere Application Server profiles before uninstalling WebSphere eXtreme Scale. You will be unable to perform the unaugment after uninstalling WebSphere eXtreme Scale. Use the manageprofiles command to unaugment existing profiles in a WebSphere eXtreme Scale environment.

Procedure

1. Log on to your system.
2. Stop all processes that are running in your environment.
   - To stop all processes that are running in your stand-alone eXtreme Scale environment, see Stopping stand-alone servers.
   - To stop all processes that are running in your WebSphere Application Server environment, see Command-line utilities.
   - To stop Liberty servers that are running in your WebSphere Application Server, see the Liberty profile server command.
3. Change to the eclipse/tools subdirectory in the directory where you installed Installation Manager.
4. Use the **imcl** command to uninstall the product.

   Windows

   ```
   imcl.exe uninstall com.ibm.websphere.v85,optional_feature_ID
     -installationDirectory installation_directory
   ```

   UNIX Linux

   ```
   ./imcl uninstall com.ibm.websphere.v85,optional_feature_ID
     -installationDirectory installation_directory
   ```

   Tips:
   - The *offering_ID* is the offering ID that is listed in Offering IDs for WebSphere eXtreme Scale product offerings.
   - You can remove a list of features that are separated by commas—the feature ID. For example,

     ```
     imcl uninstall com.ibm.websphere.WXS.v85,xs.console.feature,xs.samples.feature
     ```

     - `client` indicates the stand-alone client feature
     - `server` indicates the stand-alone server feature
     - `console` indicates the web based monitoring console
     - `samples` indicates the samples
   - If a list of features is not specified, the entire product is uninstalled.

   Go to the IBM Installation Manager Version 1.5 Information Center for more information.
5. If the uninstallation process displays a list of augmented WebSphere Application Server profiles, then you must unaugment these profiles in order to proceed with the uninstallation.
6. Optional: Uninstall IBM Installation Manager.
   Important: Before you can uninstall IBM Installation Manager, you must uninstall all of the packages that were installed by Installation Manager. Read the IBM Installation Manager Version 1.5 Information Center for information about using the uninstall script to perform this procedure.

Related concepts:
Installation overview
IBM Installation Manager and WebSphere eXtreme Scale product offerings
Related tasks:
Installing IBM Installation Manager using the command line
Installing the product using the command line

Uninstalling the product using response files

You can uninstall the product using IBM® Installation Manager with response files.

Before you begin

You must remove the WebSphere® eXtreme Scale augment from all WebSphere Application Server profiles before uninstalling WebSphere eXtreme Scale. You will be unable to perform the unaugment after uninstalling WebSphere eXtreme Scale. Use the manageprofiles command to unaugment existing profiles in a WebSphere eXtreme Scale environment.
Optional: Perform or record the installation of Installation Manager and installation of the product to a temporary installation registry on one of your systems so that you can use this temporary registry to record the uninstallation without using the standard registry where Installation Manager is installed.

About this task

Using Installation Manager, you can work with response files to uninstall the product in a variety of ways. You can record a response file using the GUI as described in the following procedure, or you can generate a new response file by hand or by taking an example and modifying it.

Procedure

1. Stop all processes that are running in your environment.
   - To stop all processes that are running in your stand-alone eXtreme Scale environment, see Stopping stand-alone servers.
   - To stop all processes that are running in your WebSphere Application Server environment, see Command-line utilities.
   - To stop Liberty servers that are running in your WebSphere Application Server, see the Liberty profile server command.
2. Optional: **Record a response file to uninstall the product:** On one of your systems, perform the following actions to record a response file that will uninstall the product:
   a. From a command line, change to the eclipse subdirectory in the directory where you installed Installation Manager.
   b. Start Installation Manager from the command line using the -record option.
   For example:
   - Windows **Administrator or non-administrator:**

     ```
     IBMIM.exe -skipInstall "C:\temp\imRegistry"
       -record C:\temp\uninstall_response_file.xml
     ```

   - UNIX Linux **Administrator:**

     ```
     ./IBMIM -skipInstall /var/temp/imRegistry
       -record /var/temp/uninstall_response_file.xml
     ```

   - UNIX Linux **Non-administrator:**

     ```
     ./IBMIM -skipInstall user_home/var/temp/imRegistry
       -record user_home/var/temp/uninstall_response_file.xml
     ```

   Tip: If you choose to use the -skipInstall parameter with a temporary installation registry created as described in *Before you begin*, Installation Manager uses the temporary installation registry while recording the response file. It is important to note that when the -skipInstall parameter is specified, no product packages are installed or uninstalled. All of the actions that you perform in Installation Manager simply update the installation data that is stored in the specified temporary registry. After the response file is generated, it can be used to uninstall the product, removing the product files and updating the standard installation registry.
   The -skipInstall operation should not be used on the actual agent data location used by Installation Manager. This is unsupported. Use a clean writable location, and re-use that location for future recording sessions.

   For more information, read the IBM Installation Manager Version 1.5 Information Center.

   c. Click Uninstall.
   d. In the Uninstall Packages window, perform the following actions.
      i. Select one of the following and the appropriate version:
         - WebSphere eXtreme Scale in a stand-alone environment
         - WebSphere eXtreme Scale Client in a stand-alone environment
         - 8.5 WebSphere eXtreme Scale for WebSphere Application Server Version 6
         - WebSphere eXtreme Scale for WebSphere Application Server Version 7
         - WebSphere eXtreme Scale for WebSphere Application Server Version 8
         - 8.5 WebSphere eXtreme Scale Client forWebSphere Application Server Version 6
         - WebSphere eXtreme Scale Client for WebSphere Application Server Version 7
         - WebSphere eXtreme Scale Client for WebSphere Application Server Version 8

      ii. Click **Next**.
      iii. Click Next.
   e. Review the summary information.
   f. Click Uninstall.
      - If the uninstallation is successful, the program displays a message that indicates success.
      - If the uninstallation is not successful, click **View log** to troubleshoot the problem.
   g. Click Finish.
   h. Click File > Exit to close Installation Manager.
3. **Use the response file to uninstall the product:** From a command line on each of the systems from which you want to uninstall the product, change to the eclipse/tools subdirectory in the directory where you installed Installation Manager and use the response file that you created to uninstall the product.
   For example:
   - Windows **Administrator or non-administrator:**

     ```
     imcl.exe
       input C:\temp\uninstall_response_file.xml
       -log C:\temp\uninstall_log.xml
     ```

   - UNIX Linux **Administrator:**

     ```
     ./imcl
       input /var/temp/uninstall_response_file.xml
       -log /var/temp/uninstall_log.xml
     ```

   - UNIX Linux **Non-administrator:**

     ```
     ./imcl
       input user_home/var/temp/uninstall_response_file.xml
       -log user_home/var/temp/uninstall_log.xml
     ```

   Go to the IBM Installation Manager Version 1.5 Information Center for more information.

4. Optional: List all installed packages to verify the uninstallation.
   UNIX Linux

   ```
   ./imcl listInstalledPackages
   ```

Windows

```
imcl listInstalledPackages
```

5. If the uninstallation process displays a list of augmented WebSphere Application Server profiles, then you must unaugment these profiles in order to proceed with the uninstallation.
6. Optional: Uninstall IBM Installation Manager.
   Important: Before you can uninstall IBM Installation Manager, you must uninstall all of the packages that were installed by Installation Manager.
   Read the IBM Installation Manager Version 1.5 Information Center for information about using the uninstall script to perform this procedure.

Related concepts:
Installation overview
IBM Installation Manager and WebSphere eXtreme Scale product offerings
Related tasks:
Installing IBM Installation Manager using response files
Installing the product using a response file

Customizing WebSphere eXtreme Scale for z/OS

Using the WebSphere® Customization Toolbox, you can generate and run customized jobs to customize WebSphere eXtreme Scale for z/OS®.

Before you begin

- Verify that your system contains the latest level of WebSphere Application Server Network Deployment:
  - If you are running Version 7.0, your system must contain fix pack 21 at a minimum. See Installing your Version 7.0 application serving environment for more information.
  - If you are running Version 8.0, your system must contain fix pack 4 at minimum. See Installing your Version 8.0 application serving environment for more information.
- Install WebSphere eXtreme Scale for z/OS. See the *WebSphere eXtreme Scale Program Directory* on the Library Page for more information.

About this task

Using the WebSphere Customization Toolbox, generate customization definitions and upload and run customized jobs to customize WebSphere eXtreme Scale for z/OS.

- z/OS Installing the WebSphere Customization Toolbox
  Install the WebSphere Customization Toolbox to customize your WebSphere eXtreme Scale for z/OS environment.
- z/OS Generating customization definitions
  Use the Profile Management Tool function within the WebSphere Customization Toolbox to generate customization definitions and create customized jobs for WebSphere eXtreme Scale for z/OS.
- z/OS Uploading and running customized jobs
  After you generate the customization definitions, you can upload and run the customized jobs that are associated with the definitions to your WebSphere eXtreme Scale for z/OS system.

Installing the WebSphere Customization Toolbox

Install the WebSphere® Customization Toolbox to customize your WebSphere eXtreme Scale for z/OS® environment.

Before you begin

- Install WebSphere eXtreme Scale for z/OS. See the *WebSphere eXtreme Scale Program Directory* on the Library Page for more information.
- You must use the latest version of the WebSphere Customization Toolbox to successfully install the product extension files.
  8.5 Attention: You can use WebSphere Customization Toolbox Version 8.5 to customize WebSphere Application Server Version 7, Version 8, and Version 8.5.

About this task

The WebSphere Customization Tools is a workstation-based graphical tool you use to create customized jobs that build WebSphere eXtreme Scale for z/OS runtime environments.

Procedure

1. Use FTP to copy the xs.wct extension file from your z/OS system to the workstation on which you are installing the WebSphere Customization Tools. The extension files are in the /usr/lpp/zWebSphere/util/WCT directory on your z/OS operating system.
2. Download and install the latest version of the WebSphere Customization Toolbox.
3. Upload the xs.wct file to the WebSphere Customization Toolbox application.
   a. Start the WebSphere Customization Toolbox application on your workstation.
   b. Click Help > Software Updates > Manage Extension.

c. From the WebSphere Customization Toolbox Extension panel, click Install.
d. From the Source Archive File panel, click Browse, navigate to the directory in which you copied the xs.wct file in step 1, and click Open.
e. Click Next on the Source Archive panel.
f. Click Next on the Extension summary panel, and click Finish on the WebSphere Customization Toolbox Extension panel.
4. From the WebSphere Customization Toolbox Extension panel, click Install.
5. From the Source Archive File panel, click Browse, navigate to the directory in which you copied the xspf.wct file in step 1, and click Open.
6. Click Next on the Summary panel.
7. Click Next on the Extension summary panel, and click Finish on the WebSphere Customization Toolbox Extension panel

What to do next

After you upload both extension files and restart the WebSphere Customization Toolbox, you can use the Profile Management Tool to generate customization definitions for WebSphere eXtreme Scale for z/OS. See Generating customization definitions for more information.

Generating customization definitions

Use the Profile Management Tool function within the WebSphere® Customization Toolbox to generate customization definitions and create customized jobs for WebSphere eXtreme Scale for z/OS®.

Before you begin

- Verify that your system contains the latest level of WebSphere Application Server Network Deployment. See Customizing WebSphere eXtreme Scale for z/OS.

About this task

You can generate customization definitions using the Profile Management Tool, which is provided in the WebSphere Customization Tools. A *customization definition* is a set of files used to create customized jobs for configuring WebSphere eXtreme Scale for z/OS.

Procedure

1. Start the Profile Management Tool.
   - Windows Click Start > Programs > IBM WebSphere > WebSphere Customization Toolbox > WebSphere Customization Toolbox. After the application starts, click the Profile Management Tool tab.
   - Linux Run the shell script, `wct.sh`, which is in the /opt/IBM/WebSphere/Toolbox/ directory. After the application starts, click the Profile Management Tool tab.
2. Using the customization locations that you have added or created for WebSphere Application Server for z/OS, select the profile that you want to augment from the Customization Locations list of the existing WebSphere Application Server product version that is installed on your z/OS operating system.
   Note: Do not use the same location that you are using for other WebSphere eXtreme Scale customization definitions.
3. Select an environment from the Customization Definitions list for WebSphere Application Server for z/OS. Click Augment to create the response file and list of instructions for augmenting the WebSphere Application Server for z/OS runtime environment. The Profile Management Tool Environment Selection panel is displayed.
4. Select the environment to augment from the Environments list, and click Next.
   - Management
   - Application server
   - Managed (custom) node

   The Profile Management Tool Augment Selection panel is displayed.
5. Select the type of augmentation to apply from the list of Augment types, and click Next.
6. Complete the fields on the panels. Specify the values for the parameters that are used to create your WebSphere Application Server for z/OS runtime environment.
7. Click Augment to generate the customization definition.
8. Click Finish to close the dialog, and continue.

What to do next

Upload the customized job to your target z/OS system. See Uploading and running customized jobs for more information.

Uploading and running customized jobs

After you generate the customization definitions, you can upload and run the customized jobs that are associated with the definitions to your WebSphere® eXtreme Scale for z/OS® system.

Before you begin

Generate the customization definitions for the jobs that you want to upload to your z/OS system. For more information, see Generating customization definitions.

About this task

Upload and run the customized jobs that you created using the WebSphere Customization Tools to administer and monitor your WebSphere eXtreme Scale for z/OS environment.

Procedure

1. Upload the customized jobs. On the Customization Definitions tab, select the jobs that you want to upload and click Process. The Profile Management Tool Select Process Type panel is displayed.
2. Select the FTP upload type for the target z/OS operating system, and click Next. Specify the required information on the Upload Customization Definition panel.
3. Click Finish.
4. Run the customized jobs. Click the Customization Instructions tab, and follow the customization instructions for each job.

Creating and augmenting profiles for WebSphere eXtreme Scale

After you install the product, create unique types of profiles and augment existing profiles for WebSphere® eXtreme Scale.

Before you begin

Install WebSphere eXtreme Scale. See Installation overview for more information.

About this task

Augmenting profiles for use with WebSphere eXtreme Scale is optional, but is required in the following usage scenarios:

- To automatically start a catalog service or container in a WebSphere Application Server process. Without augmenting the server profiles, servers can only be started programmatically using the ServerFactory API or as separate processes with the **startOgServer** script.
- To use Performance Monitoring Infrastructure (PMI) to monitor WebSphere eXtreme Scale metrics.
- To display the version of WebSphere eXtreme Scale in the WebSphere Application Server administrative console.

If you are running WebSphere eXtreme Scale within WebSphere Application Server Version 6.1 , you can use the Profile Management Tool plug-in or the **manageprofiles** command to create and augment profiles.

- 8.5+ Using the graphical user interface to create profiles
  Use the graphical user interface (GUI), which is provided by the Profile Management Tool plug-in, to create profiles for WebSphere eXtreme Scale. A profile is a set of files that define the runtime environment.
- 8.5+ Using the graphical user interface to augment profiles
  After you install the product, you can augment an existing profile to make it compatible with WebSphere eXtreme Scale.
- 8.5+ manageprofiles command
  You can use the **manageprofiles** utility to create profiles with the WebSphere eXtreme Scale template, and augment and unaugment existing application server profiles with the eXtreme Scale augment templates. To use the features of the product, your environment must contain at least one profile augmented for the product.
- 8.5+ Non-root profiles
  Give a non-root user permissions for files and directories so that the non-root user can create a profile for the product. The non-root user can also augment a profile that was created by a root user, a different non-root user, or the same non-root user.

What to do next

Depending on which task you choose to complete, launch the First steps console for assistance with configuring and testing your product environment. The First steps console is in the following directory:

- *wxs_install_root*\firststeps\wxs\firststeps.bat
- *wxs_install_root*/firststeps/wxs/firststeps.sh

You can also create or augment additional profiles by repeating any of the preceding tasks.

Using the graphical user interface to create profiles

Use the graphical user interface (GUI), which is provided by the Profile Management Tool plug-in, to create profiles for WebSphere® eXtreme Scale. A profile is a set of files that define the runtime environment.

Before you begin

You cannot use the GUI to augment profiles in the following scenario:

- **64-bit installations of WebSphere Application Server**:

The profile management tool does not exist for 64-bit installations of WebSphere Application Server. Use the **manageprofiles** script from the command line for these installations.

About this task

To use the product features, the Profile Management Tool plug-in enables the GUI to assist you in setting up profiles, such as a WebSphere Application Server profile, a deployment manager profile, a cell profile, and a custom profile. You can augment profiles during or after the installation of WebSphere eXtreme Scale.

Procedure

Use the Profile Management Tool GUI to create profiles. Choose one of the following options to start the wizard:

- Select Profile Management Tool from the First steps console.
- Access the Profile Management Tool from the Start menu.
- Run the ./pmt.sh|bat script from the *install_root*/bin/ProfileManagement directory.

What to do next

You can create additional profiles or augment existing profiles. To restart the Profile Management tool, run the **./pmt.sh|bat** command from the *was_root*/bin/ProfileManagement directory, or select Profile Management Tool in the First steps console.

Start a catalog service, start containers, and configure TCP ports in your WebSphere Application Server environment. See Configuring WebSphere eXtreme Scale with WebSphere Application Server for more information.

Using the graphical user interface to augment profiles

After you install the product, you can augment an existing profile to make it compatible with WebSphere® eXtreme Scale.

About this task

When you augment an existing profile, you change the profile by applying a product-specific augmentation template. For example, WebSphere eXtreme Scale servers do not start automatically unless the server profile is augmented with the xs_augment template.

- Augment the profile with the xs_augment template if you installed the eXtreme Scale client or the client and server.

Procedure

Use the Profile Management Tool GUI to augment profiles for eXtreme Scale. Choose one of the following options to start the wizard:

- Select Profile Management Tool from the First steps console.
- Access the Profile Management Tool from the Start menu.
- Run the ./pmt.sh|bat script from the *was_root*/bin/ProfileManagement directory.

What to do next

You can augment additional profiles. To restart the Profile Management tool, run the **./pmt.sh|bat** command from the *was_root*/bin/ProfileManagement directory, or select Profile Management Tool in the First steps console.

Start a catalog service, start containers, and configure TCP ports in your WebSphere Application Server environment. See Configuring WebSphere eXtreme Scale with WebSphere Application Server for more information.

manageprofiles command

You can use the **manageprofiles** utility to create profiles with the WebSphere® eXtreme Scale template, and augment and unaugment existing application server profiles with the eXtreme Scale augment templates. To use the features of the product, your environment must contain at least one profile augmented for the product.

- Before you can create and augment profiles, you must install eXtreme Scale . See Installing WebSphere eXtreme Scale or WebSphere eXtreme Scale Client with WebSphere Application Server for more information.

Purpose

The **manageprofiles** command creates the runtime environment for a product process in a set of files called a profile. The profile defines the runtime environment. You can perform the following actions with the **manageprofiles** command:

- Create and augment a deployment manager profile
- Create and augment a custom profile
- Create and augment stand-alone application server profile

- Create and augment a cell profile
- Unaugment any type of profile

When you augment an existing profile, you change the profile by applying a product-specific augmentation template.

- Augment the profile with the xs_augment template if you installed the eXtreme Scale client or both the client and server.

Location

The command file is in the *install_root*/bin directory.

Usage

For detailed help, use the -help parameter:

```
./manageprofiles.sh|bat -create -templatePath install_root/profileTemplates/xs_augment/dmgr -help
```

In the following sections, each task that you can perform using the **manageprofiles** command, along with a list of required parameters, is described. For details on the optional parameters to specify for each task, see the **manageprofiles** command in the WebSphere Application Server Information Center.

Create a deployment manager profile

You can use the **manageprofiles** command to create a deployment manager profile. The deployment manager administers the application servers that are federated into the cell.

Parameters

-create
: Creates a profile. (Required)

-templatePath *template_path*
: Specifies the file path to the template. (Required)
Use the following format:

```
-templatePath install_root/profileTemplates/template_type/dmgr
```

Example

- Using the xs_augment template:

```
./manageprofiles.sh|bat -create -templatePath install_root/profileTemplates/xs_augment/dmgr
```

Create a custom profile

You can use the **manageprofiles** command to create a custom profile. A custom profile is an empty node that you customize through the deployment manager to include application servers, clusters, or other Java™ processes.

Parameters

-create
: Creates a profile. (Required)

-templatePath *template_path*
: Specifies the file path to the template. (Required)
Use the following format:

```
-templatePath install_root/profileTemplates/template_type/managed
```

Example

- Using the xs_augment template:

```
./manageprofiles.sh|bat -create -templatePath install_root/profileTemplates/xs_augment/managed
```

Create a stand-alone application server profile

You can use the **manageprofiles** command to create a stand-alone application server profile.

Parameters

-create
: Creates a profile. (Required)

-templatePath *template_path*
: Specifies the file path to the template. (Required)

Use the following format:

```
-templatePath install_root/profileTemplates/template_type/default
```

Example

- Using the xs_augment template:

```
./manageprofiles.sh|bat -create -templatePath install_root/profileTemplates/xs_augment/default
```

Create a cell profile

You can use the **manageprofiles** command to create a cell profile, which consists of a deployment manager and an application server.

Parameters

Specify the following parameters in the deployment manager template:

-create
Creates a profile. (Required)
-templatePath *template_path*
Specifies the file path to the template. (Required)
Use the following format:

```
-templatePath install_root/profileTemplates/template_type/cell/dmgr
```

Specify the following parameters with the application server template:

-create
Creates a profile. (Required)
-templatePath *template_path*
Specifies the file path to the template. (Required)
Use the following format:

```
-templatePath install_root/profileTemplates/template_type/cell/default
```

Example

- Using the xs_augment template:

```
./manageprofiles.sh|bat -create -templatePath install_root/profileTemplates/xs_augment/cell/dmgr
-nodeProfilePath install_root/profiles/AppSrv01 -cellName cell01dmgr -nodeName node01dmgr
-appServerNodeName node01

./manageprofiles.sh|bat -create -templatePath install_root/profileTemplates/xs_augment/cell/default
-dmgrProfilePath install_root/profiles/Dmgr01 -portsFile
install_root/profiles/Dmgr01/properties/portdef.props -nodePortsFile
install_root/profiles/Dmgr01/properties/nodeportdef.props -cellName cell01dmgr
-nodeName node01dmgr -appServerNodeName node01
```

Augment a deployment manager profile

You can use the **manageprofiles** command to augment a deployment manager profile.

Parameters

-augment
Augments the existing profile. (Required)
-profileName
Specifies the name of the profile. (Required)
-templatePath *template_path*
Specifies the path to the template files that are located in the installation root directory. (Required)
Use the following format:

```
-templatePath install_root/profileTemplates/template_type/dmgr
```

Example

- Using the xs_augment template:

```
./manageprofiles.sh|bat -augment -profileName profile01
-templatePath install_root/profileTemplates/xs_augment/dmgr
```

Augment a custom profile

You can use the **manageprofiles** command to augment a custom profile.

Parameters

-augment
Augments the existing profile. (Required)
-profileName
Specifies the name of the profile. (Required)
-templatePath *template_path*
Specifies the path to the template files that are located in the installation root directory. (Required)
Use the following format:

```
-templatePath install_root/profileTemplates/template_type/managed
```

Example

- Using the xs_augment template:

```
./manageprofiles.sh|bat -augment -profileName profile01
-templatePath install_root/profileTemplates/xs_augment/managed
```

Augment a stand-alone application server profile

You can use the **manageprofiles** command to augment a stand-alone application server profile.

Parameters

-augment
Augments the existing profile. (Required)
-profileName
Specifies the name of the profile. (Required)
-templatePath *template_path*
Specifies the path to the template files that are located in the installation root directory. (Required)
Use the following format:

```
-templatePath install_root/profileTemplates/template_type/default
```

Example

- Using the xs_augment template:

```
./manageprofiles.sh|bat -augment -profileName profile01
-templatePath install_root/profileTemplates/xs_augment/default
```

Augment a cell profile

You can use the **manageprofiles** command to augment a cell profile.

Parameters

Specify the following parameters for the deployment manager profile:

-augment
Augments the existing profile. (Required)
-profileName
Specifies the name of the profile. (Required)
-templatePath *template_path*
Specifies the path to the template files that are located in the installation root directory. (Required)
Use the following format:

```
-templatePath install_root/profileTemplates/template_type/cell/dmgr
```

Specify the following parameters for the application server profile:

-augment
Augments the existing profile. (Required)
-profileName
Specifies the name of the profile. (Required)
-templatePath *template_path*
Specifies the path to the template files that are located in the installation root directory. (Required)
Use the following format:

```
-templatePath install_root/profileTemplates/template_type/cell/default
```

Example

- Using the xs_augment template:

```
./manageprofiles.sh|bat -augment -profileName profile01 -templatePath install_root
                                /profileTemplates/xs_augment/cell/dmgr

./manageprofiles.sh|bat -augment -profileName profile01 -templatePath install_root
                                /profileTemplates/xs_augment/cell/default
```

Unaugment a profile

To unaugment a profile, specify the -ignoreStack parameter with the -templatePath parameter in addition to specifying the required -unaugment and -profileName parameters.

Parameters

-unaugment
: Unaugments a previously augmented profile. (Required)

-profileName
: Specifies the name of the profile. The parameter is issued by default if no values are specified. (Required)

-templatePath *template_path*
: Specifies the path to the template files that are located in the installation root directory. (Optional)
Use the following format:

```
-templatePath install_root/profileTemplates/template_type/profile_type
```

where *template_type* is `xs_augment` or `pf_augment` and *profile_type* is one of four profile types:

- `dmgr`: deployment manager profile
- `managed`: custom profile
- `default`: stand-alone application server profile
- `cell`: cell profile

-ignoreStack
: Used with the -templatePath parameter to unaugment a particular profile that has been augmented. (Optional)

Example

- Using the xs_augment template:

```
./manageprofiles.sh|bat -unaugment -profileName profile01 -ignoreStack
-templatePath install_root/profileTemplates/xs_augment/profile_type
```

Non-root profiles

Give a non-root user permissions for files and directories so that the non-root user can create a profile for the product. The non-root user can also augment a profile that was created by a root user, a different non-root user, or the same non-root user.

In a WebSphere® Application Server environment, non-root (non-administrator) users are limited in being able to create and use profiles in their environment. Within the Profile Management tool plug-in, unique names and port values are disabled for non-root users. The non-root user must change the default field values in the Profile Management tool for the profile name, node name, cell name, and port assignments. Consider assigning non-root users a range of values for each of the fields. You can assign responsibility to the non-root users for adhering to their proper value ranges and for maintaining the integrity of their own definitions.

The term *installer* refers to either a root or non-root user. As an installer, you can grant non-root users permissions to create profiles and establish their own product environments. For example, a non-root user might create a product environment to test application deployment with a profile that the user owns. Specific tasks that you can complete to allow non-root profile creation include the following items:

- Creating a profile and assigning ownership of the profile directory to a non-root user so that the non-root user can start WebSphere Application Server for a specific profile.
- Granting write permission of the appropriate files and directories to a non-root user, which allows the non-root user to then create the profile. With this task, you can create a group for users who are authorized to create profiles, or give individual users the ability to create profiles.
- Installing maintenance packages for the product, which includes required services for existing profiles that are owned by a non- user. As the installer, you are the owner of any new files that the maintenance package creates.

For more information about creating profiles for non-root users, see Creating profiles for non-root users .

As an installer, you can also grant permissions for a non-root user to augment profiles. For example, a non-root user can augment a profile that is created by an installer, or augment a profile that they create. Follow the WebSphere Application Server Network Deployment non-root user augmentation process.

However, when a non-root user augments a profile that is created by the installer, the non-root user does not need to create the following files before augmentation. The following files were established during the profile creation process:

- *was_root*/logs/manageprofiles.xml

- *was_root*/properties/fsdb.xml
- *was_root*/properties/profileRegistry.xml

When a non-root user augments a profile that the user creates, the non-root user must modify the permissions for the documents that are located within the eXtreme Scale profile templates.

Attention: You can also use a non-root (non-administrator) profile for WebSphere eXtreme Scale in a stand-alone environmen, a profile that is not in WebSphere Application Server. You must change the owner of the ObjectGrid directory to the non-root profile. Then you can log in with that non-root profile and operate eXtreme Scale as you normally would for a root (administrator) profile.

Taking the first steps after installation

After complete and verify the installation, you can begin to use WebSphere® eXtreme Scale to create your data grid.

Procedure

1. Update your installation by applying maintenance.
   More information: Updating eXtreme Scale servers.
2. If you are using WebSphere eXtreme Scale for the first time, you can use the Getting started information to learn more about how to use the product.
   More information: Getting started
3. Configure the product. Create properties and XML files to define the configuration for data grids, servers, and clients. You can also configure cache or database integration, REST data services, or OSGi plug-ins.
   More information: Configuring
4. Develop an application that accesses the data grid.
   More information: Developing applications
5. Start and administer container and catalog servers with your configuration files and data grid application.
   More information: Administering
6. Monitor the performance of your configuration with the various monitoring tools.
   More information: Monitoring

Troubleshooting the product installation

IBM® Installation Manager is a common installer for many IBM software products that you use to install this version of WebSphere® eXtreme Scale.

Results

Logging and tracing:

- An easy way to view the logs is to open Installation Manager and go to File > View Log. An individual log file can be opened by selecting it in the table and then clicking the Open log file icon.
- Logs are located in the `logs` directory of Installation Manager's application data location. For example:
  - Windows **Administrative installation:**

    ```
    C:\Documents and Settings\All Users\Application Data\IBM\Installation Manager
    ```

  - Windows **Non-administrative installation:**

    ```
    C:\Documents and Settings\user_name\Application Data\IBM\Installation Manager
    ```

  - UNIX Linux **Administrative installation:**

    ```
    /var/IBM/InstallationManager
    ```

  - UNIX Linux **Non-administrative installation:**

    ```
    user_home/var/ibm/InstallationManager
    ```

- The main log files are time-stamped XML files in the `logs` directory, and they can be viewed using any standard web browser.
- The `log.properties` file in the `logs` directory specifies the level of logging or tracing that Installation Manager uses. To turn on tracing for the WebSphere eXtreme Scale plug-ins, for example, create a `log.properties` file with the following content:

  ```
  com.ibm.ws=DEBUG
  com.ibm.cic.agent.core.Engine=DEBUG
  global=DEBUG
  ```

  Restart Installation Manager as necessary, and Installation Manager outputs traces for theWebSphere eXtreme Scale plug-ins.

Notes on troubleshooting:

- UNIX Linux By default, some HP-UX systems are configured to not use DNS to resolve host names. This could result in Installation Manager not being able to connect to an external repository.
  You can ping the repository, but nslookup does not return anything.

  Work with your system administrator to configure your machine to use DNS, or use the IP address of the repository.

- In some cases, you might need to bypass existing checking mechanisms in Installation Manager.
  - On some network file systems, disk space might not be reported correctly at times; and you might need to bypass disk-space checking and proceed with your installation.
    To disable disk-space checking, specify the following system property in the config.ini file in *IM_install_root*/eclipse/configuration and restart Installation Manager:

    ```
    cic.override.disk.space=sizeunit
    ```

    where *size* is a positive integer and *unit* is blank for bytes, k for kilo, m for megabytes, or g for gigabytes. For example:

    ```
    cic.override.disk.space=120 (120 bytes)
    cic.override.disk.space=130k (130 kilobytes)
    cic.override.disk.space=140m (140 megabytes)
    cic.override.disk.space=150g (150 gigabytes)
    cic.override.disk.space=true
    ```

    Installation Manager will report a disk-space size of Long.MAX_VALUE. Instead of displaying a very large amount of available disk space, N/A is displayed.
  - To bypass operating-system prerequisite checking, add `disableOSPrereqChecking=true` to the config.ini file in *IM_install_root*/eclipse/configuration and restart Installation Manager.

  If you need to use any of these bypass methods, contact IBM Support for assistance in developing a solution that does not involve bypassing the Installation Manager checking mechanisms.
- For more information on using Installation Manager, read the IBM Installation Manager Version 1.5 Information Center.
  Read the release notes to learn more about the latest version of Installation Manager. To access the release notes, complete the following task:
  - Windows Click **Start > Programs > IBM Installation Manager > Release Notes®**.
  - UNIX Linux Go to the documentation subdirectory in the directory where Installation Manager is installed, and open the readme.html file.
- If a fatal error occurs when you try to install the product, take the following steps:
  - Make a backup copy of your current product installation directory in case IBM support needs to review it later.
  - Use Installation Manager to uninstall everything that you have installed under the product installation location (package group). You might run into errors, but they can be safely ignored.
  - Delete everything that remains in the product installation directory.
  - Use Installation Manager to reinstall the product to the same location or to a new one.

Note on version and history information: The **versionInfo** and **historyInfo** commands return version and history information based on all of the installation, uninstallation, update, and rollback activities performed on the system.

Upgrading and migrating WebSphere eXtreme Scale

You can migrate to Version 8.5 from previous versions, or you can apply maintenance packages. To avoid outages, you must consider the order in which you apply the updates to the servers in your configuration.

- Updating eXtreme Scale servers
  You can upgrade WebSphere® eXtreme Scale to a new version, either by applying maintenance or installing a new version, without interrupting service.
- Migrating to WebSphere eXtreme ScaleVersion 8.5
  With the WebSphere eXtreme Scale installer, you cannot upgrade or modify a previous installation. You must uninstall the previous version before you install the new version. You do not need to migrate your configuration files because they are backward compatible. However, if you changed any of the script files that are shipped with the product, you must reapply these changes to the updated script files.
- Installing fix packs using IBM Installation Manager
  You can use IBM® Installation Manager to update the product with the fix packs that are available for WebSphere eXtreme Scale product offerings. Fix packs can be installed from the GUI, the command line, or using response files.
- 8.5+ Uninstalling fix packs using IBM Installation Manager
  You can use IBM Installation Manager to rollback WebSphere eXtreme Scale product offerings to an earlier version. You can uninstall fix packs from the GUI, the command line, or using response files.
- 8.5+ Updating WebSphere eXtreme Scale on WebSphere Application Server
  When you migrate WebSphere Application Server to a new version, you can also migrate the WebSphere eXtreme Scale configuration to the new WebSphere Application Server installation.
- xsadmin tool to xscmd tool migration
  In previous releases, the **xsadmin** tool was a sample command-line utility to monitor the state of the environment. The **xscmd** tool has been introduced as an officially supported administrative and monitoring command-line tool. If you were previously using the **xsadmin** tool, consider migrating your commands to the new **xscmd** tool.
- Deprecated properties and APIs
  The following list of properties and APIs were deprecated in the specified releases. Use the recommended migration action to determine how to update your configuration.

Updating eXtreme Scale servers

You can upgrade WebSphere® eXtreme Scale to a new version, either by applying maintenance or installing a new version, without interrupting service.

Before you begin

You must have the binary file for the major version release or maintenance that you want to apply. You can get the latest information about the available releases and maintenance packages from the IBM support portal for WebSphere eXtreme Scale.

About this task

To upgrade without service interruption, first upgrade catalog servers, then upgrade the container servers and client servers. If you have a server and client installation on the same physical server, you can upgrade the full installations on each physical server. If you have client-only installations, upgrade these installations last.

Procedure

1. Upgrade the catalog service tier, repeating the following steps for each catalog server in the data grid. Upgrade the catalog service tier before upgrading any container servers or clients. Individual catalog servers can interoperate with version compatibility, so you can apply upgrades to one catalog server at a time without interrupting service.
   a. If you are running with quorum mechanism enabled, check for a healthy quorum status. Run the following command:

      ```
      xsadmin -quorumStatus
      ```

      ```
      xscmd -c showQuorumStatus
      ```

      This result indicates that all the catalog servers are connected.
   b. If you are using multi-master replication between two catalog service domains, dismiss the link between the two catalog service domains while you are upgrading the catalog servers.

      ```
      xsadmin -ch host -p 1099 -dismissLink domain_name
      ```

      ```
      xscmd -c dismissLink -cep host:2809 -fd domain_name
      ```

      You only need to run this command from one of the catalog service domains to remove the link between two catalog service domains.
   c. Shut down one of the catalog servers. You can use the **stopOgServer** command, the **xscmd -c teardown** command, or shut down the application server that is running the catalog service in WebSphere Application Server. There are no requirements for the order in which you stop the catalog servers, but shutting down the primary catalog server last reduces turnover. To determine which catalog server is the primary, look for the CWOBJ8106 message in the log files. Under normal conditions when the quorum mechanism is enabled, quorum is maintained when a catalog server is shut down, but it is a best practice to query quorum status after each shutdown with the **xscmd -c showQuorumStatus** command.
      You can provide a specific list of servers to stop to the **stopOgServer** command, or the **xscmd -c teardown** commands:

      ```
      stopOgServer server_name
      ```

      ```
      xsadmin -teardown server_name
      ```

      ```
      xscmd -c teardown -sl server_name
      ```

      With the previous examples, the **stopOgServer** , or **xscmd -c teardown** commands are completing the same shutdown tasks. However, you can filter the servers to stop with the **xscmd -c teardown** command. See Stopping servers gracefully with the xscmd utility for more information about filtering the servers by zone or host name. The teardown command filters out the matching servers and asks if the selected servers are correct.
   d. Install the updates on the catalog server. You can either migrate the catalog server to a new major release of the product or apply a maintenance package. See the following topics for more information:
      - To migrate from a Version 7.1.x installation: Migrating to WebSphere eXtreme ScaleVersion 8.5
   e. Restart the catalog server.
      If you are using a stand-alone environment, see Starting a stand-alone catalog service for more information. If you are using a WebSphere Application Server environment, see Starting and stopping servers in a WebSphere Application Server environment for more information.

      The catalog server runs in compatibility mode until all the catalog servers are moved to the same level. Compatibility mode mostly applies to major release migrations because new functions are not available on the servers that are not migrated. No restrictions exist on how long catalog servers can run in compatibility mode, but the best practice is to migrate all catalog servers to the same level as soon as possible.

   f. Verify that the catalog server started successfully. Ensure that the following **xscmd** commands return valid results:

      ```
      xscmd -c routetable -cep cathost:2809
      xscmd -c showMapSizes -cep cathost:2809
      ```

      Important: These commands must contain the -cep *<catalog_server_host>:<listener_port>* value for the restarted catalog server.
   g. Apply updates to the remaining catalog servers in your configuration.
2. Upgrade the container servers, repeating the following steps for each container server in the data grid. You can upgrade container servers in any order.
   a. Stop the container servers that you want to upgrade. You can stop the container servers that you want to updgrade with the **stopOgserver** command, or with the **xscmd -c teardown** command. For more information, see Stopping stand-alone servers and Stopping servers gracefully with the xscmd utility.
      By running the **xscmd -c teardown** or **stopOgserver** commands to handle multiple servers in parallel, the placement mechanism can move shards in larger groups. However, do not to take down too many servers at the same time. The resources of servers that remain might become overloaded.

b. Verify that the container servers were stopped and removed from the data grid. . Run the following **xscmd** commands and verify that the results do not contain the stopped container servers.

```
xscmd -c routetable
xscmd -c showMapSizes
```

If these commands are run too soon after container servers are stopped, correct results might not be returned. Wait a few minutes and try running the commands again.

c. Install the updates on the container servers. You can either migrate the container servers to a new major release of the product or apply a maintenance package. See the following topics for more information:
   - To migrate from a Version 7.1.x installation: Migrating to WebSphere eXtreme ScaleVersion 8.5

d. Restart your container servers.

e. Verify that the container servers were restarted and added to the data grid. Run the following **xscmd** commands and verify that the results contain the restarted container servers.

```
xscmd -c routetable
xscmd -c showMapSizes
```

If these commands are run too soon after container servers are started, correct results might not be returned. Wait a few minutes and try running the commands again.

f. Upgrade any remaining container servers in your configuration.

3. If you are using multi-master replication, reconnect your catalog service domains. Use the **xscmd -c establishLink** command to reconnect the catalog service domains.

```
xsadmin -ch host -p 1099 -establishLink dname fdHostA:2809,fdHostB:2809
```

```
xscmd -c establishLink -cep host:2809 -fd dname -fe fdHostA:2809,fdHostB:2809
```

4. 8.5+ To check that all servers are using the new version of WebSphere eXtreme Scale, issue the **xscmd -c showinfo** command.

```
xscmd -c showinfo
```

5. Upgrade the client-only installations.

What to do next

- You can also use these steps to revert to an older version or to uninstall maintenance packages. However, if you revert to Version 7.1.0 when you are using multi-master replication, the two-way replication might not function correctly when you re-establish the links. In this situation, restart both catalog service domains and re-link the catalog service domains with the **establishLink** command.

Related concepts:
Configuration considerations for multi-master topologies
8.5+ Installing fix packs using IBM Installation Manager
Related tasks:
Administering with the xscmd utility
Retrieving eXtreme Scale environment information with the xscmd utility
Configuring multiple data center topologies
Migrating to WebSphere eXtreme ScaleVersion 8.5
Starting and stopping stand-alone servers
Starting and stopping servers in a WebSphere Application Server environment

Migrating to WebSphere eXtreme ScaleVersion 8.5

With the WebSphere® eXtreme Scale installer, you cannot upgrade or modify a previous installation. You must uninstall the previous version before you install the new version. You do not need to migrate your configuration files because they are backward compatible. However, if you changed any of the script files that are shipped with the product, you must reapply these changes to the updated script files.

Before you begin

Verify that your systems meet the minimum requirements for the product versions you plan to migrate and install. See Hardware and software requirements for more information.

About this task

Merge any modified product script files with new product script files in the /bin directory to maintain your changes.
Tip: If you did not modify the script files that are installed with the product, you are not required to complete the following migration steps. Instead, you can upgrade to Version 8.5 by uninstalling the previous version and installing the new version in the same directory.

Procedure

1. Stop all processes that are using WebSphere eXtreme Scale.
   - Stop all processes that are running in your stand-alone WebSphere eXtreme Scale environment. For more information, see Stopping stand-alone servers.

- Read about command-line utilities to stop all processes that are running in your WebSphere Application Server or WebSphere Application Server Network Deployment environment.

2. Save any modified scripts from your current installation directory to a temporary directory.
3. Uninstall the product. For more information, see Uninstalling the product using IBM Installation Manager.
4. Install WebSphere eXtreme Scale Version 8.5. See Installing for more information.
5. Merge your changes from the files in the temporary directory to the new product script files in the /bin directory.
6. Start all of your WebSphere eXtreme Scale processes to begin using the product. For more information, see Administering.

Related concepts:
Configuration considerations for multi-master topologies
8.5+ Installing fix packs using IBM Installation Manager
Related tasks:
Retrieving eXtreme Scale environment information with the xscmd utility
Updating eXtreme Scale servers
Configuring multiple data center topologies
Starting and stopping stand-alone servers
Starting and stopping servers in a WebSphere Application Server environment

Updating WebSphere eXtreme Scale on WebSphere Application Server

8.5+ When you migrate WebSphere® Application Server to a new version, you can also migrate the WebSphere eXtreme Scale configuration to the new WebSphere Application Server installation.

Before you begin

- It is assumed that both WebSphere eXtreme Scale Version 7 and WebSphere eXtreme Scale Version 8 are being installed on the same server.
- Migrate WebSphere Application Server Version 7 to WebSphere Application Server Version 8. For more information, see Migrating product configurations.
- Install WebSphere eXtreme Scale Version 8 on your WebSphere Application Server Version 8 installation. For more information, see Installing WebSphere eXtreme Scale or WebSphere eXtreme Scale Client with WebSphere Application Server. All WebSphere eXtreme Scale migration scripts must be run from WebSphere eXtreme Scale Version 8.5 or later. For example, if you are migrating from Version 7.x to Version 8, run the scripts for migration from the *<WXS_v8_install_root>*/bin directory.

About this task

When you install a new version of WebSphere Application Server that has WebSphere eXtreme Scale integration, you first upgrade WebSphere Application Server with the normal process. Then, install the new version of WebSphere eXtreme Scale on your new installation. Then, you can use the **xsmigration** script to move the WebSphere eXtreme Scale configuration information to the new WebSphere Application Server installation.

Procedure

1. Migrate the deployment manager-related configuration from Version 7 to Version 8.
   a. Run the WebSphere Application Server backup script. For more information, see WASPreUpgrade command.
   b. Stop the deployment manager.
   c. Access the deployment manager server in your WebSphere eXtreme Scale configuration and run the migration script.
      i. Change the directory to: *<WXS_v8_install_root>*/bin
      ii. Run the following command:

```
xsmigration.bat|sh -targetwashome <WAS8x_HOME>
-sourcewashome <WAS7x_HOME> -targetprofilepath <WAS8x_DmgrProfile>
 -sourceprofilepath <WAS7x_DmgrProfile>
```

      where
      - *<WAS8x_HOME>* is the root location of the WebSphere Application Server Version 8.x installation. Example /opt/IBM/WebSphere8
      - *<WAS7x_HOME>* is the root location of the WebSphere Application Server Version 7.x installation Example:/opt/IBM/WebSphere7
      - *<WAS8x_DmgrProfile>* is the location of the WebSphere Application Server Version 8.x deployment manager profile. Example: /opt/IBM/WebSphere8/profiles/DMgr01
      - *<WAS7x_DmgrProfile>* is the location of the WebSphere Application Server Version 7.x deployment manager profile. Example: /opt/IBM/WebSphere7/profiles/DMgr01
2. Migrate the application server related configuration from Version 7 to Version 8.
   a. Change the directory to: *<WXS_v8_install_root>*/bin.
   b. Run the following command:

```
xsmigration.bat|sh -targetwashome <WAS8x_HOME>
-sourcewashome <WAS7x_HOME> -targetprofilepath <WAS8x_AppServerProfile>
 -sourceprofilepath <WAS7x_AppServerProfile>
```

   where
   - *<WAS8x_HOME>* is the root location of the WebSphere Application Server Version 8.x installation. Example /opt/IBM/WebSphere8
   - *<WAS7x_HOME>* is the root location of the WebSphere Application Server Version 7.x installation Example:/opt/IBM/WebSphere7
   - *<WAS8x_AppServerProfile>* is the location of the WebSphere Application Server Version 8.x application server profile. Example: /opt/IBM/WebSphere8/profiles/AppServer01
   - *<WAS7x_AppServerProfile>* is the location of the WebSphere Application Server Version 7.x application server profile. Example: /opt/IBM/WebSphere7/profiles/AppServer01

3. Restart the WebSphere Application Server Version 8 deployment manager and synchronize all the managed nodes.

xsadmin tool to xscmd tool migration

In previous releases, the **xsadmin** tool was a sample command-line utility to monitor the state of the environment. The **xscmd** tool has been introduced as an officially supported administrative and monitoring command-line tool. If you were previously using the **xsadmin** tool, consider migrating your commands to the new **xscmd** tool.

xsadmin and xscmd command equivalents

Important: 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.

Table 1. Arguments for the **xsadmin** utility and **xscmd** equivalent commands. Some **xscmd** commands have a short form and a long form. The short form commands have one dash (-), and the long form commands have two dashes (--). You can use either form interchangeably.

xsadmin Command Line Argument	xscmd Equivalent Command	xscmd Command Parameters
-bp	• -cep *hostname:listener_port* • --catalogEndpoint *hostname:listener_port*	n/a
-ch	• -cep *hostname:listener_port* • --catalogEndpoint *hostname:listener_port*	n/a
-clear	-c clearGrid	-g, -ms, -v, -m, (-cep)
-containers	• -c showPlacement -container*containerName* • -c showPlacement -server *serverName*	-e, -i, , -st, -snp, -ct, -s, -p, -hf, -z, -g, -m, -ms
-continuous	n/a	n/a
-coregroups	• • -c listCoreGroupMembers -cg *core_group*	n/a
-dismissLink <*catalog_service_domain*>	-c dismissLink	• -fd <*foreignCatalogServiceDomain*> • --foreignCatalogServiceDomain <*foreignCatalogServiceDomain*>
-dmgr	n/a - this argument is automatically determined with **xscmd**	n/a
-empties	arg specific to a new command	n/a
-establishLink <*foreign_domain_name*> <*host1:port1,host2:port2...*>	-c establishLink	• -fd <*foreignCatalogServiceDomain*> -fe <*host1:port1,host2:port2...*> • --foreignCatalogServiceDomain <*foreignCatalogServiceDomain*> -foreignEndPoints <*host1:port1,host2:port2...*>
-fc	• -ct • --container	n/a
-fh	• -hf • --hostFilter	n/a
-fm	• -m • --map	n/a
-fnp	• -snp • --serversWithNoPrimaries	n/a
-fp	• -p • --partitionId	n/a
-fs	• -s • --server	n/a
-fst	• -st <*shard_type*> • --shardType <*shard_type*> Shard values: P=primary A=asyncReplica S=syncReplica	n/a

xsadmin Command Line Argument	xscmd Equivalent Command	xscmd Command Parameters
-fz	• -z • --zone	n/a
-force	arg specific to a new command	
-g	• -g • --objectGrid	n/a
-getstatsspec	-c getStatsSpec	n/a
-getTraceSpec	-c getTraceSpec	n/a
-h	You can run help with or without a specific command name: • -h • --help • -h <*command_name*> • --help <*command_name*>	n/a
-hosts	-c listHosts	-g, -ms, -st, -c, -s, -hf, -z
-jmxUrl	• -cep *hostname*:*listener_port* • --catalogEndpoint *hostname*:*listener_port*	n/a
-l	-c listObjectGridNames	n/a
-m	• -ms • --mapSet	n/a
-mapsizes	-c showMapSizes	-g, -ms, -i, [-ct, -z, -s, -hf, sht [P,A,S], -p]
-mbeanservers	-c listAllJMXAddresses	n/a
-overridequorum	-c overrideQuorum	n/a
-password	• -pwd • --password	n/a
-p	• -cep *hostname*:*listener_port* • --catalogEndpoint *hostname*:*listener_port*	n/a
-placementStatus	-c placementServiceStatus	-g, -ms
-primaries	-c showPlacement -sf P	-e, -i, , -st, -snp, -ct, -s, -p, -hf, -z, -g, -m, -ms
-profile	To save the current security settings as a security profile: • -ssp *profile_name* • --saveSecProfile *profile_name* To use a specified security profile: • -sp *profile_name* • --securityProfile *profile_name*	
-quorumstatus	-c showQuorumStatus	n/a
-releaseShard <*container_server_name*> <*objectgrid_name*> <*map_set_name*> <*partition_name*>	-c releaseShard	-c, -g, -ms, -p
-reserved	• -sf R • --shardFilter R	n/a
-reserveShard <*container_server_name*> <*objectgrid_name*> <*map_set_name*> <*partition_name*>	-c reserveShard	-c, -g, -ms, -p
-resumeBalancing <*objectgrid_name*> <*map_set_name*>	-c resumeBalancing	-g, -ms
-revisions	-c revisions	-s,-p,-g,-m
-routetable	-c routetable	-z, -hf,-p,-g,-ms
-settracespec <*trace_string*>	-c setTraceSpec	-spec <*trace_string*>

xsadmin Command Line Argument	xscmd Equivalent Command	xscmd Command Parameters
-swapShardWithPrimary <*container_server_name*> <*objectgrid_name*> <*map_set_name*> <*partition_name*>	-c swapShardWithPrimary	-c -g, -ms, -p
-setstatsspec <*stats_spec*>	-c setStatsSpec	-spec <*stats_spec*>
-suspendBalancing <*objectgrid_name*> <*map_set_name*>	-c suspendBalancing	-g, -ms
-ssl	• -ssl • --enableSSL	n/a
-teardown	-c teardown	-f, , -st, -snp, -c, -s, -p, -hf, -z, -g, -ms, -m
-triggerPlacement	-c triggerPlacement	-g, -ms
-trustPass	• -tsp • --trustStorePassword	n/a
-trustPath	• -ts • --trustStore	n/a
-trustType	• -tst • --trustStoreType	n/a
-unassigned	-c showPlacement -sf U	-e, -i, , -st, -snp, -ct, -s, -p, -hf, -z, -g, -m, -ms
-username	• -user • --username	n/a
-v	• -v • --verbose	n/a
-xml	-c showPlacement	n/a

Related tasks:
Configuring security profiles for the xscmd utility
Administering with the xscmd utility
Monitoring with the xscmd utility

Deprecated properties and APIs

The following list of properties and APIs were deprecated in the specified releases. Use the recommended migration action to determine how to update your configuration.

8.5+

Deprecated items in Version 8.5

Table 1. Deprecated properties and APIs

Deprecation	Recommended migration action
WebSphereTransactionCallback This plug-in was used to manage data grid transactions with enterprise applications that run in a WebSphere® Application Server environment.	8.5+ The WebSphereTransactionCallback interface has been replaced by the WebSphere eXtreme Scale resource adapter, which enables Java Transaction API (JTA) transaction management. You can install this resource adapter on WebSphere Application Server or other Java Platform, Enterprise Edition (Java EE) application servers. The WebSphereTransactionCallback plug-in is not an enlisted JTA API, and therefore, is not designed to roll back the JTA transaction if the commit fails.

Deprecated items in Version 7.1.1

Table 2. Deprecated properties and APIs

Deprecation	Recommended migration action

Deprecation	Recommended migration action
com.ibm.websphere.objectgrid.plugins.builtins.TranPropListener class This class was used to propagate successful ObjectGrid transaction commit processes to other WebSphere application servers hosting the same ObjectGrid instance, based upon the ObjectGrid name.	The TranPropListener interface has been replaced by the JMSObjectGridEventListener interface, which is a JMS-based implementation of the ObjectGridEventListener interface. It supports client-side, near cache invalidation and peer-to-peer replication.
com.ibm.websphere.objectgrid.plugins.OptimisticCallback class This class was used to provide optimistic comparison operations for the values of a map.	The OptimisticCallback plug-in has been replaced by the ValueDataSerializer.Versionable interface, which you can implement when you use the DataSerializer plug-in with the COPY_TO_BYTES copy mode or when you use the @Version annotation with the EntityManager API. See the API documentation for more information.
com.ibm.websphere.objectgrid.plugins.NoVersioningOptimisticCallback plug-in This plug-in was used for optimistic locking without doing version checking. With this built-in OptimisticCallback handler, the loader handled version checking, but optimistic locking was used to ensure that committed data is always returned on a read.	The NoVersioningOptimisticCallback interface extends the OptimisticCallback interface. Therefore, use the pessimistic locking strategy with a default transaction isolation of READ_COMMITTED or lower. See Tuning locking performance for more information.
com.ibm.websphere.objectgrid.plugins.ObjectTransformer class This plug-iin was used to serialize, deserialize, and copy objects into the cache.	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.
com.ibm.websphere.objectgrid.BackingMap.setMapEventListeners method This method was used to set the list of MapEventListener objects.	Use either the addMapEventListener(EventListener) or removeMapEventListener(EventListener) methods to add or remove event listeners from a backing map.
com.ibm.websphere.objectgrid.ObjectGrid.setEventListeners method This method was used to overwrite the current list of ObjectGridEventListener objects and replace it with the supplied list of ObjectGridEventListeners objects.	Use either the addEventListener(EventListener) or removeEventListener(EventListener) methods to add or remove event listeners or life cycle listeners from the data grid.

Stabilized features in Version 7.1.1

If a feature is listed as stabilized, IBM does not currently plan to deprecate or remove this capability in a subsequent release of the product; but future investment will be focused on the alternative function. Users do not need to change any existing applications and scripts that use a stabilized function; but they should consider using the strategic alternative for new applications.

Table 3. Deprecated properties and APIs

Stabilized feature	Recommended migration action
xsadmin The xsadmin utility is provided as a sample of how you can create custom utilities for your deployment.	Use the **xscmd** utility to complete administrative tasks in the environment such as: establishing multi-master replication links, overriding quorum, and stopping groups of servers with the teardown command.

Deprecated items in Version 7.1

Table 4. Deprecated properties and APIs

Deprecation	Recommended migration action
catalog.services.cluster cell and server property: This custom property was used to define a group of catalog servers in the WebSphere Application Server configuration.	This custom property is deprecated starting in the Version 7.1 release. Create a catalog service domain in the WebSphere Application Server administrative console, which creates the same configuration as using the custom property. See Creating catalog service domains in WebSphere Application Server for more information.
CoreGroupServicesMBean MBean and interface	This MBean is deprecated starting in the Version 7.1 release. Use the CatalogServiceManagementMBean instead.
ServerMBean.updateTraceSpec() MBean operation	This operation is deprecated starting in the Version 7.1 release. Use the TraceSpec attribute on the DynamicServerMBean instead.
CoreGroupServicesMBean MBean	This MBean is deprecated starting in the Version 7.1 release. Use the CatalogServiceManagementMbean MBean instead.

Deprecation	Recommended migration action
ServiceUnavailableException exception	This exception is deprecated starting in the Version 7.1 release. Use the TargetNotAvailableException exception instead.
	The capabilities of WPF can be alternatively realized in WebSphere eXtreme Scale.
StreamQuery: A continuous query over in-flight data stored in ObjectGrid maps.	None
Static grid configuration: A static, cluster-based topology using the cluster deployment XML file.	Replaced with the improved, dynamic deployment topology for managing large data grids.
Deprecated system properties: System properties to specify the server and client properties files are deprecated.	You can still use these arguments, but change your system properties to the new values. -Dcom.ibm.websphere.objectgrid.CatalogServerProperties The property was deprecated in WebSphere eXtreme Scale Version 7.0. Use the -Dobjectgrid.server.props property. -Dcom.ibm.websphere.objectgrid.ClientProperties The property was deprecated in WebSphere eXtreme Scale Version 7.0. Use the -Dobjectgrid.client.props property. -Dobjectgrid.security.server.prop The property was deprecated in WebSphere eXtreme Scale Version 6.1.0.3. Use the -Dobjectgrid.server.prop property. -serverSecurityFile This argument was deprecated in WebSphere eXtreme Scale Version 6.1.0.3. This option is passed into the startOgServer script. Use the -serverProps argument.

- Removed properties and APIs
  If you are migrating your configuration from an earlier release of WebSphere eXtreme Scale, some features might be removed from this and earlier releases. Use the recommended migration action to determine how to update your configuration.

Related reference:
What's new in Version 8.5
Removed properties and APIs

Removed properties and APIs

If you are migrating your configuration from an earlier release of WebSphere® eXtreme Scale, some features might be removed from this and earlier releases. Use the recommended migration action to determine how to update your configuration.

If a feature is listed as deprecated in Deprecated features, IBM® might remove this capability in a subsequent release of the product. Future investment will be focused on the strategic function listed under Recommended Migration Actions in Deprecated features. Typically, a feature is not removed until at least two major releases or three full years (whichever time period is longer) after the release in which that feature is deprecated. In rare cases, it might become necessary to remove features sooner; such cases are indicated clearly and explicitly in the descriptions of these deprecated features in Deprecated features.
The following information describes removed features, APIs, scripting interfaces, tools, and publicly exposed configuration data. Where possible, the recommended replacement is identified.

8.5+

Removed items in Version 8.5

Table 1. Removed properties and APIs

Removed item	Recommended migration action
Keyword support: Keywords are string tags that can be applied to cache entries and later queried using ObjectMap API methods.	Use the index or query function to get objects with specific attributes.
MapAuthorization interface: This plug-in was used to authorize ObjectMap and JavaMap to access the principals that were represented by a subject object.	Use ObjectGridAuthorization to plug in authorization implementations. An ObjectGridAuthorization can be used to authorize permissions to ObjectGrid, ObjectMap, and JavaMap accesses.
WebSphere partitioning facility (WPF): The partitioning facility is a set of programming APIs that allowed Java™ EE applications to support asymmetric clustering.	You can configure partitioning with WebSphere eXtreme Scale.
StreamQuery: A continuous query over in-flight data stored in ObjectGrid maps.	None

Related reference:
Deprecated properties and APIs
What's new in Version 8.5

Configuring

You can configure WebSphere® eXtreme Scale to run in a stand-alone environment, or you can configure eXtreme Scale to run in an environment with WebSphere Application Server or WebSphere Application Server Network Deployment. For a WebSphere eXtreme Scale deployment to pick up configuration changes on the server side of the data grid, you must restart processes to make these changes take effect rather than being applied dynamically. However, on the client side, although you may not alter the configuration settings for an existing client instance, you can create a new client with the settings you require by using an XML file or doing so programmatically. When creating a client, you can override the default settings that come from the current server configuration.

- Configuration methods
  You can configure most aspects of the product with XML files and property files. You can also use programmatic methods, including application and system programming interfaces, plug-ins, and managed beans.
- Operational checklist
  Use the operational checklist to prepare your environment for deploying WebSphere eXtreme Scale.
- Configuring data grids
  Use an ObjectGrid descriptor XML file to configure data grids, backing maps, plug-ins, and so on. To configure WebSphere eXtreme Scale, use an ObjectGrid descriptor XML file and the ObjectGrid API. For a distributed topology, you need an ObjectGrid descriptor XML file and a deployment policy XML file.
- Configuring deployment policies
  Use the deployment policy descriptor XML file and the objectgrid descriptor XML file to manage a distributed topology. The deployment policy is encoded as an XML file that is provided to the container server. The deployment policy provides information about maps, map sets, partitions, replicas, and so on. It also controls shard placement behaviors.
- Configuring catalog and container servers
  WebSphere eXtreme Scale has two types of servers: catalog servers and container servers. Catalog servers control the placement of shards and discover and monitor the container servers. Multiple catalog servers can join a catalog service domain to provide high availability to the environment. A container server is a Java™ virtual machine (JVM) that stores the application data for the data grid.
- Configuring multiple data center topologies
  With the multi-master asynchronous replication, you link a set of catalog service domains. The connected catalog service domains are then synchronized using replication over the links. You can define the links using properties files, at run time with Java Management Extensions (JMX) programs, or with command-line utilities. The set of current links for a domain is stored in the catalog service. You can add and remove links without restarting the catalog service domain that hosts the data grid.
- Configuring ports
  You must open ports to communicate among servers and with remote servers.
- Configuring transports
  Transports enable the exchange of objects and data between different server processes in your configuration.
- Configuring Java clients
  You can configure WebSphere eXtreme Scale to run in a stand-alone environment, or in an environment with WebSphere Application Server. For a WebSphere eXtreme Scale deployment to pick up configuration changes on the server grid side, you must restart processes to make these changes take effect rather than being applied dynamically. However, on the client side, although you cannot alter the configuration settings for an existing client instance, you can create a new client instance with the settings you require by using an XML file or doing so programmatically. When creating a client, you can override the default settings that come from the current server configuration.
- Configuring eXtreme Scale connection factories
  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.
- Configuring cache integration
  WebSphere eXtreme Scale can integrate with other caching-related products. You can also use the WebSphere eXtreme Scale dynamic cache provider to plug WebSphere eXtreme Scale into the dynamic cache component in WebSphere Application Server. Another extension to WebSphere Application Server is the WebSphere eXtreme Scale HTTP session manager, which can help to cache HTTP sessions.
- Configuring database integration
  You can use WebSphere eXtreme Scale to lower the load on databases. You can use a Java Persistence API (JPA) between WebSphere eXtreme Scale and the database to integrate changes as a loader.
- Configuring REST data services
  You can use WebSphere eXtreme Scale REST data service with WebSphere Application Server version 7.0,WebSphere Application Server Community Edition and Apache Tomcat.
- Configuring OSGi-enabled plug-ins using the ObjectGrid descriptor XML file
  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.
- Configuring servers for OSGi
  WebSphere eXtreme Scale includes a server OSGi bundle, allowing starting and configuring servers and containers within an OSGi framework. The configuration topics describe how to use the eXtreme Scale server bundle, OSGi Blueprint service and eXtreme Scale configuration to run eXtreme Scale servers in an Eclipse Equinox OSGi framework.
- 8.5+ Configuring eXtreme Scale servers to run in the Liberty profile
  You must create server and catalog service definitions to run WebSphere eXtreme Scale in the Liberty profile.
- 8.5+ 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.

Configuration methods

You can configure most aspects of the product with XML files and property files. You can also use programmatic methods, including application and system programming interfaces, plug-ins, and managed beans.

About this task

Use the following files to create a basic configuration:

Server properties file
: Use the server properties file to define settings for catalog and container servers, such as trace, logging, security, ports, and so on. You can pass a server properties file to the server start script, put the file in your classpath, or define the file with system properties.

Client properties file
: Use the client properties file to set properties on your clients, including ports and security settings. You can specify the client properties file to use with a system property, by placing the file in the classpath, or by using the ClientClusterContext.getClientProperties method.

ObjectGrid descriptor XML file
: The ObjectGrid descriptor XML file describes the data grid and map configuration. Specify the file to use with the server start script for stand-alone configurations, or add the file to the application module for WebSphere® Application Server configurations.

Deployment policy descriptor XML file
: The deployment policy XML file controls shard and placement of data on the various container servers in the configuration. Specify the file to use with the server start script for stand-alone configurations, or add the file to the application module for WebSphere Application Server configurations.

Related concepts:
Embedded server API
Interacting with an ObjectGrid using the ObjectGridManager interface
Related tasks:
Configuring data grids
Connecting to distributed ObjectGrid instances programmatically
Configuring deployment policies
Related reference:
ObjectGrid descriptor XML file
Deployment policy descriptor XML file
Server properties file
Client properties file
Related information:
ObjectGridManager interface
ClientClusterContext interface
DeploymentPolicy interface

Operational checklist

Use the operational checklist to prepare your environment for deploying WebSphere® eXtreme Scale.

Table 1. Operational checklist

Checklist item	For more information

If you are using AIX®, tune the following operating system settings:
 	
TCP_KEEPINTVL
The TCP_KEEPINTVL setting is part of a socket keep-alive protocol that enables detection of network outage. The property specifies the interval between packets that are sent to validate the connection. When you are using WebSphere eXtreme Scale, set the value to `10`. To check the current setting, run the following command:
`# no -o tcp_keepintvl`
To change the current setting, run the following command:
`# no -o tcp_keepintvl=10`
The TCP_KEEPINTVL setting is in half seconds.
TCP_KEEPINIT
The TCP_KEEPINIT setting is part of a socket keep-alive protocol that enables detection of network outage. The property specifies the initial timeout value for TCP connection. When you are using WebSphere eXtreme Scale, set the value to `40`. To check the current setting, run the following commands:
`# no -o tcp_keepinit`
To change the current setting, run the following command:
`# no -o tcp_keepinit=40`
The TCP_KEEPINIT setting is in half seconds.
	• For AIX tuning information, see Tuning AIX systems.

Checklist item	For more information
Update the orb.properties file to modify the transport behavior of the grid. The orb.properties file is in the java/jre/lib directory.	ORB properties
Use parameters in the **startOgServer** script. In particular, use the following parameters: • Set heap settings with the -jvmArgs parameter. • Set application class path and properties with the -jvmArgs parameter. • Set -jvmArgs parameters for configuring agent monitoring. Port settings WebSphere eXtreme Scale has to open ports for communications for some transports. These ports are all dynamically defined. However, if a firewall is in use between containers then you must specify the ports. Use the following information about the ports: Listener port You can use the -listenerPort argument to specify the port that is used for communication between processes. Core group port You can use the -haManagerPort argument to specify the port that is used for failure detection. This argument is the same as peerPort. Note that core groups do not need to communicate across zones, so you might not need to set this port if the firewall is open to all the members of a single zone. JMX service port You can use the -JMXServicePort argument to specify the port that the JMX service should use. SSL port Passing `-Dcom.ibm.CSI.SSLPort=1234` as a -jvmArgs argument sets the SSL port to `1234`. The SSL port is the secure port peer to the listener port. Client port Used in the catalog service only. You can specify this value with the -catalogServiceEndPoints argument. The format of the value of this parameter is in the format: `serverName:hostName:clientPort:peerPort`	startOgServer script
Verify that security settings are configured correctly: • Transport (SSL) • Application (Authentication and Authorization) To verify your security settings, you can try to use a malicious client to connect to your configuration. For example, when the SSL-Required setting is configured, a client that has a TCP_IP setting with or a client with the wrong trust store should not be able to connect to the server. When authentication is required, a client with no credential, such as a user ID and password, should not be able to connect to the sever. When authorization is enforced, a client with no access authorization should not be granted the access to the server resources.	Security integration with external providers
Choose how you are going to monitor your environment. • **xscmd** tool: ◦ The JMX ports of the catalog servers need to be visible to the**xscmd** tool. The container server ports also need to be accessible for some commands that gather information from the containers. • Monitoring console: With the monitoring console, you can chart current and historical statistics. • Vendor monitoring tools: ◦ Tivoli® Enterprise Monitoring Agent ◦ CA Wily Introscope ◦ Hyperic HQ	• Monitoring with the xscmd utility • Java Management Extensions (JMX) security • Monitoring with the web console • Monitoring with the WebSphere eXtreme Scale Agent for IBM Tivoli Monitoring • Monitoring eXtreme Scale with Hyperic HQ • Monitoring eXtreme Scale applications with CA Wily Introscope

Configuring data grids

Use an ObjectGrid descriptor XML file to configure data grids, backing maps, plug-ins, and so on. To configure WebSphere® eXtreme Scale, use an ObjectGrid descriptor XML file and the ObjectGrid API. For a distributed topology, you need an ObjectGrid descriptor XML file and a deployment policy XML file.

- Configuring local deployments
  A local in-memory eXtreme Scale configuration can be created by using an ObjectGrid descriptor XML file or APIs.
- Configuring evictors with XML files
  In addition to programmatically setting a time-to-live (TTL) evictor with the BackingMap interface, you can use an XML file to configure an evictor on each BackingMap instance.
- Configuring a locking strategy in the ObjectGrid descriptor XML file
  You can define an optimistic, a pessimistic, or no locking strategy on each BackingMap in the WebSphere eXtreme Scale configuration.
- Configuring the lock timeout value in the ObjectGrid descriptor XML file
  The lock timeout value on a BackingMap instance is used to ensure that an application does not wait endlessly for a lock mode to be granted because of a deadlock condition that occurs due to an application error.
- Configuring peer-to-peer replication with JMS
  The Java™ Message Service (JMS) based peer-to-peer replication mechanism is used in both the distributed and local WebSphere eXtreme Scale environment. JMS is a core-to-core replication process and allows data updates to flow among local ObjectGrids and distributed ObjectGrids. For example, with this mechanism you can move data updates from a distributed eXtreme Scale data grid to a local eXtreme Scale grid, or from a grid to another grid in a different system domain.
- Configuring dynamic maps
  You can dynamically create maps that are based on a set of map templates. You can create your own map templates.

Related concepts:
Embedded server API
Interacting with an ObjectGrid using the ObjectGridManager interface
Distributing changes between peer JVMs
JMS event listener
Related tasks:
Configuration methods
Connecting to distributed ObjectGrid instances programmatically
Configuring deployment policies
Related reference:
ObjectGrid descriptor XML file
Deployment policy descriptor XML file
Server properties file
Client properties file
Related information:
ObjectGridManager interface
ClientClusterContext interface
DeploymentPolicy interface

Configuring local deployments

A local in-memory eXtreme Scale configuration can be created by using an ObjectGrid descriptor XML file or APIs.

About this task

To create a local deployment, you create an ObjectGrid descriptor XML file and then pass the file to the createObjectGrid methods in the ObjectGridManager interface.
As an alternative, you can also create the entire deployment programmatically with the ObjectGridManager interface.

Procedure

1. Create an ObjectGrid descriptor XML file.
   The following companyGrid.xml file is an example of an ObjectGrid descriptor XML. The first few lines of the file include the required header for each ObjectGrid XML file. The file defines an ObjectGrid instance named "CompanyGrid" and several BackingMaps named "Customer," "Item," "OrderLine," and "Order."

```
companyGrid.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="CompanyGrid">
                        <backingMap name="Customer" />
                        <backingMap name="Item" />
                        <backingMap name="OrderLine" />
                        <backingMap name="Order" />
                </objectGrid>
        </objectGrids>

</objectGridConfig>
```

2. Pass the XML file to one of the createObjectGrid methods in the ObjectGridManager interface.
   The following code sample validates the companyGrid.xml file against the XML schema, and creates the ObjectGrid instance named "CompanyGrid." The newly created ObjectGrid instance is not cached.

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid companyGrid = objectGridManager.createObjectGrid("CompanyGrid",
        new URL("file:etc/test/companyGrid.xml"), true, false);
```

What to do next

See Creating ObjectGrid instances with the ObjectGridManager interface for more information about defining all of the maps programmatically with the createObjectGrid methods on the ObjectGridManager interface.
Related concepts:
Distributing changes between peer JVMs
JMS event listener
Related reference:
ObjectGrid descriptor XML file
Deployment policy descriptor XML file

Configuring evictors with XML files

In addition to programmatically setting a time-to-live (TTL) evictor with the BackingMap interface, you can use an XML file to configure an evictor on each BackingMap instance.

Before you begin

Before you begin, decide on the type of evictor you are going to use:

- **The default time-based TTL evictor:** The default evictor uses a time-to-live (TTL) eviction policy for each BackingMap instance.
- **A pluggable evictor mechanism:** Pluggable evictors typically use an eviction policy that is based on the number of entries instead of on time.

Set the evictor settings before you start your container servers.

Procedure

- To set the default TTL evictor, add the ttlEvictorType attribute to the ObjectGrid descriptor XML file.
  The following example shows that the map1 BackingMap instance uses a NONE TTL evictor type. The map2 BackingMap instance uses either a LAST_ACCESS_TIME or LAST_UPDATE_TIME TTL evictor type. Specify only one or the other of these settings. The map2 BackingMap instance has a time-to-live value of 1800 seconds, or 30 minutes. The map3 BackingMap instance is defined to use a CREATION_TIME TTL evictor type and has a time-to-live value of 1200 seconds, or 20 minutes.

  Figure 1. Enable TimeToLive evictor with 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="grid1">
        <backingMap name="map1" ttlEvictorType="NONE" />
        <backingMap name="map2" ttlEvictorType="LAST_ACCESS_TIME|LAST_UPDATE_TIME"
                                        timeToLive="1800" />
        <backingMap name="map3" ttlEvictorType="CREATION_TIME"
                                        timeToLive="1200" />
    </objectgrid>
</objectGrids>
```

- To set a pluggable evictor, use the following example.
  Figure 2. Plugging in an evictor using 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="grid">
        <backingMap name="map1" ttlEvictorType="NONE" pluginCollectionRef="LRU" />
        <backingMap name="map2" ttlEvictorType="NONE" pluginCollectionRef="LFU" />
    </objectGrid>
</objectGrids>
<backingMapPluginCollections>
    <backingMapPlugincollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsvictenablexml_LRU">
        <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsvictenablexml_Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor">
            <property name="maxSize" type="int" value="1000" description="set max size
                                                    for each LRU queue" />
            <property name="sleepTime" type="int" value="15" description="evictor
                                                    thread sleep time" />
            <property name="numberOfLRUQueues" type="int" value="53" description="set number
                                                    of LRU queues" />
        </bean>
```

```
    </backingMapPluginCollection>
    <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsvictenablexml_LFU">
        <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsvictenablexml_Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LFUEvictor">
            <property name="maxSize" type="int" value="2000" description="set max size for each LFU heap" />
            <property name="sleepTime" type="int" value="15" description="evictor thread sleep time" />
            <property name="numberOfHeaps" type="int" value="211" description="set number of LFU heaps" />
        </bean>
    </backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

Related concepts:
Distributing changes between peer JVMs
JMS event listener
Evictors
Evictors
Tuning evictors
Plug-ins for evicting cache objects
Custom evictors
Related tasks:
Enabling evictors programmatically
Related reference:
ObjectGrid descriptor XML file
ObjectGrid descriptor XML file

Configuring a locking strategy in the ObjectGrid descriptor XML file

You can define an optimistic, a pessimistic, or no locking strategy on each BackingMap in the WebSphere® eXtreme Scale configuration.

Before you begin

Decide which locking strategy you want to use. For more information, see Locking strategies.

About this task

You can configure each BackingMap instance to use one of the following locking strategies:

- Optimistic locking mode (default)
- Pessimistic locking mode
- None

Procedure

- **Configure a pessimistic locking strategy**
  - With the lockStrategy attribute in the ObjectGrid descriptor 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="test">
            <backingMap name="pessimisticMap"
                lockStrategy="PESSIMISTIC"/>
        </objectGrid>
    </objectGrids>
</objectGridConfig>
```

- **Configure an optimistic locking strategy**
  - With the lockStrategy attribute in the ObjectGrid descriptor 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="test">
            <backingMap name="optimisticMap"
                lockStrategy="OPTIMISTIC"/>
        </objectGrid>
    </objectGrids>
</objectGridConfig>
```

- **Configure a no locking strategy**

- With the lockStrategy attribute in the ObjectGrid descriptor 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="test">
            <backingMap name="noLockingMap"
                lockStrategy="NONE"/>
        </objectGrid>
    </objectGrids>
</objectGridConfig>
```

Related concepts:
Distributing changes between peer JVMs
JMS event listener
Locking strategies
Related tasks:
Configuring and implementing locking in Java applications
Configuring the lock timeout value in the ObjectGrid descriptor XML file
Related reference:
ObjectGrid descriptor XML file

Configuring peer-to-peer replication with JMS

The Java™ Message Service (JMS) based peer-to-peer replication mechanism is used in both the distributed and local WebSphere® eXtreme Scale environment. JMS is a core-to-core replication process and allows data updates to flow among local ObjectGrids and distributed ObjectGrids. For example, with this mechanism you can move data updates from a distributed eXtreme Scale data grid to a local eXtreme Scale grid, or from a grid to another grid in a different system domain.

Before you begin

The JMS-based peer-to-peer replication mechanism is based on the built-in JMS-based ObjectGridEventListener, com.ibm.websphere.objectgrid.plugins.builtins.JMSObjectGridEventListener. For detailed information regarding enabling peer-to-peer replication mechanism, see JMS event listener.

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

The following is an XML configuration example to enable a peer-to-peer replication mechanism on an eXtreme Scale configuration:

```
peer-to-peer replication configuration - XML example
<bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txspeerrepl_ObjectGridEventListener"
className="com.ibm.websphere.objectgrid.plugins.JMSObjectGridEventListener">
        <property name="replicationRole" type="java.lang.String" value="DUAL_ROLES" description="" />
  <property name="replicationStrategy" type="java.lang.String" value="PUSH" description="" />
  <property name="jms_topicConnectionFactoryJndiName" type="java.lang.String"
              value="defaultTCF" description="" />
  <property name="jms_topicJndiName" type="java.lang.String" value="defaultTopic" description="" />
  <property name="jms_topicName" type="java.lang.String" value="defaultTopic" description="" />
  <property name="jms_userid" type="java.lang.String" value="" description="" />
  <property name="jms_password" type="java.lang.String" value="" description="" />
  <property name="jndi_properties" type="java.lang.String"
        value="java.naming.factory.initial=org.apache.activemq.jndi.ActiveMQInitialContextFactory;
        java.naming.provider.url=tcp://localhost:61616;connectionFactoryNames=defaultTCF;
        topic.defaultTopic=defaultTopic"
        description="jndi properties" />
     </bean>
```

- Distributing changes between peer JVMs
  The LogSequence and LogElement objects distribute changes between peer JVMs and communicate the changes that have occurred in an eXtreme Scale transaction with an ObjectGridEventListener plug-in.
- JMS event listener
  The JMSObjectGridEventListener is designed to support client-side near cache invalidation and a peer-to-peer replication mechanism. It is a Java Message Service (JMS) implementation of the ObjectGridEventListener interface.

Related concepts:
Distributing changes between peer JVMs
JMS event listener
Related reference:
ObjectGrid descriptor XML file

Distributing changes between peer JVMs

The LogSequence and LogElement objects distribute changes between peer JVMs and communicate the changes that have occurred in an eXtreme Scale transaction with an ObjectGridEventListener plug-in.

For more information about how Java™ Message Service (JMS) can be used to distribute transactional changes, see JMS for distributed transaction changes.

A prerequisite is that the ObjectGrid instance must be cached by the ObjectGridManager. See createObjectGrid methods for more information. The cacheInstance boolean value must be set to true.

It is not necessary for you to implement this mechanism. There is a built-in peer-to-peer replication mechanism for you to use this function. See Configuring peer-to-peer replication with JMS.

The objects provide a means for an application to easily publish changes that have occurred in an ObjectGrid using a message transport to peer ObjectGrids in remote Java virtual machines and then apply those changes on that JVM. The LogSequenceTransformer class is critical to enabling this support. This article examines how to write a listener using a Java Message Service (JMS) messaging system for propagating the messages. To that end, eXtreme Scale supports transmitting LogSequences that result from an eXtreme Scale transaction commit across WebSphere Application Server cluster members with an IBM-provided plug-in. This function is not enabled by default, but can be configured to be operational. However, when either the consumer or producer is not a WebSphere Application Server, using an external JMS messaging system might be required.

Implementing the mechanism

The LogSequenceTransformer class, and the ObjectGridEventListener, LogSequence and LogElement APIs allow any reliable publish-and-subscribe to be used to distribute the changes and filter the maps you want to distribute. The snippets in this topic show how to use these APIs with JMS to build a peer-to-peer ObjectGrid shared by applications that are hosted on a diverse set of platforms sharing a common message transport.

Initialize the plug-in

The ObjectGrid calls the initialize method of the plug-in, part of the ObjectGridEventListener interface contract, when the ObjectGrid starts. The initialize method must obtain its JMS resources, including connections, sessions, and publishers, and start the thread that is the JMS listener.

The following examples show the initialize method:

```
initialize method example
public void initialize(Session session) {
        mySession = session;
        myGrid = session.getObjectGrid();
        try {
            if (mode == null) {
                throw new ObjectGridRuntimeException("No mode specified");
            }
            if (userid != null) {
                connection = topicConnectionFactory.createTopicConnection(userid,
                                                                  password);
            } else
                connection = topicConnectionFactory.createTopicConnection();

            // need to start the connection to receive messages.
            connection.start();

            // the jms session is not transactional (false).
            jmsSession = connection.createTopicSession(false,
                                                       javax.jms.Session.AUTO_ACKNOWLEDGE);
            if (topic == null)
                if (topicName == null) {
                    throw new ObjectGridRuntimeException("Topic not specified");
                } else {
                    topic = jmsSession.createTopic(topicName);
                }
            publisher = jmsSession.createPublisher(topic);
            // start the listener thread.
            listenerRunning = true;
            listenerThread = new Thread(this);
            listenerThread.start();
        } catch (Throwable e) {
            throw new ObjectGridRuntimeException("Cannot initialize", e);
        }
    }
```

The code to start the thread uses a Java 2 Platform, Standard Edition (Java SE) thread. If you are running a WebSphere Application Server Version 6.x or a WebSphere Application Server Version 5.x Enterprise server, use the asynchronous bean application programming interface (API) to start this daemon thread. You can also use the common APIs. Following is an example replacement snippet showing the same action using a work manager:

```
// start the listener thread.
listenerRunning = true;
workManager.startWork(this, true);
```

The plug-in must also implement the Work interface instead of the Runnable interface. You also need to add a release method to set the listenerRunning variable to false. The plug-in must be provided with a WorkManager instance in its constructor or by injection if using an Inversion of Control (IoC) container.

Transmit the changes

The following is a sample transactionEnd method for publishing the local changes that are made to an ObjectGrid. This sample uses JMS, although you can use any message transport that is capable of reliable publish-and subscribe-messaging.

```
transactionEnd method example
// This method is synchronized to make sure the
```

```
    // messages are published in the order the transaction
    // were committed. If we started publishing the messages
    // in parallel then the receivers could corrupt the Map
    // as deletes may arrive before inserts etc.
    public synchronized void transactionEnd(String txid, boolean isWriteThroughEnabled,
                               boolean committed,
            Collection changes) {
        try {
            // must be write through and commited.
            if (isWriteThroughEnabled && committed) {
                // write the sequences to a byte []
                ByteArrayOutputStream bos = new ByteArrayOutputStream();
                ObjectOutputStream oos = new ObjectOutputStream(bos);
                if (publishMaps.isEmpty()) {
                    // serialize the whole collection
                    LogSequenceTransformer.serialize(changes, oos, this, mode);
                } else {
                    // filter LogSequences based on publishMaps contents
                    Collection publishChanges = new ArrayList();
                    Iterator iter = changes.iterator();
                    while (iter.hasNext()) {
                        LogSequence ls = (LogSequence) iter.next();
                        if (publishMaps.contains(ls.getMapName())) {
                            publishChanges.add(ls);
                        }
                    }
                    LogSequenceTransformer.serialize(publishChanges, oos, this, mode);
                }
                // make an object message for the changes
                oos.flush();
                ObjectMessage om = jmsSession.createObjectMessage(bos.toByteArray());
                // set properties
                om.setStringProperty(PROP_TX, txid);
                om.setStringProperty(PROP_GRIDNAME, myGrid.getName());
                // transmit it.
                publisher.publish(om);
            }
        } catch (Throwable e) {
            throw new ObjectGridRuntimeException("Cannot push changes", e);
        }
    }
```

This method uses several instance variables:

- `jmsSession` variable: A JMS session that is used to publish messages. It is created when the plug-in initializes.
- `mode` variable: The distribution mode.
- `publishMaps` variable: A set that contains the name of each map with changes to publish. If the variable is empty, then all the maps are published.
- `publisher` variable: A TopicPublisher object that is created during the plug-in initialize method

Receive and apply update messages

Following is the run method. This method runs in a loop until the application stops the loop. Each loop iteration attempts to receive a JMS message and apply it to the ObjectGrid.

```
JMS message run method example
private synchronized boolean isListenerRunning() {
        return listenerRunning;
    }

    public void run() {
        try {
            System.out.println("Listener starting");
            // get a jms session for receiving the messages.
            // Non transactional.
            TopicSession myTopicSession;
            myTopicSession = connection.createTopicSession(false, javax.jms.
                                                    Session.AUTO_ACKNOWLEDGE);

            // get a subscriber for the topic, true indicates don't receive
            // messages transmitted using publishers
            // on this connection. Otherwise, we'd receive our own updates.
            TopicSubscriber subscriber = myTopicSession.createSubscriber(topic,
                                                    null, true);
            System.out.println("Listener started");
            while (isListenerRunning()) {
                ObjectMessage om = (ObjectMessage) subscriber.receive(2000);
                if (om != null) {
                    // Use Session that was passed in on the initialize...
                    // very important to use no write through here
                    mySession.beginNoWriteThrough();
                    byte[] raw = (byte[]) om.getObject();
                    ByteArrayInputStream bis = new ByteArrayInputStream(raw);
                    ObjectInputStream ois = new ObjectInputStream(bis);
                    // inflate the LogSequences
                    Collection collection = LogSequenceTransformer.inflate(ois,
                                                                                    myGrid);
                    Iterator iter = collection.iterator();
                    while (iter.hasNext()) {
                        // process each Maps changes according to the mode when
```

```
                    // the LogSequence was serialized
                    LogSequence seq = (LogSequence) iter.next();
                    mySession.processLogSequence(seq);
                }
                mySession.commit();
            } // if there was a message
        } // while loop
        // stop the connection
        connection.close();
    } catch (IOException e) {
        System.out.println("IO Exception: " + e);
    } catch (JMSException e) {
        System.out.println("JMS Exception: " + e);
    } catch (ObjectGridException e) {
        System.out.println("ObjectGrid exception: " + e);
        System.out.println("Caused by: " + e.getCause());
    } catch (Throwable e) {
        System.out.println("Exception : " + e);
    }
    System.out.println("Listener stopped");
}
```

Related tasks:
Configuring data grids
Configuring local deployments
Configuring a locking strategy in the ObjectGrid descriptor XML file
Configuring peer-to-peer replication with JMS
Configuring evictors with XML files
Related reference:
ObjectGrid descriptor XML file

JMS event listener

The JMSObjectGridEventListener is designed to support client-side near cache invalidation and a peer-to-peer replication mechanism. It is a Java™ Message Service (JMS) implementation of the ObjectGridEventListener interface.

The client invalidation mechanism can be used in a distributed eXtreme Scale environment to ensure client near cache data to be synchronized with servers or other clients. Without this function, the client near cache could hold stale data. However, even with this JMS-based client invalidation mechanism, you have to take into consideration the timing window for updating a client near cache because of the delay for the run time in publishing updates.

The peer-to-peer replication mechanism can be used in both distributed and local eXtreme Scale environments. It is an ObjectGrid core-to-core replication process and allows data updates to flow among local ObjectGrids and distributed ObjectGrids. For example, with this mechanism you can move data updates from a distributed grid to a local ObjectGrid, or from any grid to another grid in a different system domain.

The JMSObjectGridEventListener requires the user to configure JMS and Java Naming and Directory Interface (JNDI) information in order to obtain required JMS resources. Additionally, replication-related properties must be set correctly. In a JEE environment, the JNDI should be available in both Web and Enterprise JavaBean (EJB) containers. In this case, the JNDI property is optional unless you want to obtained external JMS resources.

This event listener has properties you can configure with XML or programmatic approaches, which can be used for only client invalidation, only peer-to-peer replication, or both. Most properties are optional for customizing the behavior to achieve your required functionality.

For more information, see JMSObjectGridEventListener API.

For more information see the JMSObjectGridEventListener API.

Extending the JMSObjectGridEventListener plug-in

The JMSObjectGridEventListener plug-in allows peer ObjectGrid instances to receive updates when data in the grid has been changed or evicted. It also allows clients to be notified when entries are updated or evicted from an eXtreme Scale grid. This topic describes how to extend the JMSObjectGridEventListener plug-in to allow applications to be notified when a JMS message is received. This is most useful when using the CLIENT_SERVER_MODEL setting for client invalidation.

When running in the receiver role, the overridden JMSObjectGridEventListener.onMessage method is automatically called by the eXtreme Scale runtime when the JMSObjectGridEventListener instance receives JMS message updates from the grid. These messages wrap a collection of LogSequence. Objects. The LogSequence objects are passed to the onMessage method and the application uses the LogSequence to identify which cache entries have been inserted, deleted, updated or invalidated.

To use the onMessage extension point, applications perform the following steps.

1. Create a new class, extending the JMSObjectGridEventListener class, overriding the onMessage method.
2. Configure the extended JMSObjectGridEventListener the same way as the ObjectGridEventListener for ObjectGrid.

The extended JMSObjectGridEventListener class is a child class of the JMSObjectGridEventListener class and can only override two methods: the initialize (optional) and onMessage methods. If a child class of the JMSObjectGridEventListener class needs to use any ObjectGrid artifacts such as ObjectGrid or Session in the onMessage method, it can get these artifacts in the initialize method and cache them as instance variables. Also, in the onMessage method, cached ObjectGrid artifacts can be used to process a passed collection of LogSequences.

Note: The overridden initialize method has to invoke super.initialize method in order to initialize parent JMSObjectGridEventListener appropriately.
The following is a sample for an extended JMSObjectGridEventListener class.

```
package com.ibm.websphere.samples.objectgrid.jms.price;

import java.util.*;
import com.ibm.websphere.objectgrid.*;
import com.ibm.websphere.objectgrid.plugins.LogElement;
import com.ibm.websphere.objectgrid.plugins.LogSequence;
import com.ibm.websphere.objectgrid.plugins.builtins.JMSObjectGridEventListener;

public class ExtendedJMSObjectGridEventListener extends JMSObjectGridEventListener{
        protected static boolean debug = true;

    /**
     * This is the grid associated with this listener.
     */
    ObjectGrid grid;

    /**
     * This is the session associated with this listener.
     */
    Session session;

    String objectGridType;

    public List receivedLogSequenceList = new ArrayList();


        /* (non-Javadoc)
         * @see com.ibm.websphere.objectgrid.plugins.builtins.JMSObjectGridEventListener
                                #initialize(com.ibm.websphere.objectgrid.Session)
         */
        public void initialize(Session session) {
                // Note: if need to use any ObjectGrid artifact, this class need to get ObjectGrid
                //       from the passed Session instance and get ObjectMap from session instance
                // for any transactional ObjectGrid map operation.

                super.initialize(session);  // must invoke super's initialize method.
                this.session = session; // cache the session instance, in case need to
                //                      use it to perform map operation.
                this.grid = session.getObjectGrid(); // get ObjectGrid, in case need
                //                      to get ObjectGrid information.

                if (grid.getObjectGridType() == ObjectGrid.CLIENT)
                        objectGridType = "CLIENT";
                else if (grid.getObjectGridType() == ObjectGrid.SERVER)
                        objectGridType = "Server";

                if (debug)
                        System.out.println("ExtendedJMSObjectGridEventListener[" +
                                        objectGridType + "].initialize() : grid = "     + this.grid);
        }

        /* (non-Javadoc)
         * @see com.ibm.websphere.objectgrid.plugins.builtins.JMSObjectGridEventListener
                                #onMessage(java.util.Collection)
         */
        protected void onMessage(Collection logSequences) {
                System.out.println("ExtendedJMSObjectGridEventListener[" +
                                objectGridType + "].onMessage(): ");

                Iterator iter = logSequences.iterator();

                while (iter.hasNext()) {
            LogSequence seq = (LogSequence) iter.next();

                StringBuffer buffer = new StringBuffer();
                String mapName = seq.getMapName();
                int size = seq.size();
                buffer.append("\nLogSequence[mapName=" + mapName + ", size=" + size + ",
                                        objectGridType=" + objectGridType
                                + "]: ");

                Iterator logElementIter = seq.getAllChanges();
                for (int i = seq.size() - 1; i >= 0; --i) {
                        LogElement le = (LogElement) logElementIter.next();
                        buffer.append(le.getType() + " -> key=" + le.getCacheEntry().getKey() + ", ");
                }
                buffer.append("\n");

                receivedLogSequenceList.add(buffer.toString());

                if (debug) {
                        System.out.println("ExtendedJMSObjectGridEventListener["
                                                + objectGridType + "].onMessage(): " + buffer.toString());
                }
                }
        }

        public String dumpReceivedLogSequenceList() {
                String result = "";
                int size = receivedLogSequenceList.size();
                result = result + "\nExtendedJMSObjectGridEventListener[" + objectGridType
```

```
                                + "]: receivedLogSequenceList size = " + size + "\n";
                for (int i = 0; i < size; i++) {
                        result = result + receivedLogSequenceList.get(i) + "\n";
                }
                return result;
        }

        public String toString() {
                return "ExtendedJMSObjectGridEventListener["
                                + objectGridType + " - " + this.grid + "]";
        }
}
```

Configuration

The extended JMSObjectGridEventListener class must be configured the same way for both client invalidation and peer-to-peer replication mechanism. The following is the XML configuration example.

```
<objectGrid name="PRICEGRID">
                        <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxsjmsevntls_ObjectGridEventListener"
                                className="com.ibm.websphere.samples.objectgrid.jms.
                                        price.ExtendedJMSObjectGridEventListener">
                                <property name="invalidationModel" type="java.lang.String"
                                        value="CLIENT_SERVER_MODEL" description="" />
                                <property name="invalidationStrategy" type="java.lang.String"
                                        value="INVALIDATE" description="" />
                                <property name="jms_topicConnectionFactoryJndiName" type="java.lang.String"
                                        value="jms/TCF" description="" />
                                <property name="jms_topicJndiName" type="java.lang.String"
                                        value="GRID.PRICEGRID" description="" />
                                <property name="jms_topicName" type="java.lang.String"
                                        value="GRID.PRICEGRID" description="" />
                                <property name="jms_userid" type="java.lang.String" value=""
                                        description="" />
                                <property name="jms_password" type="java.lang.String" value=""
                                        description="" />
                        </bean>
                        <backingMap name="PRICE" pluginCollectionRef="PRICE"></backingMap>
      </objectGrid>
```

Note: The className of ObjectGridEventListener bean is configured with the extended JMSObjectGridEventListener class with the same properties as the generic JMSObjectGridEventListener.

Related concepts:
Data invalidation
Related tasks:
8.5+ Querying and invalidating data
Configuring data grids
Configuring local deployments
Configuring a locking strategy in the ObjectGrid descriptor XML file
Configuring peer-to-peer replication with JMS
Configuring evictors with XML files
Related reference:
ObjectGridEventListener plug-in
Introduction to ObjectMap
ObjectGrid descriptor XML file
Related information:
ObjectMap.invalidate method
EntityManager.invalidate method
ObjectGridEventListener interface

Configuring deployment policies

Use the deployment policy descriptor XML file and the objectgrid descriptor XML file to manage a distributed topology. The deployment policy is encoded as an XML file that is provided to the container server. The deployment policy provides information about maps, map sets, partitions, replicas, and so on. It also controls shard placement behaviors.

- Configuring distributed deployments
  Use the deployment policy descriptor XML file and the ObjectGrid descriptor XML file to manage your topology.
- Controlling shard placement with zones
  Use your deployment policy to define zones. Zones give you control over shard placement in WebSphere® eXtreme Scale. Zones are a logical, user-defined concept used to represent logical groupings of physical servers.

Related concepts:
Embedded server API
Interacting with an ObjectGrid using the ObjectGridManager interface
Related tasks:
Configuration methods
Configuring data grids

Connecting to distributed ObjectGrid instances programmatically
Related reference:
ObjectGrid descriptor XML file
Deployment policy descriptor XML file
Server properties file
Client properties file
Related information:
ObjectGridManager interface
ClientClusterContext interface
DeploymentPolicy interface

Configuring distributed deployments

Use the deployment policy descriptor XML file and the ObjectGrid descriptor XML file to manage your topology.

The deployment policy is encoded as an XML file that is provided to the eXtreme Scale container server. The XML file specifies the following information:

- The maps that belong to each map set
- The number of partitions
- The number of synchronous and asynchronous replicas

The deployment policy also controls the following placement behaviors.

- The minimum number of active container servers before placement occurs
- Automatic replacement of lost shards
- Placement of each shard from a single partition onto a different machine

Endpoint information is not pre-configured in the dynamic environment. There are no server names or physical topology information found in the deployment policy. All shards in a data grid are automatically placed into container servers by the catalog service. The catalog service uses the constraints that are defined by the deployment policy to automatically manage shard placement. This automatic shard placement leads to easy configuration for large data grids. You can also add servers to your environment as needed.

Restriction: In a WebSphere® Application Server environment, a core group size of more than 50 members is not supported.
A deployment policy XML file is passed to a container server during startup. A deployment policy must be used along with an ObjectGrid XML file. The deployment policy is not required to start a container server, but is recommended. The deployment policy must be compatible with the ObjectGrid XML file that is used with it. For each objectgridDeployment element in the deployment policy, you must include a corresponding objectGrid element in your ObjectGrid XML file. The maps in the objectgridDeployment must be consistent with the backingMap elements found in the ObjectGrid XML. Every backingMap must be referenced within only one mapSet element.

In the following example, the companyGridDpReplication.xml file is intended to be paired with the corresponding companyGrid.xml file.

```
companyGridDpReplication.xml
<?xml version="1.0" encoding="UTF-8"?>
<deploymentPolicy xmlns:xsi="http://www.w3.org./2001/XMLSchema-instance"
        xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy ../deploymentPolicy.xsd"
        xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">

        <objectgridDeployment objectgridName="CompanyGrid">
                <mapSet name="mapSet1" numberOfPartitions="11"
                        minSyncReplicas="1" maxSyncReplicas="1"
                        maxAsyncReplicas="0" numInitialContainers="4">
                        <map ref="Customer" />
                        <map ref="Item" />
                        <map ref="OrderLine" />
                        <map ref="Order" />
                </mapSet>
        </objectgridDeployment>

</deploymentPolicy>

companyGrid.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="CompanyGrid">
                        <backingMap name="Customer" />
                        <backingMap name="Item" />
                        <backingMap name="OrderLine" />
                        <backingMap name="Order" />
                </objectGrid>
        </objectGrids>

</objectGridConfig>
```

The companyGridDpReplication.xml file has one mapSet element that is divided into 11 partitions. Each partition must have exactly one synchronous replica. The number of synchronous replicas is specified by the minSyncReplicas and maxSyncReplicas attributes. Because the minSyncReplicas attribute is set to `1`, each partition in the mapSet element must have at least one synchronous replica available to process write transactions. Because the maxSyncReplicas attribute is set to `1`, each partition cannot exceed one synchronous replica. The partitions in this mapSet element have no asynchronous replicas.

The numInitialContainers attribute instructs the catalog service to defer placement until four container servers are available to support this ObjectGrid instance. The numInitialContainers attribute is ignored after the specified number of container servers has been reached.

You can also use the placementDeferralInterval property and **xscmd -c suspendBalancing** command to delay the placement of shards on the container servers.

Although the companyGridDpReplication.xml file is a basic example, a deployment policy can offer you full control over your environment.

Distributed topology

Distributed coherent caches offer increased performance, availability, and scalability, which you can configure.
WebSphere eXtreme Scale automatically balances servers. You can include additional servers without restarting WebSphere eXtreme Scale. Adding additional servers without having to restart eXtreme Scale allows you to have simple deployments and also large, terabyte-sized deployments in which thousands of servers are needed.

This deployment topology is flexible. Using the catalog service, you can add and remove servers to better use resources without removing the entire cache. You can use the **startOgServer** and **stopOgServer** commands to start and stop container servers. Both of these commands require you to specify the -catalogServiceEndPoints option. All distributed topology clients communicate to the catalog service through the Internet Interoperability Object Protocol (IIOP). All clients use the ObjectGrid interface to communicate with servers.

The dynamic configuration capability of WebSphere eXtreme Scale makes it easy to add resources to the system. Containers host the data and the catalog service allows clients to communicate with the grid of container servers. The catalog service forwards requests, allocates space in host container servers, and manages the health and availability of the overall system. Clients connect to a catalog service, retrieve a description of the container server topology, and then communicate directly to each server as needed. When the server topology changes due to the addition of new servers, or due to the failure of others, the catalog service automatically routes client requests to the appropriate server that hosts the data.

A catalog service typically exists in its own grid of Java™ virtual machines. A single catalog server can manage multiple servers. You can start a container server in a JVM by itself or load the container server into an arbitrary JVM with other container servers for different servers. A client can exist in any JVM and communicate with one or more servers. A client can also exist in the same JVM as a container server.

You can also create a deployment policy programmatically when you are embedding a container server in an existing Java process or application. For more information, see the DeploymentPolicy API documentation.

Related tasks:
Starting container servers
Configuring WebSphere Application Server applications to automatically start container servers
Controlling placement

Controlling shard placement with zones

Use your deployment policy to define zones. Zones give you control over shard placement in WebSphere® eXtreme Scale. Zones are a logical, user-defined concept used to represent logical groupings of physical servers.

- Zones for replica placement
  With zones, you can place replicas across data centers. A zone can be defined as different floors of a building, different buildings, or even different cities or other distinctions as configured with zone rules. With this capability, data grids of thousands of partitions can be managed with optional placement rules.
- Zone-preferred routing
  With zone-preferred routing, you can define how WebSphere eXtreme Scale directs transactions to zones.
- Defining zones for container servers
  Zones are collections of container servers. A container server can belong only one zone. A container server is assigned to a zone when it starts.
- Example: Zone definitions in the deployment policy descriptor XML file
  You can specify zones and zone rules with the deployment policy descriptor XML file.
- Viewing zone information with the xscmd utility
  You can use the **xscmd** utility to view information about your current zone deployment, including shard placement data.

Related concepts:
Zones
Zone-preferred routing
Related reference:
Example: Zone definitions in the deployment policy descriptor XML file
Deployment policy descriptor XML file

Zones for replica placement

With zones, you can place replicas across data centers. A zone can be defined as different floors of a building, different buildings, or even different cities or other distinctions as configured with zone rules. With this capability, data grids of thousands of partitions can be managed with optional placement rules.

Zone rules

An eXtreme Scale partition has one primary shard and zero or more replica shards. For this example, consider the following naming convention for these shards. `P` is the primary shard, `S` is a synchronous replica and `A` is an asynchronous replica. A zone rule has three components:

- A rule name
- A list of zones
- An inclusive or exclusive flag

For more information about defining a zone name for a container server, see Defining zones for container servers. A zone rule specifies the possible set of zones in which a shard can be placed. The inclusive flag indicates that after a shard is placed in a zone from the list, then all other shards are also placed in that zone. An exclusive setting indicates that each shard for a partition is placed in a different zone in the zone list. For example, using an exclusive setting means that if there are three shards (primary, and two synchronous replicas), then the zone list must have three zones.
Each shard can be associated with one zone rule. A zone rule can be shared between two shards. When a rule is shared then the inclusive or exclusive flag extends across shards of all types sharing a single rule.

Examples

A set of examples showing various scenarios and the deployment configuration to implement the scenarios follows.

Striping primaries and replicas across zones

You have three blade chassis, and want primaries that are distributed across all three, with a single synchronous replica placed in a different chassis than the primary. Define each chassis as a zone with chassis names `ALPHA`, `BETA`, and `GAMMA`. An example deployment XML follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<deploymentPolicy xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
        xsi:schemaLocation=
        "http://ibm.com/ws/objectgrid/deploymentPolicy ../deploymentPolicy.xsd"
                        xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">
                <objectgridDeployment objectgridName="library">
                        <mapSet name="ms1" numberOfPartitions="37" minSyncReplicas="1"
                                maxSyncReplicas="1" maxAsyncReplicas="0">
                        <map ref="book" />
                        <zoneMetadata>
                                <shardMapping shard="P" zoneRuleRef="stripeZone"/>
                                <shardMapping shard="S" zoneRuleRef="stripeZone"/>
                                <zoneRule name ="stripeZone" exclusivePlacement="true" >
                                        <zone name="ALPHA" />
                                        <zone name="BETA" />
                                        <zone name="GAMMA" />
                                </zoneRule>
                        </zoneMetadata>
                </mapSet>
        </objectgridDeployment>
</deploymentPolicy>
```

This deployment XML contains a grid called library with a single Map called book. It uses four partitions with a single synchronous replica. The zone metadata clause shows the definition of a single zone rule and the association of zone rules with shards. The primary and synchronous shards are both associated with the zone rule "stripeZone". The zone rule has all three zones in it and it uses exclusive placement. This rule means that if the primary for partition 0 is placed in `ALPHA` then the replica for partition 0 is placed in either `BETA` or `GAMMA`. Similarly, primaries for other partitions are placed in other zones and the replicas are placed in another zone.

Asynchronous replica in a different zone than primary and synchronous replica

In this example, two buildings exist with a high latency connection between them. You want no data loss high availability for all scenarios. However, the performance impact of synchronous replication between buildings leads you to a trade-off. You want a primary with synchronous replica in one building and an asynchronous replica in the other building. Normally, the failures are JVM crashes or computer failures rather than large-scale issues. With this topology, you can survive normal failures with no data loss. The loss of a building is rare enough that some data loss is acceptable in that case. You can make two zones, one for each building. The deployment XML file follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy ../deploymentPolicy.xsd"
                xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">

        <objectgridDeployment objectgridName="library">
                <mapSet name="ms1" numberOfPartitions="13" minSyncReplicas="1"
                        maxSyncReplicas="1" maxAsyncReplicas="1">
                        <map ref="book" />
                        <zoneMetadata>
                                <shardMapping shard="P" zoneRuleRef="primarySync"/>
                                <shardMapping shard="S" zoneRuleRef="primarySync"/>
                                <shardMapping shard="A" zoneRuleRef="aysnc"/>
                                <zoneRule name ="primarySync" exclusivePlacement="false" >
                                        <zone name="BldA" />
                                        <zone name="BldB" />
                                </zoneRule>
                                <zoneRule name="aysnc" exclusivePlacement="true">
                                        <zone name="BldA" />
                                        <zone name="BldB" />
                                </zoneRule>
                        </zoneMetadata>
                </mapSet>
        </objectgridDeployment>
</deploymentPolicy>
```

The primary and synchronous replica share a primaySync zone rule with an exclusive flag setting of false. So, after the primary or sync gets placed in a zone, then the other is also placed in the same zone. The asynchronous replica uses a second zone rule with the same zones as the primarySync zone rule but it uses the

exclusivePlacement attribute set to true. This attribute indicates that means a shard cannot be placed in a zone with another shard from the same partition. As a result, the asynchronous replica does not get placed in the same zone as the primary or synchronous replicas.

Placing all primaries in one zone and all replicas in another zone

Here, all primaries are in one specific zone and all replicas in a different zone, a primary and a single asynchronous replica. All replicas are in zone A and primaries in B.

```
<?xml version="1.0" encoding="UTF-8"?>

<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation=
                "http://ibm.com/ws/objectgrid/deploymentPolicy ../deploymentPolicy.xsd"
        xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">

        <objectgridDeployment objectgridName="library">
                <mapSet name="ms1" numberOfPartitions="13" minSyncReplicas="0"
                        maxSyncReplicas="0" maxAsyncReplicas="1">
                        <map ref="book" />
                        <zoneMetadata>
                                <shardMapping shard="P" zoneRuleRef="primaryRule"/>
                                <shardMapping shard="A" zoneRuleRef="replicaRule"/>
                                <zoneRule name ="primaryRule">
                                        <zone name="A" />
                                </zoneRule>
                                <zoneRule name="replicaRule">
                                        <zone name="B" />
                                                </zoneRule>
                                        </zoneMetadata>
                                </mapSet>
                </objectgridDeployment>
</deploymentPolicy>
```

Here, you can see two rules, one for the primaries (P) and another for the replica (A).

Zones over wide area networks (WAN)

You might want to deploy a single data grid over multiple buildings or data centers with slower network interconnections. 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. eXtreme Scale approaches this harsh environment by limiting heartbeating between zones.

Java™ virtual machines grouped into core groups do heartbeat each other. When the catalog service organizes Java virtual machines into core groups, those groups do not span zones. A leader within that group pushes membership information to the catalog service. The catalog service verifies any reported failures before taking action. It does this by attempting to connect to the suspect Java virtual machines. If the catalog service sees a false failure detection, then it takes no action as the core group partition heals in a short time.

The catalog service also heartbeats core group leaders periodically at a slow rate to handle the case of core group isolation.

Zone-preferred routing

With zone-preferred routing, you can define how WebSphere® eXtreme Scale directs transactions to zones.

You have control over where the shards of a data grid are placed. See Zones for replica placement to get more information about some basic scenarios and how to configure your deployment policy accordingly.

Zone-preferred routing givesWebSphere eXtreme Scale clients the capability to specify a preference for a particular zone or set of zones. As a result, client transactions are routed to preferred zones before attempting to route to any other zone.

Requirements for zone-preferred routing

Before attempting zone-preferred routing, ensure that the application is able to satisfy the requirements of your scenario.

Per-container partition placement is necessary to use zone-preferred routing. This placement strategy is a good fit for applications that are storing session data in the ObjectGrid. The default partition placement strategy for WebSphere eXtreme Scale is `fixed-partition`. Keys are hashed at transaction commit time to determine which partition houses the key-value pair of the map when using fixed-partition placement.

Per-container placement assigns your data to a random partition when the transaction commits time through the SessionHandle object. You must be able to reconstruct the SessionHandle object to retrieve your data from the data grid.

You can use zones to have more control over where primary shards and replica shards are placed in your domain. Using multiple zones in your deployment is advantageous when your data is in multiple physical locations. Geographically separating primaries and replicas is a way to ensure that catastrophic loss of one data center does not affect the availability of the data.

When data is spread across multiple zones, it is likely that clients are also spread across the topology. Routing clients to their local zone or data center has the obvious performance benefit of reduced network latency. Route clients to local zones or data centers when possible.

Configuring your topology for zone-preferred routing

Consider the following scenario. You have two data centers: Chicago and London. To minimize response time of clients, you want clients to read and write data to their local data center.

Primary shards must be placed in each data center so that transactions can be written locally from each location. Clients must be aware of zones to route to the local zone.

Per-container placement locates new primary shards on each container that is started. Replicas are placed according to zone and placement rules that are specified by the deployment policy. By default, a replica is placed in a different zone than its primary shard. Consider the following deployment policy for this scenario.

```
<?xml version="1.0" encoding="UTF-8"?>
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy ../deploymentPolicy.xsd"
        xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">
        <objectgridDeployment objectgridName="universe">
                <mapSet name="mapSet1" placementStrategy="PER_CONTAINER"
                        numberOfPartitions="3" maxAsyncReplicas="1">
                        <map ref="planet" />
                </mapSet>
        </objectgridDeployment>
</deploymentPolicy>
```

Each container that starts with the deployment policy receives three new primaries. Each primary has one asynchronous replica. Start each container with the appropriate zone name. Use the -zone parameter if you are starting your containers with the **startOgServer** script.

For a Chicago container server:

- UNIX Linux

  ```
  startOgServer.sh s1 -objectGridFile ../xml/universeGrid.xml
  -deploymentPolicyFile ../xml/universeDp.xml
  -catalogServiceEndPoints MyServer1.company.com:2809
  -zone Chicago
  ```

- Windows

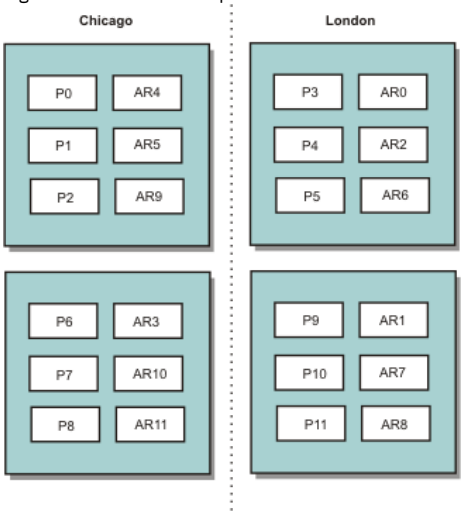
  ```
  startOgServer.bat s1 -objectGridFile ../xml/universeGrid.xml
  -deploymentPolicyFile ../xml/universeDp.xml
  -catalogServiceEndPoints MyServer1.company.com:2809
  -zone Chicago
  ```

If your containers are running in WebSphere Application Server, you must create a node group and name it with the prefix `ReplicationZone`. Servers that are running on the nodes in these node groups are placed in the appropriate zone. For example, servers running on a Chicago node might be in a node group named `ReplicationZoneChicago`.

See Zones for replica placement for more information.

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 1. Primaries and replicas in zones



Set the preferred zones for the clients. Provide a client properties file to your client Java virtual machine (JVM). Create a file named objectGridClient.properties and ensure that this file is in your classpath.

Include the preferZones property in the file. Set the property value to the appropriate zone. Clients in Chicago must have the following value in the objectGridClient.properties file:

```
preferZones=Chicago
```

The property file for London clients must contain the following value:

```
preferZones=London
```

This property instructs each client to route transactions to its local zone if possible. The topology asynchronously replicates data that is inserted into a primary shard in the local zone into the foreign zone.

Using the SessionHandle interface to route to the local zone

The per-container placement strategy does not use a hash-based algorithm to determine the location of your key-value pairs in the data grid. You must use SessionHandle objects to ensure that transactions are routed to the correct location when you are using this placement strategy. When a transaction is committed, a SessionHandle object is bound to the session if one has not already been set. The SessionHandle object can also be bound to the Session by calling the Session.getSessionHandle method before committing the transaction. The following code snippet shows a SessionHandle being bound before committing the transaction.

```
Session ogSession = objectGrid.getSession();

// binding the SessionHandle
SessionHandle sessionHandle = ogSession.getSessionHandle();

ogSession.begin();
ObjectMap map = ogSession.getMap("planet");
map.insert("planet1", "mercury");

// tran is routed to partition specified by SessionHandle
ogSession.commit();
```

Assume that the prior code was running on a client in your Chicago data center. The preferZones attribute is set to `Chicago` for this client. As a result, your deployment would route transactions to one of the primary partitions in the Chicago zone: partition 0, 1, 2, 6, 7, or 8.

The SessionHandle object provides a path back to the partition that is storing this committed data. The SessionHandle object must be reused or reconstructed and set on the Session to get back to the partition containing the committed data.

```
ogSession.setSessionHandle(sessionHandle);
ogSession.begin();

// value returned will be "mercury"
String value = map.get("planet1");
ogSession.commit();
```

The transaction in this code reuses the SessionHandle object that was created during the insert transaction. The get transaction then routes to the partition that holds the inserted data. Without the SessionHandle object, the transaction cannot retrieve the inserted data.

How container and zone failures affect zone-based routing

Generally, a client with the preferZones property set routes all transactions to the specified zone or zones. However, the loss of a container results in the promotion of a replica shard to a primary shard. A client that was previously routing to partitions in the local zone must retrieve previously inserted data from the remote zone.

Consider the following scenario. A container in the Chicago zone is lost. It previously contained primaries for partitions 0, 1, and 2. The new primary shards for these partitions are then placed in the London zone because the London zone hosted the replicas for these partitions.

Any Chicago client that is using a SessionHandle object that points to one of the failed-over partitions now reroutes to London. Chicago clients that are using new SessionHandle objects route to Chicago-based primaries.

Similarly, if the entire Chicago zone is lost, all replicas in the London zone are promoted to primaries. In this scenario, all Chicago clients route their transactions to London.

Related tasks:
Controlling shard placement with zones
Defining zones for container servers
Viewing zone information with the xscmd utility
Administering with the xscmd utility
Related reference:
Example: Zone definitions in the deployment policy descriptor XML file
Deployment policy descriptor XML file

Defining zones for container servers

Zones are collections of container servers. A container server can belong only one zone. A container server is assigned to a zone when it starts.

About this task

You must plan your zones before you start your container servers because container servers define their zone membership at startup. If you want to change the zone membership of a container server, you must restart the server with the new zone information.

Procedure

How you define zones depends on if you are using stand-alone container servers or container servers that are running within WebSphere® Application Server:

- Define zones for stand-alone container servers.
    1. Use the -zone parameter of the **startOgServer** script to specify the zone for all the containers in the started server. For more information about starting servers, see Starting and stopping stand-alone servers.

2. You can also zone names when you are starting container servers programmatically with the embedded server API. For more information, see Using the embedded server API to start and stop servers.

- Define zones for container servers that are running within WebSphere Application Server.
  You can use node groups to place container servers in specific zones. Use the following syntax to name your node group to assign it a zone: `ReplicationZone<identifier>`. When you define zones in the deployment policy, you must name the zones exactly as you named the node groups. The node group name and the zone name in the deployment policy descriptor XML file must be identical

  Important: WebSphere Application Server does not prohibit nodes from being in multiple node groups. Because container servers can be only one zone, ensure that your nodes are in exactly one ReplicationZone node group.
  For example, divide four nodes into two zones, A and B.

  1. Configure four nodes: node1, node2, node3, and node4, each node having two servers.
  2. Create a node group named ReplicationZoneA and a node group named ReplicationZoneB.
  3. Add node1 and node2 to ReplicationZoneA and add node3 and node4 to ReplicationZoneB.
  4. Define ReplicationZoneA and ReplicationZoneB in your deployment policy descriptor XML file. See Example: Zones in a WebSphere Application Server environment for an example.
  5. When the servers on node1 and node2 are started, they join ReplicationZoneA, or zone A in the WebSphere eXtreme Scale configuration. The servers on node3 and node4 join ReplicationZoneB, as zone B in the WebSphere eXtreme Scale configuration.

Related concepts:
Zones
Zone-preferred routing
Related reference:
Example: Zone definitions in the deployment policy descriptor XML file
Deployment policy descriptor XML file

Example: Zone definitions in the deployment policy descriptor XML file

You can specify zones and zone rules with the deployment policy descriptor XML file.

Example: Primary and replica shards in different zones

This example places primary shards in one zone, and replica shards in a different zone, with a single asynchronous replica. All primary shards start in the DC1 zone. Replica shards start in zone DC2.

```
<?xml version="1.0" encoding="UTF-8"?>
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy
        ../deploymentPolicy.xsd"        xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">
 <objectgridDeployment objectgridName="library">
        <mapSet name="ms1" numberOfPartitions="13" minSyncReplicas="0"
                maxSyncReplicas="0" maxAsyncReplicas="1">
                <map ref="book" />
                <zoneMetadata>
                                <shardMapping shard="P" zoneRuleRef="primaryRule"/>
                                <shardMapping shard="A" zoneRuleRef="replicaRule"/>
                                <zoneRule name="primaryRule">
                                        <zone name="DC1" />
                                </zoneRule>
                                <zoneRule name="replicaRule">
                                </zoneRule>
                </zoneMetadata>
        </mapSet>
  </objectgridDeployment>
 </deploymentPolicy>
```

One asynchronous replica is defined in the ms1 mapSet element. Therefore, two shards exist for each partition: a primary and one asynchronous replica. In the zoneMetadata element, a shardMapping element is defined for each shard: P for the primary, and DC1 for the asynchronous replica. The primaryRule attribute defines the zone set for the primary shards, which is just zone DC1, and this rule is to be used for primary shard placement. Asynchronous replicas are placed in the DC2 zone.

However, if the DC2 zone is lost, the replica shards become unavailable. The loss or failure of a container server in the DC1 zone can result in data loss, even though a replica has been specified.

To address this possibility, you can either add a zone or add a replica, as described in the following sections.

Example: Add a zone, striping shards

The following code configures a new zone:

```
<?xml version="1.0" encoding="UTF-8"?>
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy
        ../deploymentPolicy.xsd" xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">
        <objectgridDeployment objectgridName="library">
                <mapSet name="ms1" numberOfPartitions="13" minSyncReplicas="0"
                        maxSyncReplicas="0" maxAsyncReplicas="1">
                                <map ref="book" />
```

```
                                <zoneMetadata>
                                        <shardMapping shard="P" zoneRuleRef="stripeRule"/>
                                        <shardMapping shard="A" zoneRuleRef="stripeRule"/>
                                        <zoneRule name="stripeRule" exclusivePlacement="true">
                                                <zone name="A" />
                                                <zone name="B" />
                                                <zone name="C" />
                                        </zoneRule>
                                </zoneMetadata>
                        </mapSet>
                </objectgridDeployment>
</deploymentPolicy>
```

Three total zones have been defined in this code: A, B, and C. Instead of separate primary and replica zone rules, a shared zone rule named stripeRule is defined. This rule includes all of the zones, with the exclusivePlacement attribute set to `true`. The eXtreme Scale placement policy ensures that primary and replica shards are in separate zones. This striping of placement causes primary and replica shards to spread across both zones to conform to this policy. Adding a third zone C ensures that losing any one zone does not result in data loss, and still leaves primary and replica shards for each partition. A zone failure results in the loss of either the primary shard, the replica shard, or neither. Any lost shard is replaced from the surviving shard in a surviving zone, placing it in the other surviving zone.

Example: Add a replica and define multiple data centers

The classic two data-center scenario has high speed, low latency networks in each data center, but high latency between the data centers. Synchronous replicas are used in each data center where the low latency minimizes the impact of replication on response times. Asynchronous replication is used between data centers, so the high latency network has no impact on response time.

```
<?xml version="1.0" encoding="UTF-8"?>
 <deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy
        ../deploymentPolicy.xsd" xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">
        <objectgridDeployment objectgridName="library">
                <mapSet name="ms1" numberOfPartitions="13" minSyncReplicas="1"
                        maxSyncReplicas="1" maxAsyncReplicas="1">
                        <map ref="book" />
                        <zoneMetadata>
                                <shardMapping shard="P" zoneRuleRef="primarySync"/>
                                <shardMapping shard="S" zoneRuleRef="primarySync"/>
                                <shardMapping shard="A" zoneRuleRef="async"/>
                                <zoneRule name="primarySync" exclusivePlacement="false">
                                        <zone name="DC1" />
                                        <zone name="DC2" />
                                </zoneRule>
                                <zoneRule name="async" exclusivePlacement="true">
                                        <zone name="DC1" />
                                        <zone name="DC2" />
                                </zoneRule>
                        </zoneMetadata>
                </mapSet>
  </objectgridDeployment>
 </deploymentPolicy>
```

The primary and synchronous replica share the primarySync rule with an exclusivePlacement attribute setting of `false`. The exclusivePlacement attribute set to false creates a configuration with the primary and synchronous replica shards of each partition placed in the same zone. The asynchronous replica shard uses a second zone rule with mostly the same zones as the primarySync zone rule. However the asynchronous replica uses the exclusivePlacement attribute set to `true`. The exclusivePlacement attribute, when set to true, means that a shard cannot be placed in a zone with another shard from the same partition. As a result, the asynchronous replica shard does not get placed in the same zone as the primary or synchronous replica shard. There are three shards per partition in this mapSet: a primary, and both a synchronous and asynchronous replica, so there are three shardMapping elements, one for each shard.

If a zone is lost, any asynchronous replicas are lost, and not regenerated, because they have no separate zone. If the primary and replica shards are lost, then the surviving asynchronous replica is promoted to primary, and a new synchronous replica is created in the zone. The primaries and replicas are striped across each zone.

With exclusive placement, each shard has its own zone: You must have enough zones for all the shards you want to place in their own zones. If a rule has one zone, only one shard can be placed in the zone. With two zones, you can have up to two shards in the zone.

Example: Zones in a WebSphere Application Server environment

The following code configures a new zone:

```
<?xml version="1.0" encoding="UTF-8"?>
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy
        ../deploymentPolicy.xsd" xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">
        <objectgridDeployment objectgridName="library">
                <mapSet name="ms1" numberOfPartitions="13" minSyncReplicas="0"
                        maxSyncReplicas="0" maxAsyncReplicas="1">
                                <map ref="book" />
                                <zoneMetadata>
                                        <shardMapping shard="P" zoneRuleRef="stripeRule"/>
                                        <shardMapping shard="A" zoneRuleRef="stripeRule"/>
                                        <zoneRule name="stripeRule" exclusivePlacement="true">
                                                <zone name="ReplicationZoneA" />
                                                <zone name="ReplicationZoneB" />
                                                <zone name="ReplicationZoneC" />
```

```
                                        </zoneRule>
                                    </zoneMetadata>
                </mapSet>
            </objectgridDeployment>
</deploymentPolicy>
```

For this example, three node groups are defined in the WebSphere® Application Server environment: ReplicationZoneA, ReplicationZoneB, and ReplicationZoneC. The node group name and the zone name in the deployment policy descriptor XML file must be identical, and must contain the text `ReplicationZone<identifier>`. This file defines a similar configuration to the striping shards example, but shows the required naming for a WebSphere Application Server configuration.

Related concepts:
Zones
Zone-preferred routing
Related tasks:
Controlling shard placement with zones
Defining zones for container servers
Viewing zone information with the xscmd utility
Administering with the xscmd utility

Viewing zone information with the xscmd utility

You can use the **xscmd** utility to view information about your current zone deployment, including shard placement data.

Before you begin

- Deploy a distributed data grid with multiple data centers. See Zone-preferred routing for more information.

About this task

You can determine information about your configuration related to zone settings by using the **xscmd** utility that ships with the product.

Procedure

Use the **xscmd** utility to determine information about the shards of data. Run the following command:

```
xscmd -c showPlacement -z zone_name
```

Example

You can also run a simpler scenario by using the getting started sample: *wxs_install_root*/ObjectGrid/gettingstarted. See Tutorial: Getting started with WebSphere eXtreme Scale for more information.

1. Start a catalog server:

   ```
   runcat.bat
   ```

2. Determine your required number of replicas, zone rules, containers, and other settings such as with the following command:

   ```
   startOgServer.bat serverA0 -objectgridFile xml\objectgrid.xml
   -deploymentPolicyFile xml\deployment.xml -zone zoneA
   ```

3. You can stop container processes to simulate failure in the data grid:

   ```
   stopOgServer.bat serverA0,serverA1,serverB0 -catalogServiceEndPoints localhost:2809
   ```

   .
   If the server that contains the last shard of a partition is stopped, eXtreme Scale allocates a new primary shard. You can check for data loss:
   - The **runclient** script inserts and reads item in your data grid.
   - The **xscmd -c showMapSizes** command shows the number of items in the data grid.
4. Show active container servers with the following command:

   ```
   xscmd -c showPlacement -z zone_name
   ```

Related concepts:
Zones
Zone-preferred routing
Related tasks:
Administering with the xscmd utility
Related reference:
Example: Zone definitions in the deployment policy descriptor XML file
Deployment policy descriptor XML file

Configuring catalog and container servers

WebSphere® eXtreme Scale has two types of servers: catalog servers and container servers. Catalog servers control the placement of shards and discover and monitor the container servers. Multiple catalog servers can join a catalog service domain to provide high availability to the environment. A container server is a Java™ virtual machine (JVM) that stores the application data for the data grid.

- Configuring catalog servers and catalog service domains
  The catalog service hosts logic that is typically idle during steady states. As a result, the catalog service minimally influences scalability. The service is built to service hundreds of container servers that become available simultaneously. For high availability, configure the catalog service into a catalog service domain.
- Configuring container servers
  The container server stores application data for the data grid. This data is generally broken into parts, which are called partitions, which are hosted across multiple container servers. Each container server in turn hosts a subset of the complete data.
- Configuring eXtreme Scale servers to run in the Liberty profile
  You must create server and catalog service definitions to run WebSphere eXtreme Scale in the Liberty profile.

Related concepts:
Installation topologies
Catalog service
Container servers, partitions, and shards
Related tasks:
Starting and stopping stand-alone servers
Using the embedded server API to start and stop servers
Related reference:
Server properties file
ObjectGrid descriptor XML file

Configuring eXtreme Scale servers to run in the Liberty profile

8.5+ You must create server and catalog service definitions to run WebSphere® eXtreme Scale in the Liberty profile.

Before you begin

Install the WebSphere Application Server Liberty profile and WebSphere eXtreme Scale. For more information, see Installing the Liberty profile.

Procedure

1. Create a server definition; for example:

   ```
   <wlp install_root>/bin/server create <your server name>
   ```

2. Find the server.xml file under your server definition and open it in an editor. A commented feature manager stanza already exists. You can find the server definition in the following directory:

   ```
   <wlp install_root>/usr/servers/<your server name>
   ```

3. Create a catalog service definition. To define a catalog service in a server, you need the WebSphere eXtreme Scale server feature and the server configuration file.
   a. Add the server feature to the server definition: 8.5

   ```
   <featureManager>
        <feature>eXtremeScale_server-1.0</feature>
   </featureManager>
   ```

   b. Add the configuration to tell the server feature that this server is a catalog service; for example:

   ```
   <com.ibm.ws.xs.server.config isCatalog="true"/>
   ```

What to do next

After you configure eXtreme Scale servers, you can start yours servers in the Liberty profile. For more information, see Starting and stopping servers in the Liberty profile.

Configuring catalog servers and catalog service domains

The catalog service hosts logic that is typically idle during steady states. As a result, the catalog service minimally influences scalability. The service is built to service hundreds of container servers that become available simultaneously. For high availability, configure the catalog service into a catalog service domain.

Before you begin

After a catalog service domain is started, the members of the data grid bind together. Carefully plan your catalog service domain topology, because you cannot modify your catalog service domain configuration at run time. Spread out the data grid as diversely as possible to prevent errors.

The best practice to avoid a single point of failure for your catalog service domain is to start a minimum of three catalog servers on three different nodes.

If you are using only two nodes, configure two catalog servers on each of the two nodes for a total of four catalog server processes. Creating this configuration ensures that when only one of the nodes is started, the required two catalog servers are running. You must start at least two catalog servers at the same time. When catalog servers start, they look for other catalog servers in the configuration, and do not start successfully until at least one other catalog sever is found.

Procedure

- Configure stand-alone catalog servers and catalog service domains.
  You configure stand-alone catalog server and catalog service domains with parameters and property files that you pass to the start server command or to the embedded server API.
  - Example: Configuring catalog service domains
  - Starting and stopping stand-alone servers
  - Catalog server properties
- Configure catalog servers and catalog service domains in WebSphere® Application Server
  Configure catalog servers that run in WebSphere Application Server with the WebSphere Application Server administrative console, administrative tasks, and the server properties file. The server life cycle is controlled by the process life cycle within WebSphere Application Server. When processes start or stop in WebSphere Application Server, the catalog servers that are running on these processes also start or stop.
  - Creating catalog service domains in WebSphere Application Server
  - Configuring the catalog service in WebSphere Application Server

- Example: Configuring catalog service domains
  When you are using the catalog service, a minimum of two catalog servers are required to avoid a single point of failure. Depending on the number of nodes in your environment, you can create different configurations to ensure that at least two catalog servers are always running.
- Configuring WebSphere eXtreme Scale with WebSphere Application Server
  You can run catalog service and container server processes in WebSphere Application Server. The process to configure these servers is different than a stand-alone configuration. The catalog service can automatically start in WebSphere Application Server servers or deployment managers. Container process start when an eXtreme Scale application is deployed and started in the WebSphere Application Server environment.
- Configuring the quorum mechanism
  Configure the quorum mechanism for each catalog server. You must enable the quorum mechanism on all of the catalog servers in the catalog service domain. Changing the quorum configuration requires a restart.
- Tuning the heartbeat interval setting for failover detection
  You can configure the amount of time between system checks for failed servers with the heartbeat interval setting. This setting applies to catalog servers only.

Related concepts:
Installation topologies
Catalog service
Related tasks:
Configuring eXtreme Scale connection factories
Related reference:
Client properties file
Server properties file
startOgServer script
Related information:
Catalog service domain settings

Example: Configuring catalog service domains

When you are using the catalog service, a minimum of two catalog servers are required to avoid a single point of failure. Depending on the number of nodes in your environment, you can create different configurations to ensure that at least two catalog servers are always running.

Example: Starting four catalog servers on two nodes in a stand-alone environment

The following script starts catalog servers cs0 and cs1 on the host1 node, and starts catalog servers cs2 and cs3 on the host2 node.

```
./startOgServer.sh|bat cs0 -listenerPort 2809 -catalogServiceEndPoints
cs0:host1:6601:6602,cs1:host1:6603:6604,cs2:host2:6601:6602,cs3:host2:6603:6604
-quorum true -jvmArgs -Xmx256m

./startOgServer.sh|bat cs1 -listenerPort 2810 -catalogServiceEndPoints
cs0:host1:6601:6602,cs1:host1:6603:6604,cs2:host2:6601:6602,cs3:host2:6603:6604
-quorum true -jvmArgs -Xmx256m

./startOgServer.sh|bat cs2 -listenerPort 2809 -catalogServiceEndPoints
cs0:host1:6601:6602,cs1:host1:6603:6604,cs2:host2:6601:6602,cs3:host2:6603:6604
-quorum true -jvmArgs -Xmx256m

./startOgServer.sh|bat cs3 -listenerPort 2810 -catalogServiceEndPoints
cs0:host1:6601:6602,cs1:host1:6603:6604,cs2:host2:6601:6602,cs3:host2:6603:6604
-quorum true -jvmArgs -Xmx256m
```

Remember: You must use the -listenerPort option because the catalog servers that are running on a node each require a unique port number.

Example: Starting multiple catalog servers in a WebSphere Application Server environment

Catalog servers start automatically in a WebSphere® Application Server environment. You can define multiple catalog servers to start by creating a catalog service domain. After you specify multiple endpoints in the catalog service domain, restart the included application servers so that the catalog servers start in parallel.

- **WebSphere Application Server Network Deployment**: You can choose multiple existing application servers from the cell to be members of your catalog service domain. Since, you can only start a cluster of catalog servers that are in the same core group, verify that any application servers in a catalog service domain are in the same core group.
- **Base WebSphere Application Server**: You can only start a single catalog server in a base application server using the **startServer** command . To start a cluster, use the **startXsServer** command.

Related tasks:
Configuring WebSphere eXtreme Scale with WebSphere Application Server
Configuring the catalog service in WebSphere Application Server
Creating catalog service domains in WebSphere Application Server
Starting and stopping servers in a WebSphere Application Server environment
Troubleshooting administration
Administering with the xscmd utility
Starting and stopping stand-alone servers
Starting a stand-alone catalog service
Related reference:
Catalog service domain administrative tasks
Server properties file

Configuring WebSphere eXtreme Scale with WebSphere Application Server

You can run catalog service and container server processes in WebSphere® Application Server. The process to configure these servers is different than a stand-alone configuration. The catalog service can automatically start in WebSphere Application Server servers or deployment managers. Container process start when an eXtreme Scale application is deployed and started in the WebSphere Application Server environment.

About this task

Attention: Do not collocate your container servers with catalog servers in a production environment. Include the catalog service in multiple node agent processes or in an application server that is not hosting an eXtreme Scale application.

- Configuring the catalog service in WebSphere Application Server
  Catalog service processes can run in WebSphere Application Server. The server life cycle in WebSphere Application Server determines when the catalog service starts and stops.

Related concepts:
Interoperability with other products
Monitoring with vendor tools
Installation topologies
Tuning garbage collection with WebSphere Real Time
Example: Configuring catalog service domains
Catalog service
High availability catalog service
Administering
Related reference:
Catalog service domain administrative tasks
Server properties file
Related information:
Configure WebSphere Commerce to use WebSphere eXtreme Scale for dynamic cache to improve performance and scale
WebSphere Business Process Management and Connectivity integration

Configuring the catalog service in WebSphere Application Server

Catalog service processes can run in WebSphere® Application Server. The server life cycle in WebSphere Application Server determines when the catalog service starts and stops.

Procedure

1. Choose one or more WebSphere Application Server processes to augment with the WebSphere eXtreme Scale profile. See Creating and augmenting profiles for WebSphere eXtreme Scale for more information. If you want the catalog service to start automatically in WebSphere Application Server Network Deployment on the deployment manager, install WebSphere eXtreme Scale on the deployment manager node and augment the deployment manager profile.

2. Configure server properties files for the WebSphere Application Server processes and add to the class path for the node. See Server properties file for more information.
3. Configure a catalog service domain. The catalog service domain is a group of catalog servers within your environment. See Creating catalog service domains in WebSphere Application Server for more information.
4. Start the WebSphere Application Server processes that are hosting the catalog servers. See Starting and stopping servers in a WebSphere Application Server environment for more information.

- Creating catalog service domains in WebSphere Application Server
  Catalog service domains define a group of catalog servers that manage the placement of shards and monitor the health of container servers in your data grid.

Related concepts:
Example: Configuring catalog service domains
Installation topologies
Catalog service
High availability catalog service
Container servers, partitions, and shards
Related tasks:
Configuring WebSphere Application Server applications to automatically start container servers
Configuring container servers in WebSphere Application Server
Controlling placement
Related reference:
Catalog service domain administrative tasks
Server properties file
Deployment policy descriptor XML file
ObjectGrid descriptor XML file

Creating catalog service domains in WebSphere Application Server

Catalog service domains define a group of catalog servers that manage the placement of shards and monitor the health of container servers in your data grid.

Before you begin

- Install WebSphere® eXtreme Scale on WebSphere Application Server. See Installing WebSphere eXtreme Scale or WebSphere eXtreme Scale Client with WebSphere Application Server for more information.

About this task

By creating a catalog service domain, you are defining a highly available collection of catalog servers.
These catalog servers can run in WebSphere Application Server within a single cell and core group. The catalog service domain can also define a remote group of servers that run in different Java™ SE processes or other WebSphere Application Server cells.

For catalog servers that run on existing application servers within the cell: When you define a catalog service domain that places catalog servers on the application servers within the cell, the core group mechanisms of WebSphere Application Server are used. The catalog service automatically starts on the application servers in the cell. As a result, the members of a single catalog service domain cannot span the boundaries of a core group, and a catalog service domain therefore cannot span cells. However, WebSphere eXtreme Scale container servers and clients can span cells by connecting to a catalog server across cell boundaries, such as a stand-alone catalog service domain or a catalog service domain embedded in another cell.

For remote catalog servers: You can connect WebSphere eXtreme Scale containers and clients to a catalog service domain that is running in another WebSphere Application Server cell or that are running 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 should 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** 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. See Starting a stand-alone catalog service for more information about starting catalog servers.

Attention: Do not collocate the catalog services with WebSphere eXtreme Scale container servers in a production environment. Include the catalog service in multiple node agent processes or in an application server that is not hosting a WebSphere eXtreme Scale application.

Procedure

1. Create the catalog service domain.
   a. In the WebSphere Application Server administrative console, click System administration > WebSphere eXtreme Scale > Catalog service domains > New.
   b. Define a name, default value, and JMX authentication credentials for your catalog service domain. If you are configuring remote endpoints for the catalog service domain, the name of the catalog service domain should match the name of the catalog service domain that you specify when you start the catalog servers.
   c. Add catalog server endpoints. You can either select existing application servers or add remote servers that are running a catalog service.
2. Test the connection to the catalog servers within your catalog service domain. For existing application servers, catalog servers start when the associated application server is started. For remote application servers, you must start the servers manually using the **startOgServer** command or embedded server API.
   a. In the WebSphere Application Server administrative console, click System administration > WebSphere eXtreme Scale > Catalog service domains.

b. Select the catalog service domain that you want to test and click Test connection. When you click this button, all of the defined catalog service domain end points are queried one by one, if any one end point is available, returns a message that indicates that the connection to the catalog service domain was successful.

- Catalog service domain administrative tasks
  You can use the Jacl or Jython scripting languages to manage catalog service domains in your WebSphere Application Server configuration.
- Catalog service domain collection
  Use this page to manage catalog service domains. Catalog service domains define a group of catalog servers that manage the placement of shards and monitors the health of container servers in your data grid.
- Catalog service domain settings
  Use this page to manage the settings for a specific catalog service domain. Catalog service domains define a group of catalog servers that manage the placement of shards and monitors the health of container servers in your data grid. You can define a catalog service domain that is in the same cell as your deployment manager. You can also define remote catalog service domains if your WebSphere eXtreme Scale configuration is in a different cell or your data grid is made up of Java SE processes.
- Client security properties
  Use this page to configure client security for a catalog service domain. These settings apply to all the servers in your catalog service domain. These properties can be overridden by specifying a splicer.properties file with the com.ibm.websphere.xs.sessionFilterProps custom property or by splicing the application EAR file.
- Catalog service domain custom properties
  You can further edit the configuration of the catalog service domain by defining custom properties.

Related concepts:
Example: Configuring catalog service domains
Installation topologies
Catalog service
High availability catalog service
Related reference:
Catalog service domain administrative tasks
Server properties file

Catalog service domain administrative tasks

You can use the Jacl or Jython scripting languages to manage catalog service domains in your WebSphere® Application Server configuration.

Requirements

You must have the WebSphere eXtreme Scale Client installed in your WebSphere Application Server environment.

List all administrative tasks

To get a list of all of the administrative tasks that are associated with catalog service domains, run the following command with **wsadmin**:

- Using Jacl:

```
wsadmin>$AdminTask help XSDomainManagement
```

- Using a Jython string:

```
wsadmin>print AdminTask.help ('XSDomainManagement')
```

Commands

The administrative tasks for catalog service domains include the following commands:

- createXSDomain
- deleteXSDomain
- getDefaultXSDomain
- listXSDomains
- modifyXSDomain
- testXSDomainConnection
- testXSServerConnection

List all administrative task command arguments

To get a list of all of the command arguments associated with catalog service domain administrative tasks, run the following command with **wsadmin**:

- Using Jacl:

```
wsadmin>$AdminTask help <command>
wsadmin>$AdminTask help <command> <commandStep>
Example: wsadmin>$AdminTask help createXSDomain defineDomainServers
```

- Using a Jython string:

```
wsadmin>print AdminTask.help ('<command>')
Example: wsadmin>print AdminTask.help ('createXSDomain')
```

createXSDomain

The **createXSDomain** command registers a new catalog service domain.

Table 1. createXSDomain command arguments

Argument	Description
-name (required)	Specifies the name of the catalog service domain that you want to create.
-default	Specifies whether the catalog service domain is the default for the cell. The default value is `true`. (Boolean: set to `true` or `false`)

Table 2. defineDomainServers step arguments

Argument	Description
name_of_endpoint	Specifies the name of the catalog service domain endpoint. • **For existing application servers**: Specifies the name of the endpoint must be in the following format, using backslashes: *cell_name\node_name\server_name* • **For remote servers**: Specifies the host name of the remote server. You can have the same name for multiple endpoints, but the client port values must be unique for each endpoint.
custom_properties	Specifies custom properties for the catalog service domain endpoint. If you do not have any custom properties, use a set of double quotation marks (`""`) for this argument.
endpoint_ports	Specifies the port numbers for the catalog service domain endpoint. The ports must be specified in the following order: *<client_port>*, *<listener_port>* For existing application servers where only a client port is required, enter the client port value as either `"2809"` or `"2809,"`. For remote servers where only a listener port is required, enter the listener port value as: `",9810"` Client Port Specifies the port that is used for communication between the catalog servers in the catalog service domain. This value is only required for existing application servers (catalog servers that are running in WebSphere Application Server processes) and can be set to any port that is not being used elsewhere. Listener Port Specifies the port that is used for communication with clients. This value is required for remote endpoints and must match the value used when the catalog service was started. The listener port is used by clients and containers to communicate with the catalog service. **For WebSphere eXtreme Scale remote endpoints**: Defines the Object Request Broker (ORB) listener port for containers and clients to communicate with the catalog service through the ORB. For WebSphere Application Server endpoints, the listener port value is optional because the value is inherited from the BOOTSTRAP_ADDRESS port configuration.

Table 3. configureClientSecurity step arguments

Argument	Description
-securityEnabled	Specifies that client security is enabled for the catalog server. The server properties file that is associated with the selected catalog server must have a matching securityEnabled setting in the server properties file. If these settings do not match, an exception results. (Boolean: set to `true` or `false`)
-credentialAuthentication (optional)	Indicates if credential authentication is enforced or supported. Never No client certificate authentication is enforced. Required Credential authentication is always enforced. If the server does not support credential authentication, the client cannot to connect to the server. Supported (Default) Credential authentication is enforced only if both the client and server support credential authentication.
-authenticationRetryCount (optional)	Specifies the number of times that authentication gets tried again if the credential is expired. If you do not want to try authentication again, set the value to `0`. The default value is `0`.
-credentialGeneratorClass	Indicates the com.ibm.websphere.objectgrid.security.plugins.builtins. WSTokenCredentialGenerator implementation class, so the client retrieves the security tokens from the thread.

Argument	Description
-credentialGeneratorProps	Specifies the properties for the CredentialGenerator implementation class. The properties are sent to the object with the setProperties(String) method. The credential generator properties value is used only when a value is specified for the Credential generator class field. Properties for **com.ibm.websphere.objectgrid.security.plugins.UserPasswordCredentialGenerator** includes userid_password which can be defined as “userid password”. Note: Because parsing of the userid_password property depends upon the space character as the value separator, userids and passwords which contain spaces must use the “\20” escape character to represent a space. For example: If the userid is “Test User Id” and the password is “Test Password”, the userid_password property should be entered as: “Test\20User\20Id Test\20Password”. Properties for **com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredentialGenerator** includes the property subject_type, which can be defined as either“runAs” or “caller”.

Return value:

Batch mode example usage

Batch mode requires correct formatting of the command entry. Consider using interactive mode to ensure the values that you enter are processed correctly. When you use batch mode, you must define the -defineDomainServers step arguments using a specific array of properties. This array of properties is in the format *name_of_endpoint custom_properties endpoint_ports*. The *endpoint_ports* value is a list of ports that must be specified in the following order: *<client_port>,<listener_port>*.

- Create a catalog service domain of remote endpoints using Jacl:

```
$AdminTask createXSDomain {-name TestDomain -default true -defineDomainServers
{{xhost1.ibm.com "" ,2809}} -configureClientSecurity {-securityEnabled false
-credentialAuthentication Required -authenticationRetryCount 0 -credentialGeneratorClass
com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredentialGenerator
-credentialGeneratorProps "manager manager1"}}
```

- Create a catalog service domain of remote endpoints using Jython string:

```
AdminTask.createXSDomain('[-name TestDomain -default true
-defineDomainServers [[xhost1.ibm.com "" ,2809]
[xhost2.ibm.com "" ,2809]] -configureClientSecurity [-securityEnabled false
-credentialAuthentication Required -authenticationRetryCount 0 -credentialGeneratorClass
com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredentialGenerator
-credentialGeneratorProps "manager manager1"] ]')
```

- Create a catalog service domain of existing application server endpoints using Jacl:

```
$AdminTask createXSDomain {-name TestDomain -default true -defineDomainServers
 {{cellName/nodeName/serverName "" 1109}}}
```

Interactive mode example usage

- Using Jacl:

```
$AdminTask createXSDomain {-interactive}
```

- Using Jython string:

```
AdminTask.createXSDomain ('[-interactive]')
```

deleteXSDomain

The **deleteXSDomain** command deletes a catalog service domain.

Required parameters:

-name
Specifies the name of the catalog service domain to delete.

Return value:

Batch mode example usage

- Using Jacl:

```
$AdminTask deleteXSDomain {-name TestDomain }
```

- Using Jython string:

```
AdminTask.deleteXSDomain('[-name TestDomain ]')
```

Interactive mode example usage

- Using Jacl:

```
$AdminTask deleteXSDomain {-interactive}
```

- Using Jython string:

```
AdminTask.deleteXSDomain ('[-interactive]')
```

getDefaultXSDomain

The **getDefaultXSDomain** command returns the default catalog service domain for the cell.

Required parameters: None

Return value: The name of the default catalog service domain.

Batch mode example usage

- Using Jacl:

```
$AdminTask getDefaultXSDomain
```

- Using Jython string:

```
AdminTask.getDefaultXSDomain
```

Interactive mode example usage

- Using Jacl:

```
$AdminTask getDefaultXSDomain {-interactive}
```

- Using Jython string:

```
AdminTask.getDefaultXSDomain ('[-interactive]')
```

listXSDomains

The **listXSDomains** command returns a list of the existing catalog service domains.

Required parameters: None

Return value: A list of all of the catalog service domains in the cell.

Batch mode example usage

- Using Jacl:

```
$AdminTask listXSDomains
```

- Using Jython string:

```
AdminTask.listXSDomains
```

Interactive mode example usage

- Using Jacl:

```
$AdminTask listXSDomains {-interactive}
```

- Using Jython string:

```
AdminTask.listXSDomains ('[-interactive]')
```

modifyXSDomain

The **modifyXSDomain** command modifies an existing catalog service domain.

Batch mode requires correct formatting of the command entry. Consider using interactive mode to ensure the values that you enter are processed correctly. When you use batch mode, you must define the -modifyEndpoints, -addEndpoints and -removeEndpoints step arguments using a specific array of properties. This array of properties is in the format *name_of_endpoint host_name custom_properties endpoint_ports*. The *endpoint_ports* value is a list of ports that must be specified in the following order: *<client_port>,<listener_port>*.

Table 4. modifyXSDomain command arguments

Argument	Description
-name (required)	Specifies the name of the catalog service domain that you want to edit.
-default	If set to `true`, specifies that the selected catalog service domain is the default for the cell. (Boolean)

Table 5. modifyEndpoints step arguments

Argument	Description

Argument	Description
name_of_endpoint	Specifies the name of the catalog service domain endpoint. • **For existing application servers**: Specifies the name of the endpoint in the following format, using backslashes: *cell_name\node_name\server_name* • **For remote servers**: Specifies the host name of the remote server. You can have the same name for multiple endpoints, but the listener port values must be unique for each endpoint.
endpoint_ports	Specifies the port numbers for the catalog service domain endpoint. The endpoints must be specified in the following order: *<client_port>,<listener_port>* For existing application servers where only a client port is required, enter the client port value as either: "2809" or "2809,". For remote servers where only a listener port is required, enter the listener port value as: ",9810". Client Port Specifies the port that is used for communication between the catalog servers in the catalog service domain. This value is only required for existing application servers (catalog servers that are running in WebSphere Application Server processes) and can be set to any port that is not being used elsewhere. Listener Port Specifies the port that is used for communication with clients. This value is required for remote endpoints and must match the value used when the catalog service was started. The listener port is used by clients and containers to communicate with the catalog service. **For WebSphere eXtreme Scale remote endpoints**: Defines the Object Request Broker (ORB) listener port for containers and clients to communicate with the catalog service. For WebSphere Application Server endpoints, specifying the listener port value is optional. The value is inherited from the BOOTSTRAP_ADDRESS port configuration.

Table 6. addEndpoints step arguments

Argument	Description
name_of_endpoint	Specifies the name of the catalog service domain endpoint. • **For existing application servers**: Specifies the name of the endpoint in the following format, using backslashes: *cell_name\node_name\server_name* • **For remote servers**: Specifies the host name of the remote server. You can have the same name for multiple endpoints, but the listener port values must be unique for each endpoint.
custom_properties	Specifies custom properties for the catalog service domain endpoint. If you do not have any custom properties, use a set of double quotation marks ("") for this argument.
endpoint_ports	Specifies the port numbers for the catalog service domain endpoint. The endpoints must be specified in the following order: *<client_port>,<listener_port>* For existing application servers where only a client port is required, enter the client port value as either: "2809" or "2809,". For remote servers where only a listener port is required, enter the listener port value as : ",9810". Client Port Specifies the port that is used for communication between the catalog servers in the catalog service domain. This value is only required for existing application servers (catalog servers that are running in WebSphere Application Server processes) and can be set to any port that is not being used elsewhere. Listener Port Specifies the port that is used for communication with clients. This value is required for remote endpoints and must match the value used when the catalog service was started. The listener port is used by clients and containers to communicate with the catalog service. **For WebSphere eXtreme Scale remote endpoints**: Defines the Object Request Broker (ORB) listener port for containers and clients to communicate with the catalog service through the ORB. For WebSphere Application Server endpoints, specifying the listener port value is optional because the value is inherited from the BOOTSTRAP_ADDRESS port configuration.

Table 7. removeEndpoints step arguments

Argument	Description
name_of_endpoint	Specifies the name of the catalog service endpoint to delete.

Table 8. configureClientSecurity step arguments

Argument	Description
-securityEnabled	Specifies that client security is enabled for the catalog server. The server properties file that is associated with the selected catalog server must have a matching securityEnabled setting in the server properties file. If these settings do not match, an exception results. (Boolean: set to `true` or `false`)

Argument	Description
-credentialAuthentication (optional)	Indicates whether credential authentication is enforced or supported. Never No client certificate authentication is enforced. Required Credential authentication is always enforced. If the server does not support credential authentication, the client cannot to connect to the server. Supported (Default) Credential authentication is enforced only if both the client and server support credential authentication.
-authenticationRetryCount (optional)	Specifies the number of times that authentication gets tried again if the credential is expired. If you do not want to try authentication again, set the value to `0`. The default value is `0`.
-credentialGeneratorClass	Indicates the com.ibm.websphere.objectgrid.security.plugins.builtins. WSTokenCredentialGenerator implementation class, so the client retrieves the security tokens from the thread.
-credentialGeneratorProps	Specifies the properties for the CredentialGenerator implementation class. The properties are sent to the object with the setProperties(String) method. The credential generator properties value is used only when a value is specified for the Credential generator class field. Properties for **`com.ibm.websphere.objectgrid.security.plugins.UserPasswordCredentialGenerator`** includes userid_password which can be defined as “userid password”. Note: Because parsing of the userid_password property depends upon the space character as the value separator, userids and passwords which contain spaces must use the “`\20`” escape character to represent a space. For example: If the userid is “`Test User Id`” and the password is “`Test Password`”, the userid_password property should be entered as: “`Test\20User\20Id Test\20Password`”. Properties for **`com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredentialGenerator`** includes the property subject_type, which can be defined as either“`runAs`” or “`caller`”.

Return value:

Batch mode example usage

- Using Jacl:

```
$AdminTask modifyXSDomain {-name TestDomain -default true -modifyEndpoints
{{xhost1.ibm.com "" ,2809}} -addEndpoints {{xhost2.ibm.com "" ,2809}}}
-removeEndpoints {{xhost3.ibm.com}}}
```

- Using Jython string:

```
AdminTask.modifyXSDomain('[-name TestDomain
-default false -modifyEndpoints [[xhost1.ibm.com "" ,2809]]
-addEndpoints [[xhost3.ibm.com "" ,2809]]
-removeEndpoints [[xhost2.ibm.com]]]')
```

- Using the client security specification during the modify command:

```
$AdminTask modifyXSDomain {-name myDomain -default false
-configureClientSecurity {-securityEnabled true -
Supported -authenticationRetryCount 1 -credentialGeneratorClass
com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredentialGenerator
 -credentialGeneratorProps "manager manager1"}}
```

Interactive mode example usage

- Using Jacl:

```
$AdminTask modifyXSDomain {-interactive}
```

- Using Jython string:

```
AdminTask.modifyXSDomain ('[-interactive]')
```

testXSDomainConnection

The **testXSDomainConnection** command tests the connection to a catalog service domain.

Required parameters:

-name
Specifies the name of the catalog service domain to which to test the connection.

Optional parameters

-timeout
Specifies the maximum amount of time to wait for the connection, in seconds.

Return value: If a connection can be made, returns `started`, otherwise, returns `stopped`.

Batch mode example usage

- Using Jacl:

```
$Admintask testXSDomainConnection
```

- Using Jython string:

```
AdminTask.testXSDomainConnection
```

Interactive mode example usage

- Using Jacl:

```
$AdminTask testXSDomainConnection {-interactive}
```

- Using Jython string:

```
AdminTask.testXSDomainConnection ('[-interactive]')
```

testXSServerConnection

The **testXSServerConnection** command tests the connection to a catalog server. This command works for both stand-alone servers and servers that are a part of a catalog service domain.

Required parameters:

host
Specifies the host on which the catalog server resides.
listenerPort
Specifies the listener port for the catalog server.

Optional parameters

timeout
Specifies the maximum amount of time to wait for a connection to the catalog server, in seconds.
domain
Specifies the name of a catalog service domain. If you define a value for this parameter, the client security properties for the specified catalog service domain are used to test the connection. Otherwise, a search occurs to find the catalog service domain for the specified host and listener port. If a catalog service domain is found, the client security properties that are defined for the catalog service domain are used to test the server. Otherwise, no client security properties are used during the test.

Return value: If a connection can be made, returns `started`, otherwise returns `stopped`.

Batch mode example usage

- Using Jacl:

```
$Admintask testXSServerConnection {-host xhost1.ibm.com -listenerPort 2809}
```

- Using Jython string:

```
AdminTask.testXSServerConnection('[-host xshost3.ibm.com -listenerPort 2809]')
```

Interactive mode example usage

- Using Jacl:

```
$AdminTask testXSServerConnection {-interactive}
```

- Using Jython string:

```
AdminTask.testXSServerConnection ('[-interactive]')
```

Related concepts:
Example: Configuring catalog service domains
Installation topologies
Catalog service
High availability catalog service
Related tasks:
Configuring WebSphere eXtreme Scale with WebSphere Application Server
Configuring the catalog service in WebSphere Application Server
Creating catalog service domains in WebSphere Application Server
Starting and stopping servers in a WebSphere Application Server environment

Catalog service domain collection

Use this page to manage catalog service domains. Catalog service domains define a group of catalog servers that manage the placement of shards and monitors the health of container servers in your data grid.

To view this administrative console page, click System administration > WebSphere eXtreme Scale > Catalog service domains. To create a new catalog service domain, click New. To delete a catalog service domain, select the catalog service domain you want to remove and click Delete.

Test Connection

When you click the Test connection button, all of the defined catalog service domain end points are queried one by one, if any one end point is available, returns a message that indicates that the connection to the catalog service domain was successful. You can use this button to test that you have configured the connection and security information correctly.

Set Default

Defines the catalog service domain that is used as the default. Select one catalog service domain as the default and click Set default. Only one catalog service domain can be selected as the default.

Name

Specifies the name for the catalog service domain.

Default

Specifies which catalog service domain in the list is the default. The default catalog service domain is indicated with the following icon:  .

Catalog service domain settings

Use this page to manage the settings for a specific catalog service domain. Catalog service domains define a group of catalog servers that manage the placement of shards and monitors the health of container servers in your data grid. You can define a catalog service domain that is in the same cell as your deployment manager. You can also define remote catalog service domains if your WebSphere® eXtreme Scale configuration is in a different cell or your data grid is made up of Java™ SE processes.

To view this administrative console page, click System administration > WebSphere eXtreme Scale > Catalog service domains > *catalog_service_domain_name*.

Related tasks:
Configuring eXtreme Scale connection factories
Configuring catalog servers and catalog service domains
Related reference:
Client properties file

Test connection

When you click the Test connection button, all of the defined catalog service domain end points are queried one by one, if any one end point is available, returns a message that indicates that the connection to the catalog service domain was successful. You can use this button to test that you have configured the connection and security information correctly.

Name

Specifies the name of the catalog service domain.

Enable this catalog service domain as the default unless another catalog service domain is explicitly specified

If you select this check box, the selected catalog service domain becomes the default catalog service domain for the cell. Each server profile in the cell that is augmented with the WebSphere eXtreme Scale profile belongs to the selected catalog service domain.

For WebSphere eXtreme Scale, all eXtreme Scale containers that are embedded in Java EE application modules connect to the default domain. Clients can connect to the default domain using the ServerFactory.getServerProperties().getCatalogServiceBootstrap() API to retrieve the catalog service endpoints to use when calling the ObjectGridManager.connect() API.

If you change the default domain to point to a different set of catalog servers, then all containers and clients refer to the new domain after they are restarted.

Catalog servers

Specifies a list of catalog servers that belong to this catalog service domain.

Click New to add a catalog server to the list. This catalog server must already exist in the eXtreme Scale configuration. You can also edit or delete a server from the list by selecting the endpoint and then clicking Edit or Delete. Define the following properties for each catalog server endpoint:

Catalog server endpoint
: Specifies the name of the existing application server or remote server on which the catalog service is running. A catalog service domain cannot contain a mix of existing application servers and remote server endpoints.

  - Existing application server: Specifies the path of an application server, node agent, or deployment manager in the cell. A catalog service starts automatically in the selected server. Select from the list of the existing application servers. All of the application servers that you define within the catalog service domain must be in the same core group.
  - Remote server: Specifies the host name of the remote catalog server.
    For WebSphere eXtreme Scale remote endpoints: Specifies the host name of the remote catalog server process. You must start the remote servers with the **startOgServer** script or the embedded server API.

Client Port
: Specifies the port that is used for communication between the catalog servers in the catalog service domain. This value is required for catalog servers that are running in WebSphere Application Server processes. You can set the value to any port that is not being used by another process.

Listener Port
: Specifies the port that is used for communication with clients. This value is required for remote endpoints and must match the value used when the catalog service was started. The listener port is used by clients and containers to communicate with the catalog service.
For WebSphere eXtreme Scale remote endpoints: Defines the Object Request Broker (ORB) listener port for containers and clients to communicate with the catalog service through the ORB. For WebSphere Application Server endpoints, the listener port value is inherited from the BOOTSTRAP_ADDRESS port configuration.

Status

Table 1. Catalog server endpoint status

Icon	Definition
	Unknown
	Started
	Stopped

Client security properties

Use this page to configure client security for a catalog service domain. These settings apply to all the servers in your catalog service domain. These properties can be overridden by specifying a splicer.properties file with the com.ibm.websphere.xs.sessionFilterProps custom property or by splicing the application EAR file.

To view this administrative console page, click System administration > WebSphere eXtreme Scale > Catalog service domains > *catalog_service_domain_name* > Client security properties.

Enable client security

Specifies that client security is enabled for the catalog server. The server properties file that is associated with the selected catalog server must have a matching securityEnabled setting in the server properties file. If these settings do not match, an exception results.

Credential authentication

Indicates if credential authentication is enforced or supported.

Never
: No client credential authentication is enforced.

Required
: Credential authentication is always enforced. If the server does not support credential authentication, the client cannot to connect to the server.

Supported
: Credential authentication is enforced only if both the client and server support credential authentication.

Authentication retry count

Specifies the number of times that authentication gets tried again if the credential is expired.

If you do not want to try authentication again, set the value to `0`.

Credential generator class

Indicates the com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator implementation class, so the client retrieves the credential from the CredentialGenerator object.

You can choose from two predefined credential generator classes, or you can specify a custom credential generator. If you choose a custom credential generator, you must indicate the name of the credential generator class.

- com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredentialGenerator
- com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredentialGenerator
- Custom credential generator

Subject type

Specifies if you are using the J2EE caller or the J2EE runAs subject type. You must specify this value when you choose the WSTokenCredentialGenerator credential generator.

- **runAs**: The subject contains the principal of the J2EE run as identity and the J2EE run as credential.
- **caller**: The subject contains the principal of the J2EE caller and the J2EE caller credential.

User ID

Specify a user ID when you are using the UserPasswordCredentialGenerator credential generator implementation.

Password

Specify a password when you are using the UserPasswordCredentialGenerator credential generator implementation.

Credential generator properties

Specifies the properties for a custom CredentialGenerator implementation class. The properties are set in the object with the setProperties(String) method. The credential generator properties value is used only when a value is specified for the Credential generator class field.

Catalog service domain custom properties

You can further edit the configuration of the catalog service domain by defining custom properties.

To view this administrative console page, click System administration > WebSphere eXtreme Scale > Catalog service domains > Custom properties. To create a new custom property, click New.

Name

Specifies the name of the custom property for the catalog service domain.

Value

Specifies a value for the custom property for the catalog service domain.

Configuring the quorum mechanism

Configure the quorum mechanism for each catalog server. You must enable the quorum mechanism on all of the catalog servers in the catalog service domain. Changing the quorum configuration requires a restart.

Before you begin

The configuration must support the following requirements:

- **IP configuration:** Any addressable element on the network must be able to connect to any other addressable element on the network unimpeded. All the firewalls in the configuration must allow all traffic to flow between the IP addresses and ports that are being used to host catalog servers and container servers.

About this task

For more information about the quorum mechanism, see Catalog server quorums.

Changing the quorum configuration requires that you restart the catalog server. You can choose one of the following processes to enable the quorum configuration on the catalog servers in your catalog service domain:

- Stop the entire catalog service domain, change the setting, and restart the catalog service domain.
- Stop and start one catalog service at a time. With this process, some catalog servers have quorum enabled while other catalog servers do not. If container server outages occur during this process, container server lifecycle events occur only if the primary catalog server does not have quorum yet, or when you override quorum.

Procedure

Enable quorum on the catalog servers. In WebSphere® Application Server, you must configure quorum with the server properties file. In a stand-alone environment you can either use the properties method or enable quorum when you start the server:

- **Set the `enableQuorum=true` property in the server properties file.**
  You can use this configuration in a WebSphere Application Server or stand-alone environment. See the following example objectGridServer.properties file:

  ```
  catalogClusterEndPoints=cat0:cat0.domain.com:6600:6601,
  cat1:cat1.domain.com:6600:6601
  catalogServiceEndPoints= cat0.domain.com:2809, cat1.domain.com:2809
  enableQuorum=true
  ```

  For more information about configuring the properties file, see Server properties file.
- **Pass the `-quorum` enabled flag on thestartOgServer command.**
  You can use this configuration method when you start stand-alone servers only.

  ```
  # bin/startOgServer cat0 -serverProps objectGridServer.properties -quorum true
  ```

  For more information about the **startOgServer** command, see Starting and stopping stand-alone servers.

Results

By enabling the quorum mechanism on the catalog servers within a catalog service domain, all the catalog servers must be available for data grid placement operations to occur. If a short network brownout occurs, placement operations are temporarily stopped until all the catalog servers in the quorum are available. To verify the quorum settings, run the xscmd command **showQuorumStatus**.
You can add catalog servers to the quorum by repeating these steps.

What to do next

For more information about dealing with outages and overriding the quorum mechanism, see Managing data center failures when quorum is enabled.
Related concepts:
High availability
Core groups
High availability catalog service
Catalog server quorums
Replication for availability
Installation topologies
Catalog service
Related tasks:
Managing data center failures
Managing data center failures when quorum is enabled
Administering with the xscmd utility
Related reference:
Server properties file
startOgServer script
showQuorumStatus command

Tuning the heartbeat interval setting for failover detection

You can configure the amount of time between system checks for failed servers with the heartbeat interval setting. This setting applies to catalog servers only.

About this task

Configuring failover depends on the type of environment you are using. If you are using a stand-alone environment, you can configure failover with the command line. If you are using a WebSphere® Application Server Network Deployment environment, you must configure failover in the WebSphere Application Server Network Deployment administrative console.

Procedure

- Configure failover for stand-alone environments.
  - With the -heartbeat parameter in the **startOgServer** script when you start the catalog server.
  - With the heartBeatFrequencyLevel property in the server properties file for the catalog server.

  Use one of the following values:

Table 1. Valid heartbeat values

Value	Action	Description
-1	Aggressive	Specifies an aggressive heartbeat level. With this value, failures are detected more quickly, but more processor and network resources are used. This level is more sensitive to missing heartbeats when the server is busy. Failovers are typically detected within 5 seconds.

Value	Action	Description
0	Typical (default)	Specifies a heartbeat level at a typical rate. With this value, failover detection occurs at a reasonable rate without overusing resources. Failovers are typically detected within 30 seconds.
1	Relaxed	Specifies a relaxed heartbeat level. With this value, a decreased heartbeat frequency increases the time to detect failures, but also decreases processor and network use. Failovers are typically detected within 180 seconds.

An aggressive heartbeat interval can be useful when the processes and network are stable. If the network or processes are not optimally configured, heartbeats might be missed, which can result in a false failure detection.

- Configure failover for WebSphere Application Server environments.
  You can configure WebSphere Application Server Network Deployment Version 6.1 and later to allow WebSphere eXtreme Scale to fail over very quickly. The default failover time for hard failures is approximately 200 seconds. A hard failure is a physical computer or server crash, network cable disconnection or operating system error. Failures because of process crashes or soft failures typically fail over in less than one second. Failure detection for soft failures occurs when the network sockets from the dead process are closed automatically by the operating system for the server hosting the process.

  Core group heartbeat configuration

  WebSphere eXtreme Scale running in a WebSphere Application Server process inherits the failover characteristics from the core group settings of the application server. The following sections describe how to configure the core group heartbeat settings for different versions of WebSphere Application Server Network Deployment:

  - **Update the core group settings for WebSphere Application Server Network Deployment Version 6.1 and 7.0:**
    Specify the heartbeat interval in seconds on WebSphere Application Server versions from Version 6.0 through Version 6.1.0.12 or in milliseconds starting with Version 6.1.0.13. You must also specify the number of missed heartbeats. This value indicates how many heartbeats can be missed before a peer Java™ virtual machine (JVM) is considered as failed. The hard failure detection time is approximately the product of the heartbeat interval and the number of missed heartbeats.

    These properties are specified using custom properties on the core group using the WebSphere administrative console. See Core group custom properties for configuration details. These properties must be specified for all core groups used by the application:
    - The heartbeat interval is specified using either the IBM_CS_FD_PERIOD_SEC custom property for seconds or the IBM_CS_FD_PERIOD_MILLIS custom property for milliseconds (requires Version 6.1.0.13 or later).
    - The number of missed heartbeats is specified using the IBM_CS_FD_CONSECUTIVE_MISSED custom property.

    The default value for the IBM_CS_FD_PERIOD_SEC property is 20 and for the IBM_CS_FD_CONSECUTIVE_MISSED property is 10. If the IBM_CS_FD_PERIOD_MILLIS property is specified, then it overrides any of the set IBM_CS_FD_PERIOD_SEC custom properties. The values of these properties are positive integer values.

    Use the following settings to achieve a 1500 ms failure detection time for WebSphere Application Server Network Deployment Version 6.x servers:
    - Set `IBM_CS_FD_PERIOD_MILLIS = 750` (WebSphere Application Server Network Deployment V6.1.0.13 and later)
    - Set `IBM_CS_FD_CONSECUTIVE_MISSED = 2`
  - **Update the core group settings for WebSphere Application Server Network Deployment Version 7.0**

    WebSphere Application Server Network Deployment Version 7.0 provides two core group settings that can be adjusted to increase or decrease failover detection:

    - **Heartbeat transmission period.** The default is 30000 milliseconds.
    - **Heartbeat timeout period.** The default is 180000 milliseconds.

    For more details on how change these settings, see the WebSphere Application Server Network Deployment Information center: Discovery and failure detection settings.

    Use the following settings to achieve a 1500 ms failure detection time for WebSphere Application Server Network Deployment Version 7 servers:

    - Set the heartbeat transmission period to 750 milliseconds.
    - Set the heartbeat timeout period to 1500 milliseconds.

What to do next

When these settings are modified to provide short failover times, there are some system-tuning issues to be aware of. First, Java is not a real-time environment. It is possible for threads to be delayed if the JVM is experiencing long garbage collection times. Threads might also be delayed if the machine hosting the JVM is heavily loaded (due to the JVM itself or other processes running on the machine). If threads are delayed, heartbeats might not be sent on time. In the worst case, they might be delayed by the required failover time. If threads are delayed, false failure detections occur. The system must be tuned and sized to ensure that false failure detections do not happen in production. Adequate load testing is the best way to ensure this.

Note: The current version of eXtreme Scale supports WebSphere Real Time.
Related concepts:
High availability
Core groups
High availability catalog service
Catalog server quorums
Replication for availability
Installation topologies
Catalog service
Related reference:
Server properties file
startOgServer script

Configuring container servers

The container server stores application data for the data grid. This data is generally broken into parts, which are called partitions, which are hosted across multiple container servers. Each container server in turn hosts a subset of the complete data.

About this task

- **Stand-alone container servers**:
  Configure stand-alone container servers with a server properties file and a deployment policy XML file. Control the life cycle of a container server with the start and stop scripts or with the embedded server API.
- **Container servers that start in WebSphere® Application Server:**
  Configure container servers in WebSphere Application Server with a server properties file and deployment policy XML file that is embedded into a Java EE application module. The life cycle of the container servers is controlled by the application. Container servers start and stop with the application.
- Container server reconnect properties
  Use Java virtual machine (JVM) properties to configure how your container server reconnects to the data grid if the container server becomes disconnected.
- Configuring container servers in WebSphere Application Server
  Configure container servers in WebSphere Application Server by using a server properties file and deployment policy XML file that is embedded into a Java EE application module. Container servers stop and start when the application is stopped and started.

Container server reconnect properties

Use Java™ virtual machine (JVM) properties to configure how your container server reconnects to the data grid if the container server becomes disconnected.

JVM system properties

If a container server becomes disconnected from the data grid, WebSphere® eXtreme Scale attempts to reconnect those container servers. By setting system properties, you can control how the container reconnects. You can set these properties when you start a container server. Some properties are applicable to WebSphere eXtreme Scale in a stand-alone environment, while others are only applicable in an integrated WebSphere Application Server environment. For example, when a container server is started in a stand-alone environment, you can set these properties as an option from a command console:

```
startOgServer.sh server01 -objectgridFile objectgrid.xml -deploymentPolicyFile deployment.xml -
Dcom.ibm.websphere.objectgrid.container.reconnect.restart=false
```

For more information, see Starting and stopping stand-alone servers. If you want to set the appropriate property for WebSphere eXtreme Scale for WebSphere Application Server, you can use the WebSphere Integrated Solutions Console tool. This tool is a graphical user interface that is accessible from the WebSphere Application Server environment, and is installed as an extension to the WebSphere ISC.

`com.ibm.websphere.objectgrid.container.reconnect.block.reconnect.time`
: Defines the amount of time (in milliseconds) to block another container reconnect call. Only valid when a container server is started for the product offering:WebSphere eXtreme Scale for WebSphere Application Server.
  Default: 30000 milliseconds

`com.ibm.websphere.objectgrid.container.reconnect.min.successful.heartbeats`
: Defines the minimum number of successful heartbeats before a container can be stopped. Only valid when a container server is started for the product offering: WebSphere eXtreme Scale for WebSphere Application Server.
  Default: 10

`com.ibm.websphere.objectgrid.container.reconnect.restart`
: Defines whether container reconnect can restart the JVM. Only valid when a container server is started for WebSphere eXtreme Scale in a stand-alone environment.
  Default: true

`com.ibm.websphere.objectgrid.container.reconnect.restart.command`
: Defines the command that is used to restart the container. This command overrides the dynamically generated command if this property is not specified. Each argument in the command is delimited by the value that is defined in the property 'com.ibm.websphere.objectgrid.container.reconnect.restart.delimiter'. Only for use with OSGi container servers. The following is an example of a valid command that uses the `','` as a delimiter.
  Default: `" "` An empty string identifies not to restart a container server on OSGi.

  Example: `/usr/bin/java,-Dprop=value,SomeClass,arg1,arg2`

  You must specify this property if you want OSGi containers to restart when the conditions for container restart be met. Otherwise, the container server is stopped and the JVM continues to run, including any other OSGi services within that JVM.

`com.ibm.websphere.objectgrid.container.reconnect.restart.delimiter`
: Defines the delimiter that is used for tokenizing the command that is specified in the property 'com.ibm.websphere.objectgrid.container.reconnect.restart.command'. A blank space is not to be used as a delimiter as parsing issues can result. Only for use with OSGi container servers.
  Default: `","` A comma is the default delimiter.

`com.ibm.websphere.objectgrid.container.reconnect.restart.delay`

Defines the time (in milliseconds) to delay before you proceed with the startup on the newly created child container when the JVM is restarted. Only valid when a container server is started for the product offering: WebSphere eXtreme Scale in a stand-alone environment.
Default: 2000 milliseconds

`com.ibm.websphere.objectgrid.container.reconnect.restart.parent.timeout`
Defines the time (in milliseconds) for the newly created child container to wait for its parent to disconnect before timing out when the JVM is restarted. Only valid when a container server is started for the product offering: WebSphere eXtreme Scale in a stand-alone environment.
Default: 180000 milliseconds

`com.ibm.websphere.objectgrid.container.reconnect.retry.forever`
Defines whether the container attempts to reconnect to the container server forever. Only valid when a container server is started for the product offering:WebSphere eXtreme Scale for WebSphere Application Server.
Default: false

Configuring container servers in WebSphere Application Server

Configure container servers in WebSphere® Application Server by using a server properties file and deployment policy XML file that is embedded into a Java™ EE application module. Container servers stop and start when the application is stopped and started.

Before you begin

Configure a catalog service domain. See Creating catalog service domains in WebSphere Application Server for more information.

About this task

To create container servers in WebSphere Application Server, you must embed the WebSphere eXtreme Scale configuration XML files to create the container servers within the application module.

Procedure

1. Identify the application servers on which you want to deploy the Java EE application that contains the WebSphere eXtreme Scale container server definitions. Verify that the target application server profiles have been augmented with the WebSphere eXtreme Scale profile. In a production environment, do not collocate the servers that you use for container servers with the catalog servers. See Creating and augmenting profiles for WebSphere eXtreme Scale for more information.
2. Configure a server properties file and add the server properties file to the class path for each target application server node. See Server properties file for more information.
3. Add the ObjectGrid descriptor XML file and deployment policy XML file to the application module. See Configuring WebSphere Application Server applications to automatically start container servers for more information.

- Configuring WebSphere Application Server applications to automatically start container servers
  Container servers in a WebSphere Application Server environment start automatically when a module starts that has the eXtreme Scale XML files included.

Related concepts:
Installation topologies
Catalog service
Container servers, partitions, and shards
Related tasks:
Configuring the catalog service in WebSphere Application Server
Configuring WebSphere Application Server applications to automatically start container servers
Controlling placement
Related reference:
Deployment policy descriptor XML file
ObjectGrid descriptor XML file
Server properties file

Configuring WebSphere Application Server applications to automatically start container servers

Container servers in a WebSphere® Application Server environment start automatically when a module starts that has the eXtreme Scale XML files included.

Before you begin

WebSphere Application Server and WebSphere eXtreme Scale must be installed, and you must be able to access the WebSphere Application Server administrative console.

About this task

Java™ Platform, Enterprise Edition applications have complex class loader rules that greatly complicate loading classes when using a shared data grid within a Java EE server. A Java EE application is typically a single Enterprise Archive (EAR) file. The EAR file contains one or more deployed Enterprise JavaBeans (EJB) or web archive (WAR) modules.

WebSphere eXtreme Scale watches for each module start and looks for eXtreme Scale XML files. If the catalog service detects that a module starts with the XML files, the application server is registered as a container server Java virtual machine (JVM). By registering the container servers with the catalog service, the same application can be deployed in different data grids, but used as a single data grid by the catalog service. The catalog service is not concerned with cells, grids, or dynamic grids. A single data grid can span multiple cells if required.

Procedure

1. Package your EAR file to have modules that include the eXtreme Scale XML files in the META-INF folder. WebSphere eXtreme Scale detects the presence of the objectGrid.xml and objectGridDeployment.xml files in the META-INF folder of EJB and WEB modules when they start. If only an objectGrid.xml file is found, then the JVM is assumed to be client. Otherwise, it is assumed this JVM acts as a container for the data grid that is defined in the objectGridDeployment.xml file.
   You must use the correct names for these XML files. The file names are case-sensitive. If the files are not present, then the container does not start. You can check the systemout.log file for messages that indicate that shards are being placed. An EJB module or WAR module using eXtreme Scale must have eXtreme Scale XML files in its META-INF directory.

   The eXtreme Scale XML files include:
   - An ObjectGrid descriptor XML file, named objectGrid.xml. See ObjectGrid descriptor XML file for more information.
   - A deployment descriptor XML file named objectGridDeployment.xml. See Deployment policy descriptor XML file for more information.
   - (Optional) An entity metadata descriptor XML file, if entities are used. The entity.xml file name must match the name that is specified in the objectGrid.xml file. See Entity metadata descriptor XML file for more information.

   The run time detects these files, then contacts the catalog service to inform it that another container is available to host shards for that eXtreme Scale.
   Tip: If your application has entities and you are planning to use one container server, set the minSyncReplicas value to `0` in the deployment descriptor XML file. Otherwise, you might see one of the following messages in the SystemOut.log file because placement cannot occur until another server starts to meet the minSyncReplica policy:

   ```
   CWPRJ1005E: Error resolving entity association. Entity=entity_name,
   association=association_name.
   ```

   ```
   CWOBJ3013E: The EntityMetadata repository is not available.  Timeout
   threshold reached when trying to register the entity: entity_name.
   ```

2. Deploy and start your application.
   The container starts automatically when the module is started. The catalog service starts to place partition primaries and replicas (shards) as soon as possible. This placement occurs immediately unless you configure the environment to delay placement. For more information, see Controlling placement.

What to do next

Applications within the same cell as the containers can connect to these data grids by using a ObjectGridManager.connect(null, null) method and then call the getObjectGrid(ccc, "object grid name") method. The connect or getObjectGrid methods might be blocked until the containers have placed the shards, but this blocking is only an issue when the data grid is starting.

ClassLoaders

Any plug-ins or objects stored in an eXtreme Scale are loaded on a certain class loader. Two EJB modules in the same EAR can include these objects. The objects are the same but are loaded with different ClassLoaders. If application A stores a Person object in a map that is local to the server, application B receives a `ClassCastException` if it tries to read that object. This exception occurs because application B loaded the Person object on a different class loader.

One approach to resolve this problem is to have a root module contain the necessary plug-ins and objects that are stored in the eXtreme Scale. Each module that uses eXtreme Scale must reference that module for its classes. Another resolution is to place these shared objects in a utility JAR file that is on a common class loader shared by both modules and applications. The objects can also be placed in the WebSphere classes or lib/ext directory, however this placement complicates the deployment.

EJB modules in an EAR file typically share the same ClassLoader and are not affected by this problem. Each WAR module has its own ClassLoader and is affected by this problem.

Connecting to a data grid client-only

If the catalog.services.cluster property is defined in the cell, node or server custom properties, any module in the EAR file can call the ObjectGridManager.connect (ServerFactory.getServerProperties().getCatalogServiceBootstrap(), null, null) method to get a ClientClusterContext. The module can also call the ObjectGridManager.getObjectGrid(ccc, "grid name") method to gain a reference to the data grid. If any application objects are stored in Maps, verify that those objects are present in a common ClassLoader.

Java clients or clients outside the cell can connect with the bootstrap IIOP port of the catalog service. In WebSphere Application Server, the deployment manager hosts the catalog service by default. The client can then obtain a ClientClusterContext and the named data grid.

Entity manager

With the entity manager, the tuples are stored in the maps instead of application objects, resulting in fewer class loader problems. Plug-ins can be a problem, however. Also note that a client override ObjectGrid descriptor XML file is always required when connecting to a data grid that has entities defined: ObjectGridManager.connect("host:port[,host:port], null, objectGridOverride) or ObjectGridManager.connect(null, objectGridOverride).

Related concepts:
Installation topologies
Catalog service
Container servers, partitions, and shards

Related tasks:
Configuring the catalog service in WebSphere Application Server
Configuring container servers in WebSphere Application Server
Controlling placement
Starting and stopping secure servers
Related reference:
Deployment policy descriptor XML file
Configuring distributed deployments
ObjectGrid descriptor XML file
Server properties file

Configuring multiple data center topologies

With the multi-master asynchronous replication, you link a set of catalog service domains. The connected catalog service domains are then synchronized using replication over the links. You can define the links using properties files, at run time with Java Management Extensions (JMX) programs, or with command-line utilities. The set of current links for a domain is stored in the catalog service. You can add and remove links without restarting the catalog service domain that hosts the data grid.

Before you begin

- See Planning multiple data center topologies for more information about multi-master replication topologies and design considerations. You can configure links among catalog service domains with the server properties file to form the topology during server startup. You can also configure links at run time.
- If you are using loaders in your multi-master replication topology, you must plan how you are going to maintain accurate data between the data centers. The approaches that you can use vary depending on the topology you are using. For more information, see Loader considerations in a multi-master topology.

Procedure

- Define links in the server properties file for the catalog server of each catalog service domain in the topology, for bootstrap purposes.
  See Server properties file for more information about defining this file for the catalog server.

  Important: Property names are case-sensitive.

  Local Domain name:
  Specify the name of the catalog service domain for the current catalog server:

  ```
  domainName=domain1
  ```

  An optional list of foreign domain names:
  Specify the names of catalog service domains to which you want to link in the multi-master replication topology:

  ```
  foreignDomains=domain2,domain3,domain4
  ```

  An optional list of endpoints for the foreign domain names:
  Specifies the connection information for the catalog servers of the foreign domains:

  ```
  domain2.endPoints=hostB1:2809, hostB2:2809
  ```

  If a foreign domain has multiple catalog servers, specify all of them.

- Use the **xscmd** utility or JMX programming to add or remove links at run time.
  The links for a domain are kept in the catalog service in replicated memory. This set of links can be changed at any time by the administrator without requiring a restart of this domain or any other domain. The **xscmd** utility includes several options for working with links.

  The **xscmd** utility connects to a catalog service and thus a single catalog service domain. Therefore, the **xscmd** utility can be used to create and destroy links between the domain it attaches to and any other domain.

  Use the command line to create a link, for example:

  ```
  xscmd -c establishLink -cep host:2809 -fd dname -fe fdHostA:2809,fdHostB:2809
  ```

  The command establishes a new link between the local domain and the foreign domain named `dname`. The `dname` catalog service is running at `fdHostA:2809` and `fdHostB:2809`. The local catalog service domain has a catalog service listener host and port of `host:2809`. Specify all catalog service endpoints from the foreign domain so that fault tolerance connectivity to the domain is possible. Do not use a single `host:port` pair for the catalog service of the foreign catalog service domain.

  You can use any local catalog service JVM with **xscmd** and the using the -cep option. If the catalog server is hosted in a WebSphere Application Server deployment manager, then the port is usually 9809.

  The ports specified for the foreign domain are not JMX ports. They are the usual ports you would use for eXtreme Scale clients.

  After the command to add a new link is issued, the catalog service instructs all containers under its management to begin replicating to the foreign domain. A link is not needed on both sides. It is only necessary to create a link on one side.

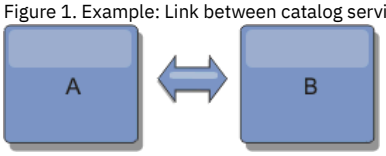
  Use the command line to remove a link, for example:

  ```
  xscmd -c dismissLink -cep host:2809 -fd dname
  ```

The command connects to the catalog service for a domain and instructs it to stop replicating to a specific domain. A link needs to be dismissed from one side only.
Attention: You can run the establish or dismiss link commands multiple times. If the link does not enter the correct status or is disjoint, run the command again.

Examples

Figure 1. Example: Link between catalog service domains



Suppose that you want to configure a two-domain setup involving catalog service domains A and B.
Here is the server properties file for the catalog server in domain A:

```
domainName=A
foreignDomains=B
B.endPoints=hostB1:2809, hostB2:2809
```

Here is the server properties file for the catalog server in domain B. Notice the similarity between the two property files.

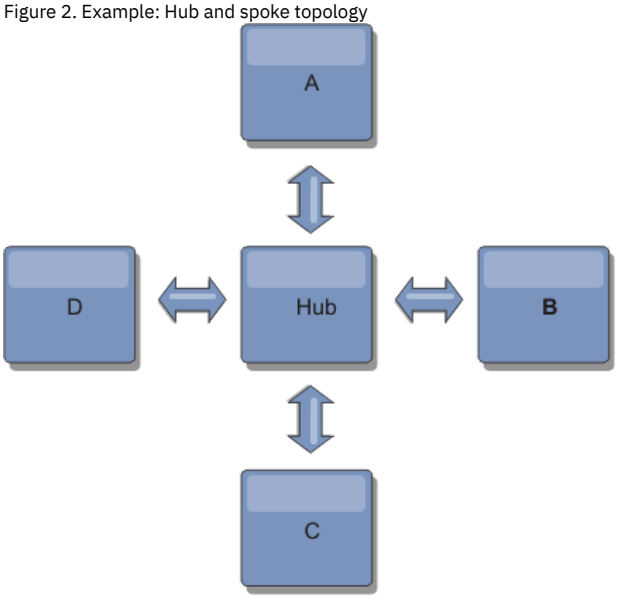
```
domainName=B
foreignDomains=A
A.endPoints=hostA1:2809,hostA2:2809
```

After the two domains are started, then any data grids with the following characteristics are replicated between the domains.

- Has a private catalog service with a unique domain name
- Has the same data grid name as other grids in the domain
- Has the same number of partitions as other data grids in the domain
- Has a data grid that uses the fixed partition placement strategy. A data grid that is configured to use a per container placement strategy cannot be replicated.
- Has the same number of partitions (it might or might not have the same number and types of replicas)
- Has the same data types being replicated as other data grids in the domain
- Has the same map set name, map names, and dynamic map templates as other data grids in the domain

The replication policy of a catalog service domain is ignored.
The preceding example shows how to configure each domain to have a link to the other domain, but it is necessary only to define a link in one direction. This fact is especially useful in hub and spoke topologies, allowing a much simpler configuration. The hub property file does not require updates as spokes are added, and each spoke file needs only to include hub information. Similarly, a ring topology requires each domain to have only a link to the previous and next domain in the ring.

Figure 2. Example: Hub and spoke topology



The hub and four spokes (domains A, B, C, and D) has server property files like the following examples.

```
domainName=Hub
```

Spoke A has the following server properties:

```
domainName=A
foreignDomains=Hub
Hub.endPoints=hostH1:2809, hostH2:2809
```

Spoke B has the following server properties:

```
domainName=B
foreignDomains=Hub
Hub.endPoints=hostH1:2809, hostH2:2809
```

Spoke C has the following server properties:

```
domainName=C
foreignDomains=Hub
Hub.endPoints=hostH1:2809, hostH2:2809
```

Spoke D has the following properties:

```
domainName=D
foreignDomains=Hub
Hub.endPoints=hostH1:2809, hostH2:2809
```

What to do next

- If you need to check or troubleshoot problems with the links between your catalog service domains, you can use the **xscmd** utility. For more information about commands to help you troubleshoot your multiple data center configuration, see Troubleshooting multiple data center configurations.
- You can provide a custom collision arbiter to resolve collisions between the catalog service domains. See Developing custom arbiters for multi-master replication for more information.

Related concepts:
Planning multiple data center topologies
Topologies for multi-master replication
Configuration considerations for multi-master topologies
Design considerations for multi-master replication
Loader considerations in a multi-master topology
Configuration considerations for multi-master topologies
8.5+ Installing fix packs using IBM Installation Manager
Related tasks:
Developing custom arbiters for multi-master replication
Retrieving eXtreme Scale environment information with the xscmd utility
Updating eXtreme Scale servers
Migrating to WebSphere eXtreme ScaleVersion 8.5
Starting and stopping stand-alone servers
Starting and stopping servers in a WebSphere Application Server environment
Troubleshooting multiple data center configurations
Related information:
dW Improve response time and data availability with WebSphere eXtreme Scale multi-master capability

Configuring ports

You must open ports to communicate among servers and with remote servers.

- Configuring ports in stand-alone mode
  You can configure the necessary ports for servers and clients in an eXtreme scale deployment by using command-line parameters, property files or programmatically. Most examples included in the following sections describe command-line parameters to the **startOgServer** script. Equivalent configuration options can also be set in properties files, using the embedded server API or the client API.
- Configuring ports in a WebSphere Application Server environment
  WebSphere eXtreme Scale catalog services, container servers and clients, when running in WebSphere Application Server processes, utilize ports and services already defined for the process.
- Servers with multiple network cards
  You can run eXtreme Scale processes on a server that has more than one network card.

Related concepts:
Planning for network ports
Servers with multiple network cards

Configuring ports in stand-alone mode

You can configure the necessary ports for servers and clients in an eXtreme scale deployment by using command-line parameters, property files or programmatically. Most examples included in the following sections describe command-line parameters to the **startOgServer** script. Equivalent configuration options can also be set in properties files, using the embedded server API or the client API.

Procedure

1. Start catalog service endpoints.
   WebSphere® eXtreme Scale uses IIOP to communicate between Java™ virtual machines. The catalog service JVMs are the only processes that require the explicit configuration of ports for the IIOP services and group services ports. Other processes dynamically allocate ports.

a. Specify client and peer ports. The client port and peer port are used for communication between catalog services in a catalog service domain. To specify the client port and peer port, use the following command-line option:

-catalogServiceEndPoints <serverName:hostName:clientPort:peerPort>
Specifies a list of catalog servers to link together into a catalog service domain. Each attribute is defined as follows:

serverName
Specifies the name of the catalog server.
hostName
Specifies the host name for the computer where the server is launched.
clientPort
Specifies the port that is used for peer catalog service communication.
peerPort
This value is the same as the haManagerPort. Specifies the port that is used for peer catalog service communication.

The following example starts the cs1 catalog server, which is in the same catalog service domain as the cs2 and cs3 servers:

```
startOgServer.bat|sh cs1 -catalogServiceEndPoints
cs1:MyServer1.company.com:6601:6602,cs2:MyServer2.company.com:6601:6602,cs3:MyServer3.company.com:6601:66
02
```

If you start additional catalog servers, they must include the same servers in the -catalogServiceEndPoints argument. The order of the list can be different, but the servers contained in the list must be the same for each catalog server. Do not put any spaces in the list.

You can also set the catalog service end points with the catalogClusterEndPoints server property.

b. Set the listener host and port. The listener port is used for communication between catalog services in a catalog service domain, and for communication between catalog services and container servers and clients. To specify the listener port and listener host, use the following command-line options:

-listenerHost <host name>
Specifies the host name to which the Object Request Broker (ORB) transport protocol binds for communication. The value must be a fully qualified domain name or IP address. If your configuration involves multiple network cards, set the listener host and port to the IP address for which to bind. By setting the listener and host port, it allows the transport mechanism in the JVM know which IP address to use. If you do not specify which IP address to use, symptoms such as connection timeouts, unusual API failures, and clients that seem to hang can occur.

Default: `localhost`
-listenerPort <port>
Specifies the port number to which the ORB transport protocol binds for communication. **Default:** `2809`

You can also set the listener port and listener host with the listenerHost and listenerPort server properties.

c. Optional: Set the JMX service port.
The JMX service port is used for communication from JMX clients. To specify the JMX service port, use the following command-line option:

-JMXServicePort <port>
Required only for WebSphere eXtreme Scale in a stand-alone environment. Specifies the port number on which the MBean server listens for communication with Java Management Extensions (JMX).
Default: `1099` for catalog servers

You can also set the JMX service port with the JMXServicePort server property.

d. Optional: Set the JMX connector port.
The JMX connector port is used for communication from JMX clients. To specify the JMX connector port, use the following command-line option:

-JMXConnectorPort <port>
Defines the Secure Sockets Layer (SSL) port to which the Java Management Extensions (JMX) service binds. Only required if an SSL transport protocol is needed for JMX data.

You can also set the JMX connector port with the JMXConnectorPort server property.

e. Set the Secure Socket Layer (SSL) port.
When security is enabled, an SSL port is also required. To specify the SSL port, use the following command-line option:

```
-jvmArgs -Dcom.ibm.CSI.SSLPort=<sslPort>
```

Figure 1. Example using the command line. Start the first catalog server on hostA.
An example of the command follows:

```
./startOgServer.sh cs1 -listenerHost hostA -listenerPort 2809
-catalogServiceEndPoints cs1:hostA:6601:6611,cs2:hostB:6601:6611
```

Start the second catalog server on hostB. An example of the command follows:

```
./startOgServer.sh cs2 -listenerHost hostB -listenerPort 2809
-catalogServiceEndPoints cs1:hostA:6601:6611,cs2:hostB:6601:6611
```

2. Start container server endpoints.
The following command starts a container server to use with the example catalog service:

```
./startOgServer.sh c0 -catalogServiceEndPoints hostA:2809,hostB:2809
```

The container server Java virtual machines use two ports. The HA manager port is used for internal communication between peer container servers and catalog servers. The listener port is used for IIOP communication between peer container servers, catalog servers, and clients. The listener host is used to

bind the ORB to a specific network adapter. If you do not specify, both ports are dynamically selected. However, if you want to explicitly configure ports, such as in a firewall environment, you can use a command-line options to specify the ORB port.

a. Specify the listener host and port. To specify the listener port and listener host, use the following command-line options:

-listenerHost <host name>
Specifies the host name to which the Object Request Broker (ORB) transport protocol binds for communication. The value must be a fully qualified domain name or IP address. If your configuration involves multiple network cards, set the listener host and port to the IP address for which to bind. By setting the listener and host port, it allows the transport mechanism in the JVM know which IP address to use. If you do not specify which IP address to use, symptoms such as connection timeouts, unusual API failures, and clients that seem to hang can occur.

Default: `localhost`

-listenerPort <port>
Specifies the port number to which the ORB transport protocol binds for communication. **Default:** `2809`

You can also set listener port and listener host with the listenerHost and listenerPort server properties.

b. Specify the HA manager port. To specify the HA manager port, use the following command-line option:

-haManagerPort <port>
Specifies the port that is used by the high availability (HA) manager for heartbeat communication between peer container servers. The haManagerPort port is only used for peer-to-peer communication between container servers that are in same domain. If the haManagerPort property is not defined, then an ephemeral port is used. In WebSphere Application Server, this setting is inherited by the high availability manager port configuration.

You can also set the HA manager port with the HAManagerPort server property.

c. Optional: Specify the SSL port.
When security is enabled, a Secure Socket Layer (SSL) port is also required. To specify the SSL port, use the following command-line option:

```
-jvmArgs -Dcom.ibm.CSI.SSLPort=<sslPort>
```

d. Optional: Specify the JMX service port.

-JMXServicePort <port>
Required only for WebSphere eXtreme Scale in a stand-alone environment. Specifies the port number on which the MBean server listens for communication with Java Management Extensions (JMX).
Default: `1099` for catalog servers

You can also set the JMX service port with the JMXServicePort server property.

e. Optional: Set the JMX connector port.
The JMX connector port is used for communication from JMX clients. To specify the JMX connector port, use the following command-line option:

-JMXConnectorPort <port>
Defines the Secure Sockets Layer (SSL) port to which the Java Management Extensions (JMX) service binds. Only required if an SSL transport protocol is needed for JMX data.

You can also set the JMX connector port with the JMXConnectorPort server property.

3. Start client endpoints.
Clients must know the catalog service listener end points only. Clients retrieve end points for container server Java virtual machines, which are the Java virtual machines that hold the data, automatically from the catalog service. To connect to the catalog service in the previous example, the client must pass the following list of `host:port` pairs to the connect API:

```
hostA:2809,hostB:2809
```

The client can also receive callbacks from container servers when using the DataGrid API. These callbacks communicate using IIOP with the ORB listener port. To specify the port and network adapter to receive callbacks, set the listenerHost and listenerPort properties in the client properties file.

When security is enabled, a Secure Socket Layer (SSL) port is also required. To specify the SSL port, use the following system property when starting the client process:

```
-jvmArgs -Dcom.ibm.CSI.SSLPort=<sslPort>
```

Related concepts:
Planning for network ports
Servers with multiple network cards

Configuring ports in a WebSphere Application Server environment

WebSphere eXtreme Scale catalog services, container servers and clients, when running in WebSphere Application Server processes, utilize ports and services already defined for the process.

About this task

The following sections explain details relating to using ports in your deployment.

1. Catalog service endpoints
   WebSphere eXtreme Scale catalog services run in any WebSphere Application Server process and are configured using the administrative console or using administrative tasks. All ports are inherited by the process except for the client port, which is explicitly configured. For details on which ports are used by the catalog service, see Planning for network ports. For details on configuring a catalog service domain, see High availability catalog service.

2. Container server endpoints
   WebSphere eXtreme Scale container servers are hosted within Java EE modules. The container servers use the ports defined for the application server process. For details on which ports are used by the container service, see Planning for network ports. For details on starting a container within a Java EE module such as an Enterprise JavaBeans™ (EJB) or Web module, see Configuring WebSphere Application Server applications to automatically start container servers.

3. Client endpoints
   WebSphere eXtreme Scale clients are hosted within Java EE web or EJB modules.

   Clients programmatically connect to the catalog service domain using the ObjectGridManager.connect() API. When connecting to a catalog service domain hosted within the same cell, the client connection will automatically find the default catalog service domain by using the following API call on the ObjectGridManager:

   ```
   connect(securityProps, overRideObjectGridXML)
   ```

   If the default catalog service domain is hosted remotely (external to the cell), the catalog service endpoints must be specified using the following method on the ObjectGridManager API:

   ```
   connect(catalogServerEndpoints, securityProps, overRideObjectGridXml)
   ```

   If the default catalog service domain is defined in the cell, then the CatalogServerProperties API can be used to retrieve the catalog server addresses. The XSDomainManagement administrative task can also be used to retrieve any configured catalog service domain endpoints.

Related concepts:
Planning for network ports
Servers with multiple network cards

Servers with multiple network cards

You can run eXtreme Scale processes on a server that has more than one network card.

If a server or client is running on a server that contains more than one network card, then you must specify the network port and host name in your eXtreme Scale configuration to bind to a specified card. If this configuration is not specified, then the eXtreme Scale runtime automatically chooses a network post and host name, which may result in connection failures or slower performance.

When you are setting the host name for eXtreme Scale processes that are embedded in WebSphere® Application Server, you might need to consider the WebSphere Application Server or other stack products in your configuration. For an example, see Technote: Configuring the node agent on one NIC and its application server on another NIC, which is on a different subnet, can lead to ORB errors .

For catalog or container servers, you must set the listener host and listener port in one of the following ways:

- In the server properties file.
- Command-line parameter on the **startOgServer** script.

For clients, you cannot use the command line, and must use client properties.

Related tasks:
Configuring ports
Configuring ports in stand-alone mode
Configuring ports in a WebSphere Application Server environment

Configuring transports

Transports enable the exchange of objects and data between different server processes in your configuration.

About this task

8.5+ The main transport mechanism is the Object Request Broker (ORB). This mechanism stores cache entries on the Java™ heap. Using the ORB as the transport mechanism is required in the following configuration scenarios:

- When you are using a system other than x86 64-bit Linux.
- When you are using container servers that are running in a WebSphere® Application Server environment.
- When you are using evictor plug-ins or composite indexes.

ORB
When you use the ORB transport, communication between clients and servers, and between servers within the environment, is handled by the ORB.

- Configuring Object Request Brokers
  The Object Request Broker (ORB) is used by WebSphere eXtreme Scale to communicate over a TCP stack. Use the orb.properties file to pass the

properties that are used by the ORB to modify the transport behavior of the data grid. No action is required to use the ORB provided by WebSphere eXtreme Scale or WebSphere Application Server for your WebSphere eXtreme Scale servers.

Configuring Object Request Brokers

The Object Request Broker (ORB) is used by WebSphere® eXtreme Scale to communicate over a TCP stack. Use the orb.properties file to pass the properties that are used by the ORB to modify the transport behavior of the data grid. No action is required to use the ORB provided by WebSphere eXtreme Scale or WebSphere Application Server for your WebSphere eXtreme Scale servers.

- Configuring the Object Request Broker in a WebSphere Application Server environment
  You can use WebSphere eXtreme Scale with applications that use the Object Request Broker (ORB) directly in WebSphere Application Server or WebSphere Application Server Network Deployment environments.
- Configuring the Object Request Broker with stand-alone WebSphere eXtreme Scale processes
  You can use WebSphere eXtreme Scale with applications that use the Object Request Broker (ORB) directly in environments that do not contain WebSphere Application Server or WebSphere Application Server Network Deployment.
- Configuring a custom Object Request Broker
  WebSphere eXtreme Scale uses the Object Request Broker (ORB) to enable communication among processes. No action is required to use the Object Request Broker (ORB) provided by WebSphere eXtreme Scale or WebSphere Application Server for your WebSphere eXtreme Scale servers. Little effort is required to use the same ORBs for your WebSphere eXtreme Scale clients. If instead you must use a custom ORB, the ORB supplied with the IBM® SDK is a good choice, although you must configure the ORB. ORBs from other vendors can be used, also with configuration.

Related reference:
ORB properties

Configuring the Object Request Broker in a WebSphere Application Server environment

You can use WebSphere® eXtreme Scale with applications that use the Object Request Broker (ORB) directly in WebSphere Application Server or WebSphere Application Server Network Deployment environments.

Procedure

1. Name your application servers appropriately.
   You cannot have servers in a WebSphere Application Server environment with the same name when the servers are using the ORB to communicate with each other. You can resolve this restriction by specifying the system property -Dcom.ibm.websphere.orb.uniqueServerName=true for the processes that have the same name. For example, when servers with the name `server1` on each node are used as a catalog service domain, or where multiple node agents are used to form a catalog service domain.

2. Tune the ORB properties within the WebSphere Application Server configuration.
   See ORB properties for more information about the properties that you can tune. Depending on the property, you might change a setting in the administrative console or in the *was_root*properties/orb.properties file.

3. If you are using multiple network interface cards, you must set the ORB_LISTENER_ADDRESS value in the ports panel in the WebSphere Application Server administrative console. Repeat this step for each application server in the configuration.
   a. For an application server, click Servers > Application servers > *server_name*. Under Communications, click Ports. The Ports panel is displayed for the specified server.
   b. Click Details and edit the ORB_LISTENER_ADDRESS value.
   c. Enter the IP address in the Host field. This value must be a private address for a multiple network interface environment.
   Note: DNS host names are not supported for the ORB_LISTENER_ADDRESS value.
   d. Enter the port number in the Port field. The port number specifies the port for which the service is configured to accept client requests. The port value is used with the host name.

What to do next

You can use the **wxsLogAnalyzer** tool to verify the ORB settings across your environment. See Analyzing log and trace data for more information.

Configuring the Object Request Broker with stand-alone WebSphere eXtreme Scale processes

You can use WebSphere® eXtreme Scale with applications that use the Object Request Broker (ORB) directly in environments that do not contain WebSphere Application Server or WebSphere Application Server Network Deployment.

Before you begin

If you use the ORB within the same process as eXtreme Scale when you are running applications, or other components and frameworks, that are not included with eXtreme Scale, you might need to complete additional tasks to ensure that eXtreme Scale runs correctly in your environment.

About this task

Add the ObjectGridInitializer property to the orb.properties file to initialize the use of the ORB in your environment. Use the ORB to enable communication between eXtreme Scale processes and other processes that are in your environment.

Procedure

1. The stand-alone installation does not include an orb.properties file. You must put an orb.properties file is in the java/jre/lib directory. For descriptions of the properties and settings, see ORB properties.
2. In the orb.properties file, type the following line, and save your changes:

```
org.omg.PortableInterceptor.ORBInitializerClass.com.ibm.ws.objectgrid.corba.ObjectGridInitializer
```

Results

eXtreme Scale correctly initializes the ORB and coexists with other applications for which the ORB is enabled.

To use a custom version of the ORB with eXtreme Scale, see Configuring a custom Object Request Broker.

What to do next

You can use the **xsLogAnalyzer** tool to verify the ORB settings across your environment. See Analyzing log and trace data for more information.

Related reference:
ORB properties

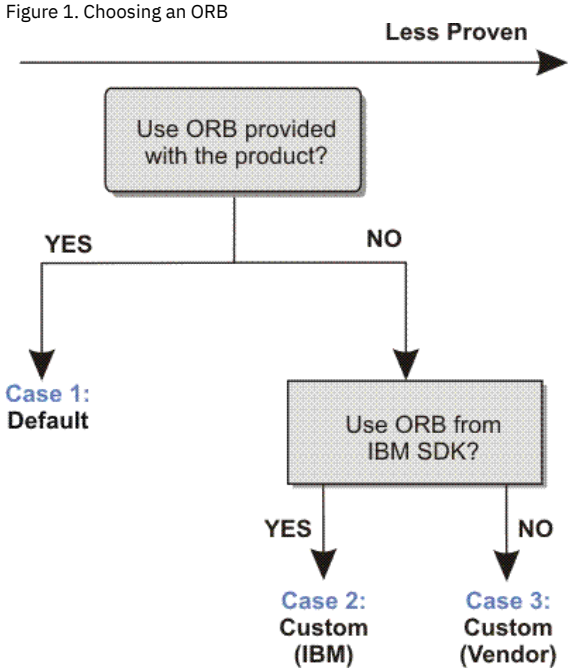
Configuring a custom Object Request Broker

WebSphere® eXtreme Scale uses the Object Request Broker (ORB) to enable communication among processes. No action is required to use the Object Request Broker (ORB) provided by WebSphere eXtreme Scale or WebSphere Application Server for your WebSphere eXtreme Scale servers. Little effort is required to use the same ORBs for your WebSphere eXtreme Scale clients. If instead you must use a custom ORB, the ORB supplied with the IBM® SDK is a good choice, although you must configure the ORB. ORBs from other vendors can be used, also with configuration.

Before you begin

Determine if you are using the ORB provided with WebSphere eXtreme Scale or WebSphere Application Server, the ORB provided with the IBM SDK, or an external vendor ORB.

Figure 1. Choosing an ORB



You can make separate decisions for the WebSphere eXtreme Scale server processes and WebSphere eXtreme Scale client processes. While eXtreme Scale supports developer kits from most vendors, it is recommended you use the ORB that is supplied with eXtreme Scale for both your server and client processes. eXtreme Scale does not support the ORB that is supplied with the Oracle Java™ Development Kit (JDK).

About this task

Become familiar with the configuration that is required to use the ORB of your choice.

Case 1: Default ORB

- For your WebSphere eXtreme Scale server processes, no configuration is required to use the ORB provided with WebSphere eXtreme Scale or WebSphere Application Server.
- For your WebSphere eXtreme Scale client processes, minimal classpath configuration is required to use the ORB provided with WebSphere eXtreme Scale or WebSphere Application Server.

Case 2: Custom ORB (IBM)
To configure your WebSphere eXtreme Scale client processes to use the ORB provided with the IBM SDK, see the instructions later in this topic. You can use the IBM ORB whether you are using the IBM SDK or another development kit. You can use IBM SDK Version 5 or later.

Case 3: Custom ORB (supplied by an external vendor)
Using a vendor ORB for your WebSphere eXtreme Scale client processes is the least tested option. Any problems that you encounter when you use ORBs from independent software vendors must be reproducible with the IBM ORB and compatible JRE before you contact support.
The ORB supplied with the Oracle Java Development Kit (JDK) is not supported.

Procedure

- Configure your client processes to use one of the default ORBs (**Case 1**). Use the following JVM argument :

  ```
  -jvmArgs -Djava.endorsed.dirs=default_ORB_directory${pathSeparator}JRE_HOME/lib/endorsed
  ```

  The default ORB directory is: *wxs_home*/lib/endorsed. Updating the following properties in the orb.properties file might also be necessary:

  ```
  org.omg.CORBA.ORBClass=com.ibm.CORBA.iiop.ORB
  org.omg.CORBA.ORBSingletonClass=com.ibm.rmi.corba.ORBSingleton
  ```

- Configure client or server processes to use IBM SDK Version 5 **(Case 2)**.
  1. Copy the ORB Java archive (JAR) files into an empty directory, or the *custom_ORB_directory*.
     - ibmorb.jar
     - ibmorbapi.jar
  2. Specify the *custom_ORB_directory* directory as an endorsed directory in the scripts that start the Java command.
     Tip: If your Java commands already specify an endorsed directory, another option is to place the *custom_ORB_directory* directory under the existing endorsed directory. By placing the *custom_ORB_directory* directory under the existing endorsed directory, updating the scripts is not necessary. If you decide to update the scripts anyway, be sure to add the *custom_ORB_directory* directory as a prefix to your existing `-Djava.endorsed.dirs=` argument, rather than completely replacing the existing argument.
     - Update scripts for a stand-alone eXtreme Scale environment.
       Edit the path for the *OBJECTGRID_ENDORSED_DIRS* variable in the setupCmdLine.bat|sh file to specify the *custom_ORB_directory*. Save your changes.
     - Update scripts when eXtreme Scale is embedded in a WebSphere Application Server environment.
       Add the following system property and parameters to the startOgServer script:

       ```
       -jvmArgs -Djava.endorsed.dirs=custom_ORB_directory
       ```

     - Update custom scripts that you use to start a client application process or a server process.

       ```
       -Djava.endorsed.dirs=custom_ORB_directory
       ```

Related reference:
ORB properties

Configuring Java clients

You can configure WebSphere® eXtreme Scale to run in a stand-alone environment, or in an environment with WebSphere Application Server. For a WebSphere eXtreme Scale deployment to pick up configuration changes on the server grid side, you must restart processes to make these changes take effect rather than being applied dynamically. However, on the client side, although you cannot alter the configuration settings for an existing client instance, you can create a new client instance with the settings you require by using an XML file or doing so programmatically. When creating a client, you can override the default settings that come from the current server configuration.

You can configure an eXtreme Scale client (Java client only) in the following ways, each of which can be done with a client override XML file or programmatically:

- XML configuration
- Programmatic configuration
- Spring Framework configuration
- Disabling the near cache

- Java client overrides
  You can configure a WebSphere eXtreme Scale client based on your requirements by overriding the server settings. You can override several plug-ins and attributes.
- Configuring Java clients with an XML configuration
  You can use an ObjectGrid configuration XML file to override settings on the client side.

- Configuring the REST gateway with an XML configuration
  You can use a wxsRestGateway.properties file to override data grid settings on the client side.
- Configuring Java clients programmatically
  You can override client-side settings programmatically. Create an ObjectGridConfiguration object that is similar in structure to the server-side ObjectGrid instance.
- Configuring the 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 data grid on each client, serving as a cache for the remote, server-side cache. The near cache is enabled by default when locking is disabled, or is configured as optimistic, and cannot be used when configured as pessimistic.
- Configuring an evictor for the near cache
  To control the size of the near cache, configure a client-side override that enables an evictor on the client.
- Configuring Java Message Service (JMS)-based client synchronization
  You can use JMS-based client synchronization to keep data from the client near cache synchronized with other severs and clients.

Configuring Java clients with an XML configuration

You can use an ObjectGrid configuration XML file to override settings on the client side.

About this task

To change the settings on a WebSphere® eXtreme Scale client, create an ObjectGrid XML file that is similar in structure to the file that was used for the container server.

For a list of the plug-ins and attributes that you can override on the client, see Java client overrides.

Procedure

1. Create an ObjectGrid configuration XML file for the client. This file is similar in structure to the file for the container server.
   Assume that the following XML file was paired with a deployment policy XML file, and these files were used to start a container server.

```
companyGridServerSide.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="CompanyGrid">
            <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsconfigclixml_TransactionCallb
ack"
                className="com.company.MyTxCallback" />
            <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsconfigclixml_ObjectGridEventL
istener"
                className="com.company.MyOgEventListener" />
            <backingMap name="Customer"
                pluginCollectionRef="customerPlugins" />
            <backingMap name="Item" />
            <backingMap name="OrderLine" numberOfBuckets="1049"
                timeToLive="1600" ttlEvictorType="LAST_ACCESS_TIME" />
            <backingMap name="Order" lockStrategy="PESSIMISTIC"
                pluginCollectionRef="orderPlugins" />
        </objectGrid>
    </objectGrids>

    <backingMapPluginCollections>
        <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsconfigclixml_customerPlugins"
>
            <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsconfigclixml_Evictor"
                className="com.ibm.websphere.objectGrid.plugins.builtins.LRUEvictor" />
            <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsconfigclixml_MapEventListener
"
                className="com.company.MyMapEventListener" />
        </backingMapPluginCollection>
        <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsconfigclixml_orderPlugins">
            <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsconfigclixml_MapIndexPlugin"
                className="com.company.MyMapIndexPlugin" />
        </backingMapPluginCollection>
    </backingMapPluginCollections>
</objectGridConfig>
```

   On a container server, the ObjectGrid instance named CompanyGrid behaves as defined by the companyGridServerSide.xml file. By default, the CompanyGrid client has the same settings as the CompanyGrid instance that is running on the server. The following ObjectGrid XML file can be used to

specify some of the attributes and plug-ins on the CompanyGrid client:

companyGridClientSide.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="CompanyGrid">
            <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsconfigclixml_TransactionCallb
ack"
                classname="com.company.MyClientTxCallback" />
            <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsconfigclixml_ObjectGridEventL
istener" classname="" />
                        <backingMap name="Customer" numberOfBuckets="1429"
            pluginCollectionRef="customerPlugins" />
            <backingMap name="Item" />
            <backingMap name="OrderLine" numberOfBuckets="701"
                timeToLive="800" ttlEvictorType="LAST_ACCESS_TIME" />
            <backingMap name="Order" lockStrategy="PESSIMISTIC"
                pluginCollectionRef="orderPlugins" />
        </objectGrid>
    </objectGrids>

    <backingMapPluginCollections>
        <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsconfigclixml_customerPlugins"
>
            <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsconfigclixml_Evictor"
                classname="com.ibm.websphere.objectGrid.plugins.builtins.LRUEvictor" />
            <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsconfigclixml_MapEventListener
" classname="" />
        </backingMapPluginCollection>
        <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsconfigclixml_orderPlugins">
            <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsconfigclixml_MapIndexPlugin"
                classname="com.company.MyMapIndexPlugin" />
        </backingMapPluginCollection>
    </backingMapPluginCollections>
</objectGridConfig>
```

The XML file defines the following overrides:

- The TransactionCallback bean on the client is com.company.MyClientTxCallback instead of the server-side setting of com.company.MyTxCallback.
- The client does not have an ObjectGridEventListener plug-in because the className value is the empty string.
- The client sets the numberOfBuckets to 1429 for the Customer backingMap, retains its Evictor plug-in, and removes the MapEventListener plug-in.
- The numberOfBuckets and timeToLive attributes of the OrderLine backingMap changed.
- Although a different lockStrategy attribute is specified, there is no effect because the lockStrategy attribute is not supported for a client override.

2. Create the client with the XML file.
To create the CompanyGrid client with the companyGridClientSide.xml file, pass the ObjectGrid XML file as a URL to one of the connect methods on the ObjectGridManager interface:

```
ObjectGridManager ogManager =
        ObjectGridManagerFactory.ObjectGridManager();
ClientClusterContext clientClusterContext =
        ogManager.connect("MyServer1.company.com:2809", null, new URL(
                "file:xml/companyGridClientSide.xml"));
```

Related concepts:
Java client overrides
Related reference:
ObjectGrid descriptor XML file
Client properties file

Configuring Java Message Service (JMS)-based client synchronization

You can use JMS-based client synchronization to keep data from the client near cache synchronized with other severs and clients.

Near cache

You can use the built-in Java™ Message Service (JMS)-based com.ibm.websphere.objectgrid.plugins.builtins.JMSObjectGridEventListener class to enable the client invalidation mechanism within a distributed eXtreme Scale environment.

The client invalidation mechanism is the solution for the issue of stale data in client near cache in distributed eXtreme Scale environment. This mechanism ensures that the client near cache is synchronized with servers or other clients. However, even with this JMS-based client invalidation mechanism, the client near cache does not immediately update. A delay occurs when the run time publishes updates.

Two models are available for the client invalidation mechanism in a distributed eXtreme Scale environment:

- Client-server model: In this model, all server processes are in a publisher role that publishes all the transaction changes to the designated JMS destination. All client processes are in receiver roles and receive all transactional changes from the designated JMS destination.
- Client as dual roles model: In this model, all server processes have nothing to do with the JMS destination. All client processes are both JMS publisher and receiver roles. Transactional changes that occur on the client are published to the JMS destination and all the clients receive these transactional changes.

For more information, see JMS event listener.

Client-server model

In a client-server model, the servers are in a JMS publisher role and the client is in JMS receiver role.

```
client-server model XML example
<?xml version="1.0" encoding="UTF-8"?>
<?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="AgentObjectGrid">
      <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxscchval_ObjectGridEventListener"
        className="com.ibm.websphere.objectgrid.plugins.builtins.JMSObjectGridEventListener">
        <property name="invalidationModel" type="java.lang.String" value="CLIENT_SERVER_MODEL" description="" />
        <property name="invalidationStrategy" type="java.lang.String" value="PUSH" description="" />
        <property name="mapsToPublish" type="java.lang.String" value="agent;profile;pessimisticMap" description="" />
        <property name="jms_topicConnectionFactoryJndiName" type="java.lang.String" value="defaultTCF" description="" />
        <property name="jms_topicJndiName" type="java.lang.String" value="defaultTopic" description="" />
        <property name="jms_topicName" type="java.lang.String" value="defaultTopic" description="" />
        <property name="jms_userid" type="java.lang.String" value="" description="" />
        <property name="jms_password" type="java.lang.String" value="" description="" />
        <property name="jndi_properties" type="java.lang.String"
          value="java.naming.factory.initial=org.apache.activemq.jndi.ActiveMQInitialContextFactory;
                                    java.naming.provider.url=
                                tcp://localhost:61616;connectionFactoryNames=defaultTCF;topic.defaultTopic=defaultTopic"
          description="jndi properties" />
      </bean>

      <backingMap name="agent" readOnly="false" pluginCollectionRef="agent" preloadMode="false"
        lockStrategy="OPTIMISTIC" copyMode="COPY_ON_READ_AND_COMMIT" ttlEvictorType="LAST_ACCESS_TIME"
        timeToLive="28800" />
      <backingMap name="profile" readOnly="false" pluginCollectionRef="profile" preloadMode="false"
        lockStrategy="OPTIMISTIC" copyMode="COPY_ON_READ_AND_COMMIT" ttlEvictorType="LAST_ACCESS_TIME"
        timeToLive="2700" />
      <backingMap name="pessimisticMap" readOnly="false" pluginCollectionRef="pessimisticMap" preloadMode="false"
        lockStrategy="PESSIMISTIC" copyMode="COPY_ON_READ_AND_COMMIT" ttlEvictorType="LAST_ACCESS_TIME"
        timeToLive="2700" />
      <backingMap name="excludedMap1" readOnly="false" pluginCollectionRef="excludedMap1" preloadMode="false"
        lockStrategy="OPTIMISTIC" copyMode="COPY_ON_READ_AND_COMMIT" ttlEvictorType="LAST_ACCESS_TIME"
        timeToLive="2700" />
      <backingMap name="excludedMap2" readOnly="false" pluginCollectionRef="excludedMap2" preloadMode="false"
        lockStrategy="OPTIMISTIC" copyMode="COPY_ON_READ_AND_COMMIT" ttlEvictorType="LAST_ACCESS_TIME"
        timeToLive="2700" />
    </objectGrid>
  </objectGrids>

  <backingMapPluginCollections>
    <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxscchval_agent">
      <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxscchval_ObjectTransformer"
className="com.ibm.ws.objectgrid.test.scenario.AgentObjectTransformer" />
    </backingMapPluginCollection>
    <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxscchval_profile">
      <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxscchval_ObjectTransformer"
className="com.ibm.ws.objectgrid.test.scenario.ProfileObjectTransformer" />
      <bean id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxscchval_Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor">
        <property name="maxSize" type="int" value="2000" description="set max size for LRU evictor" />
        <property name="sleepTime" type="int" value="15" description="evictor thread sleep time" />
        <property name="numberOfLRUQueues" type="int" value="50" description="set number of LRU queues" />
      </bean>
    </backingMapPluginCollection>

    <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxscchval_pessimisticMap" />
    <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxscchval_excludedMap1" />
    <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxscchval_excludedMap2" />
  </backingMapPluginCollections>

</objectGridConfig>
```

Client as dual roles model

In client as dual roles model, each client has both JMS publisher and receiver roles. The client publishes every committed transactional change to a designated JMS destination and receives all the committed transactional changes from other clients. The server has nothing to do with JMS in this model.

```
dual-roles model XML example
<?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="AgentObjectGrid">
      <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxscchval_ObjectGridEventListener"
        className="com.ibm.websphere.objectgrid.plugins.builtins.JMSObjectGridEventListener">
        <property name="invalidationModel" type="java.lang.String" value="CLIENT_AS_DUAL_ROLES_MODEL" description="" />
        <property name="invalidationStrategy" type="java.lang.String" value="PUSH" description="" />
        <property name="mapsToPublish" type="java.lang.String" value="agent;profile;pessimisticMap" description="" />
        <property name="jms_topicConnectionFactoryJndiName" type="java.lang.String" value="defaultTCF" description="" />
        <property name="jms_topicJndiName" type="java.lang.String" value="defaultTopic" description="" />
        <property name="jms_topicName" type="java.lang.String" value="defaultTopic" description="" />
        <property name="jms_userid" type="java.lang.String" value="" description="" />
        <property name="jms_password" type="java.lang.String" value="" description="" />
        <property name="jndi_properties" type="java.lang.String"

value="java.naming.factory.initial=org.apache.activemq.jndi.ActiveMQInitialContextFactory;java.naming.provider.url=

tcp://localhost:61616;connectionFactoryNames=defaultTCF;topic.defaultTopic=defaultTopic"
          description="jndi properties" />
      </bean>

      <backingMap name="agent" readOnly="false" pluginCollectionRef="agent" preloadMode="false"
        lockStrategy="OPTIMISTIC" copyMode="COPY_ON_READ_AND_COMMIT" ttlEvictorType="LAST_ACCESS_TIME"
        timeToLive="28800" />
      <backingMap name="profile" readOnly="false" pluginCollectionRef="profile" preloadMode="false"
        lockStrategy="OPTIMISTIC" copyMode="COPY_ON_READ_AND_COMMIT" ttlEvictorType="LAST_ACCESS_TIME"
        timeToLive="2700" />
      <backingMap name="pessimisticMap" readOnly="false" pluginCollectionRef="pessimisticMap" preloadMode="false"
        lockStrategy="PESSIMISTIC" copyMode="COPY_ON_READ_AND_COMMIT" ttlEvictorType="LAST_ACCESS_TIME"
        timeToLive="2700" />
      <backingMap name="excludedMap1" readOnly="false" pluginCollectionRef="excludedMap1" preloadMode="false"
        lockStrategy="OPTIMISTIC" copyMode="COPY_ON_READ_AND_COMMIT" ttlEvictorType="LAST_ACCESS_TIME"
        timeToLive="2700" />
      <backingMap name="excludedMap2" readOnly="false" pluginCollectionRef="excludedMap2" preloadMode="false"
        lockStrategy="OPTIMISTIC" copyMode="COPY_ON_READ_AND_COMMIT" ttlEvictorType="LAST_ACCESS_TIME"
        timeToLive="2700" />
    </objectGrid>
  </objectGrids>

  <backingMapPluginCollections>
    <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxscchval_agent">
      <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxscchval_ObjectTransformer"
className="com.ibm.ws.objectgrid.test.scenario.AgentObjectTransformer" />
    </backingMapPluginCollection>
    <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxscchval_profile">
      <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxscchval_ObjectTransformer"
className="com.ibm.ws.objectgrid.test.scenario.ProfileObjectTransformer" />
      <bean id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxscchval_Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor">
        <property name="maxSize" type="int" value="2000" description="set max size for LRU evictor" />
        <property name="sleepTime" type="int" value="15" description="evictor thread sleep time" />
        <property name="numberOfLRUQueues" type="int" value="50" description="set number of LRU queues" />
      </bean>
    </backingMapPluginCollection>

    <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxscchval_pessimisticMap" />
    <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxscchval_excludedMap1" />
    <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxscchval_excludedMap2" />
  </backingMapPluginCollections>

</objectGridConfig>
```

Related tasks:
Configuring the near cache
Related reference:
ObjectGrid descriptor XML file

Configuring request and retry timeout values

About this task

You can configure settings for the eXtreme Scale client that control how long the client attempts to create network connections, how long the client attempts to process a data grid request to a partition, and how long it attempts to retry that request to the partition, before it returns an exception to your application.

Factors for tuning request and retry timeout values in IBM eXtremeIO (XIO) and Object Request Broker (ORB)

For some tuning options, where you set the values depends on which transport you are using, either XIO or ORB. These transport-level tuning options have the initial impact on interactions with your client because they govern how long the transport attempts network socket connections and how long an individual remote procedure call (RPC) analogous to a data grid operation is given to complete.

When you tune these values, consider what your environment can tolerate under peak load conditions as well as steady state conditions. If you tune the intervals too far under the default values (30 seconds for request timeout, for example), your operations might fail prematurely. Consider the following factors:

- Network latencies
- Coupling of grid interactions with external resources like databases
- Garbage collection pauses resulting from your combination of heap size, heap usage, and garbage collection tuning policies

ORB settings for tuning request and retry timeout values

The following timeout settings exist for the ORB:

com.ibm.CORBA.ConnectionTimeout
: Specifies the amount of time that the ORB attempts to create a socket connection with the remote location before the attempts time out. The ORB caches these connections, and therefore, this operation is not done on every request.

com.ibm.CORBA.RequestTimeout
: Specifies the amount of time that the ORB waits for an RPC to complete before timing out.

com.ibm.CORBA.FragmentTimeout
: Reference the IBM ORB documentation for precise details. The product provides default settings for this value.

com.ibm.CORBA.LocateRequestTimeout
: Reference the IBM ORB documentation for precise details. The product provides default settings for this value.

com.ibm.CORBA.SocketWriteTimeout
: Specifies how many seconds a socket write waits before giving up.

As you tune the RequestTimeout and ConnectionTimeout settings, adjusting them based on the default recommendations can be appropriate. You can also set these settings with the same value, where you define these settings that are based on how long you want the request timeout to be.

XIO settings for tuning request and retry timeout values

With XIO, the following consolidated settings exist:

- The xioTimeout setting determines how long the XIO transport attempts to establish a network socket connection.
- There is no equivalent to the LocateRequest setting and the FragmentTimeout setting in the ORB.
- The xioRequestTimeout value specifies how many seconds any request waits for a response before giving up. This property influences the amount of time a client takes to fail over if a network outage failure occurs. If you set this property too low, requests might time out inadvertently. Carefully consider the value of this property to prevent inadvertent timeouts.

Common settings for tuning request and retry timeout values

The next level of tuning is the requestRetryTimeout. With each transport type, after it throws a system exception because an RPC did not complete in time, the eXtreme Scale client can use the additional time that is defined by the requestRetryTimeout setting (for example, the request timeout is 10 seconds, and the retry request timeout is 20 seconds) to specify how long it takes to complete the following actions:

- Asynchronously asks the catalog server for the latest routing table in case partitions are located elsewhere because of a failover.
- Takes new routes and retries the request, or stops trying and throws an exception to your application.

The requestRetryTimeout property is set in milliseconds. Set the value greater than zero for the request to be retried on exceptions for which retry is available. Set the value to `0` to fail without retries on exceptions. To use the default behavior, remove the property or set the value to `-1`.

XIO failture detection

The properties, xioRequestTimeout, xioTimeout, and requestRetryTimeout have an impact on the XIO failure detection system, in that the clients will be quicker to tell the catalog that a container might be failing, and therefore, trigger the catalog to attempt communication with the container. Where a failure exists, shard failure recovery is initiated for the container shards. Similarly, catalog calls to containers over XIO are governed by the xioRequestTimeout and xioTimeout properties.

Ways to set request retry timeout

You can configure the request retry timeout value on the client properties file or in a session. The session value overrides the client properties setting. If the value is set to greater than zero, the request is tried until either the timeout condition is met or a permanent failure occurs. A permanent failure might be a `DuplicateKeyException` exception. A value of zero indicates the fail-fast mode setting and the data grid does not attempt to try the transaction again after any type of transaction.

Transaction timeout and request retry timeout

During run time, the transaction timeout value is used with the request retry timeout value, ensuring that the request retry timeout does not exceed the transaction timeout.

Two types of transactions exist: Autocommit transactions, and transactions that use explicit begin and commit methods. The valid exceptions for retry differ between these two types of transactions:

- For transactions that are called within a session, transactions are tried again for ORB CORBA SystemException (TransportException for XIO) and eXtreme Scale client TargetNotAvailable exceptions.
- Autocommit transactions are tried again for CORBA SystemException and eXtreme Scale client availability exceptions. These exceptions include the ReplicationVotedToRollbackTransactionException, TargetNotAvailable, and AvailabilityException exceptions.

Application or other permanent failures return immediately and the client does not try the transaction again. These permanent failures include the DuplicateKeyException and KeyNotFoundException exceptions. Use the fail-fast setting to return all exceptions without trying transactions again after any exceptions.

Exceptions where the client tries the transaction again:

- ReplicationVotedToRollbackTransactionException (only on autocommit)
- TargetNotAvailable
- org.omg.CORBA.SystemException (TransportException is the XIO equivalent of this ORB system exception.)
- AvailabilityException (only on autocommit)
- LockTimeoutException (only on autocommit)
- UnavailableServiceException (only on autocommit)

Permanent exceptions, where the transaction is not tried again:

- DuplicateKeyException
- KeyNotFoundException
- LoaderException
- TransactionAffinityException
- LockDeadlockException
- OptimisticCollisionException

Procedure

- Set the request retry timeout value in a client property file.
  To set the requestRetryTimeout value on a client, add or modify the requestRetryTimeout property in the Client properties file. The client properties is the objectGridClient.properties file by default. The requestRetryTimeout property is set in milliseconds. Set the value greater than zero for the request to be retried on exceptions for which retry is available. Set the value to `0` to fail without retries on exceptions. To use the default behavior, remove the property or set the value to `-1`. An example of the value in the objectGridClient.properties file follows:

  ```
  requestRetryTimeout = 30000
  ```

  The requestRetryTimeout value is specified in milliseconds. In the example, if the value is used on an ObjectGrid instance, the requestRetryTimeout value is 30 seconds.

- Set the request retry timeout value programmatically.
  To set the client properties programmatically, first create a client properties file in an appropriate <location> for your application. In the following example, the client properties file refers to the objectGridClient.properties snippet in the previous section. After you connect to ObjectGridManager instance, set the client properties as described. Then, when you have an ObjectGrid instance, the instance has the client properties that you defined in the file. If you change the client properties file, you must explicitly get a new ObjectGrid instance each time.

  ```
  ObjectGridManager manager = ObjectGridManagerFactory.getObjectGridManager();
  String objectGridName = "testObjectGrid";
  URL clientXML = null;
  ClientClusterContext ccc = manager.connect("localhost:2809", null, clientXML);
  File file = new File("<location>/objectGridClient.properties");
  URL url = file.toURI().toURL();
  ccc.setClientProperties(objectGridName, url);
  ObjectGrid objectGrid = ogManager.getObjectGrid(ccc, objectGridName);
  ```

- Set the override file during a session commit.
  To set the request retry timeout on a session or to override the requestRetryTimeout client property, call the setRequestRetryTimeout(long) method on the Session interface.

  ```
  Session sessionA = objectGrid.getSession();
  sessionA.setRequestRetryTimeout(30000);
  ObjectMap mapA = sessionA.getMap("payroll");
  String key = "key:" + j;
  mapA.insert(key, "valueA");
  ```

  This session now uses a requestRetryTimeout value of 30000 ms or 30 seconds, regardless of the value that is set in the client properties file. For more information about the session interface, see Using Sessions to access data in the grid.

Example

Consider the following example, where the client can handle network latency, garbage collection, general contention on the server as a result of setting short timeout values. The requestRetryTimeout property is `10` seconds, and the xioTimeout property matches the ORB ConnectionTimeout value, which is `5` seconds.

Table 1. Data grid configurations for ORB and eXtremeIO transport types

Grid Type	ORB	XIO

Grid Type	ORB	XIO

A Java™ or .NET client application that accesses an eXtreme Scale API directly	• Modify the orb.properties file for your client application. Set the following values:
	  ◦ com.ibm.CORBA.RequestTimeout=5
	  ◦ com.ibm.CORBA.ConnectTimeout=5
	  ◦ com.ibm.CORBA.FragmentTimeout=5
	  ◦ com.ibm.CORBA.LocateRequestTimeout=5
	  ◦ com.ibm.CORBA.SocketWriteTimeout=5
	Note: With WebSphere Application Server, you control the ORB settings through the deployment manager, and not through an orb.properties file.
	• Modify the objectGridClient.properties file for the client application with requestRetryTimeout=10000.
		Modify the objectGridClient.properties file for your client application with the following values:
		• xioRequestTimeout=5000. This value is in milliseconds and is equivalent to com.ibm.CORBA.RequestTimeout.
		• xioTimeout=5. This value is in seconds and is equivalent to com.ibm.CORBA.ConnectTimeout.
		• requestRetryTimeout=10000. This value is in milliseconds and is also used for the ORB transport.
		• ORB FragmentTimeout and LocateRequestTimeout have no XIO equivalent values.

HTTP session	Same ORB configuration as a Java or .NET client application that accesses an eXtreme Scale API directly	Same XIO configuration as a Java or .NET client application that accesses an eXtreme Scale API directly
Dynamic cache	Same ORB configuration as a Java or .NET client application that accesses an eXtreme Scale API directly. For dynamic cache instances, you can set the following additional property on the cache instance: • `com.ibm.websphere.xs.dynacache.request_retry_timeout_override=10000` You can use the requestRetryTimeout setting with the client properties in the class path instead of this cache instance property, if you want it to be the same for every dynamic cache instance.	Same XIO configuration as a Java or .NET client application that accesses an eXtreme Scale API directly. For dynamic cache instances, you can set the following additional property on the cache instance: • `com.ibm.websphere.xs.dynacache.request_retry_timeout_override=10000` You can use the requestRetryTimeout setting with the client properties in the class path instead of this cache instance property, if you want it to be the same for every dynamic cache instance.

Related concepts:
Using Sessions to access data in the grid

Configuring eXtreme Scale connection factories

8.5+ 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 1. 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.

Property Name	Type	Description
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 for more information.

- Configuring Eclipse environments to use eXtreme Scale connection factories
  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.

Previous topic: Installing an eXtreme Scale resource adapter
Next topic: 8.5+ Configuring Eclipse environments to use eXtreme Scale connection factories
Related tasks:
Configuring catalog servers and catalog service domains
Related reference:
Client properties file
Related information:
Catalog service domain settings
Configuring connection factories for resource adapters within applications
Configuring Java EE Connector connection factories in the administrative console
Configuring new J2C connection factories using wsadmin scripting
J2C connection factories collection
Connection factory JNDI name practices

Configuring Eclipse environments to use eXtreme Scale connection factories

8.5+ 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.

Previous topic: 8.5+ Configuring eXtreme Scale connection factories
Next topic: 8.5+ Configuring applications to connect with eXtreme Scale

Configuring applications to connect with eXtreme Scale

8.5+ 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.

Previous topic: 8.5+ Configuring Eclipse environments to use eXtreme Scale connection factories
Next topic: 8.5+ Securing J2C client connections

Related information:
Unshareable and shareable connections
Resource reference benefits
Creating or changing a resource reference

Configuring cache integration

WebSphere® eXtreme Scale can integrate with other caching-related products. You can also use the WebSphere eXtreme Scale dynamic cache provider to plug WebSphere eXtreme Scale into the dynamic cache component in WebSphere Application Server. Another extension to WebSphere Application Server is the WebSphere eXtreme Scale HTTP session manager, which can help to cache HTTP sessions.

- Configuring HTTP session managers
  The HTTP session manager provides session replication capabilities for an associated application. The session manager works with the web container to create and manage the life cycles of HTTP sessions that are associated with the application.
- Configuring the dynamic cache provider for WebSphere eXtreme Scale
  Installing and configuring the dynamic cache provider for eXtreme Scale depends on what your requirements are and the environment you have set up.
- JPA level 2 (L2) cache plug-in
  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.
- 8.5+ Configuring a 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.

Related tasks:
Troubleshooting loaders
Configuring JPA loaders

Configuring HTTP session managers

The HTTP session manager provides session replication capabilities for an associated application. The session manager works with the web container to create and manage the life cycles of HTTP sessions that are associated with the application.

About this task

- Configuring the HTTP session manager with WebSphere Application Server
  While WebSphere Application Server provides session management function, the performance degrades as the number of requests increases. WebSphere eXtreme Scale comes bundled with a session management implementation that provides session replication, high availability, better scalability, and more robust configuration options.
- Configuring HTTP session manager with WebSphere Portal
  You can persist HTTP sessions from WebSphere Portal into a data grid.
- Configuring the HTTP session manager for various application servers
  WebSphere eXtreme Scale is bundled with a session management implementation that overrides the default session manager for a web container. This implementation provides session replication, high availability, better scalability, and configuration options. You can enable the WebSphere eXtreme Scale session replication manager and generic embedded ObjectGrid container startup for various application servers such as Tomcat.
- XML files for HTTP session manager configuration
  When you start a container server that stores HTTP session data, you can either use the default XML files or you can specify customized XML files. These files create specific ObjectGrid names, number of replicas, and so on.
- Servlet context initialization parameters
  The following list of servlet context initialization parameters can be specified in the splicer properties file as required in the chosen splicing method.
- splicer.properties file
  The splicer.properties file contains all of the configuration options for configuring a servlet-filter-based session manager.

Related concepts:
HTTP session management
Related tasks:
Troubleshooting cache integration
Configuring the HTTP session manager with WebSphere Application Server
Configuring WebSphere Application Server HTTP session persistence to a data grid
Configuring HTTP session manager with WebSphere Portal
Configuring the HTTP session manager for various application servers
Related reference:
XML files for HTTP session manager configuration
Servlet context initialization parameters
splicer.properties file

Configuring the HTTP session manager with WebSphere Application Server

While WebSphere® Application Server provides session management function, the performance degrades as the number of requests increases. WebSphere eXtreme Scale comes bundled with a session management implementation that provides session replication, high availability, better scalability, and more robust configuration options.

Before you begin

- WebSphere eXtreme Scale must be installed on your WebSphere Application Server or WebSphere Application Server Network Deployment cell to use the eXtreme Scale session manager. For more information, see Installing WebSphere eXtreme Scale or WebSphere eXtreme Scale Client with WebSphere Application Server.
- When WebSphere eXtreme Scale for HTTP session replication is used on WebSphere Application Server, the Allow overflow session management setting must be checked for every applicable web application and application server that hosts that web application. For more information, see Session management settings.
- Global security must be enabled in the WebSphere Application Server administrative console, if the catalog servers within your catalog service domain have Secure Sockets Layer (SSL) enabled. It must also be enabled if you want to use SSL for a catalog service domain with SSL supported. You require SSL for a catalog server by setting the transportType attribute to `SSL-Required` in the Server properties file. For more information about configuring global security, see Global security settings.

About this task

The WebSphere eXtreme Scale HTTP session manager supports both embedded and remote servers for caching.

- **Embedded scenario**

  In the embedded scenario, the data grid servers are collocated in the same processes where the servlets run. The session manager can communicate directly with the local ObjectGrid instance, avoiding costly network delays.

  If you are using WebSphere Application Server, place the supplied *wxs_home*/session/samples/objectGrid.xml and *wxs_home*/session/samples/objectGridDeployment.xml files into the META-INF directories of your web archive (WAR) files. eXtreme Scale automatically detects these files when the application starts and automatically starts the eXtreme Scale containers in the same process as the session manager.

  You can modify the objectGridDeployment.xml file. Modifying this file depends on whether you want to use synchronous or asynchronous replication and how many replicas you want configured.

- **Remote servers scenario**

In the remote servers scenario, the container servers that are run are in different processes than the servlets. The session manager communicates with a remote container server. To use a remote, network-attached container server, the session manager must be configured with the host names and port numbers of the catalog service domain. The session manager then uses an eXtreme Scale client connection to communicate with the catalog server and the container servers.

If the container servers are starting in independent, stand-alone processes, start the data grid containers with the objectGridStandAlone.xml and objectGridDeploymentStandAlone.xml files that are supplied in the session manager samples directory.

Procedure

1. Splice your application so that it can use the session manager. To use the session manager, you must add the appropriate filter declarations to the web deployment descriptors for the application. In addition, session manager configuration parameters are passed in to the session manager in the form of servlet context initialization parameters in the deployment descriptors. There are multiple ways in which you can introduce this information into your application:
   - **Auto-splice with WebSphere Application Server**
     You can configure your application to use the HTTP session manager for the data grid when you install your application. You can also edit the application or server configuration to use the WebSphere eXtreme Scale HTTP session manager. For more information, see Configuring WebSphere Application Server HTTP session persistence to a data grid.
   - **Splice the application with the addObjectGridFilter script**
     For more information about running this script, see Splicing a session data grid application with the addObjectGridFilter script.
   - **Auto-splice the application with custom properties**
     You do not need to manually splice your applications when the application is running in WebSphere Application Server or WebSphere Application Server Network Deployment.

     You can use this auto-splice option when your environment meets the following conditions:
     - You are using a deployment manager. The cell, server, and application scope are available scopes and are only available when you are running in a deployment manager. If you require a different scope, manually splice your web applications.
     - The splicer.properties file must be in at the same path on all nodes. The nodes are hosting an application server or applications that are being spliced for session replication. For mixed environments containing Windows and UNIX nodes, this option is not possible, so you must manually splice the application.

     Add the `com.ibm.websphere.xs.sessionFilterProps` custom property to either a cell or a server to set the splicer.properties file location for all of the web applications at that scope. The file exists on the deployment manager. If you want to indicate the splicer.properties file for a specific application with a cell-level custom property, enter the name of the custom property as: `<application_name>,com.ibm.websphere.xs.sessionFilterProps`, where *application_name* indicates the name of the application for which you want to apply the custom property. The value is the location of the splicer.properties file your applications require. An example path for the location of a file follows: /opt/splicer.properties.
   - **Manually splice the application with the Ant build script**
     WebSphere eXtreme Scale ships with a build.xml file that can be used by Apache Ant, which is included in the *was_root*/bin folder of a WebSphere Application Server installation. You can modify the build.xml file to change the session manager configuration properties. The configuration properties are identical to the property names in the splicer.properties file. You modify the build.xml file, start the Ant process by running the following command:
     - UNIX ant.sh, ws_ant.sh
     - Windows ant.bat, ws_ant.bat

     (UNIX) or (Windows).
   - **Manually update the web descriptor**
     Edit the web.xml file that is packaged with the web application to incorporate the filter declaration, its servlet mapping, and servlet context initialization parameters. Do not use this method because it is prone to errors.

   For a list of the parameters that you can use, see Servlet context initialization parameters.
2. Deploy the application. Deploy the application with your normal set of steps for a server or cluster. After you deploy the application, you can start the application.
3. Access the application. You can now access the application, which interacts with the session manager and WebSphere eXtreme Scale.

What to do next

You can change most of the configuration attributes for the session manager when you instrument your application to use the session manager. These attributes include: synchronous or asynchronous replication, in-memory session table size, and so on. Apart from the attributes that can be changed at application instrumentation time, the only other configuration attributes that you can change after the application deployment are the attributes that are related to the WebSphere eXtreme Scale server cluster topology and the way that their clients (session managers) connect to them.
Remote scenario behavior: If the entire data grid that is hosting the application session data is unreachable from the web container client, the client instead uses the base web container in WebSphere Application Server for session management. The data grid might be unreachable in the following scenarios:

- A network problem between the web container and the remote container servers.
- The remote container server processes have been stopped.

The number of session references kept in memory, which is specified by sessionTableSize parameter, is still maintained when the sessions are stored in the base web container. The least recently used sessions are invalidated from the web container session cache when the sessionTableSize value is exceeded. If the remote data grid becomes available, sessions that were invalidated from the web container cache can retrieve data from the remote data grid and load the data into a new session. If the entire remote data grid is not available and the session is invalidated from the session cache, the user session data is lost. Because of this issue, do not shut down the entire production remote data grid when the system is running under load.

- Configuring WebSphere Application Server HTTP session persistence to a data grid
  You can configure your WebSphere Application Server application to persist sessions to a data grid. This data grid can be in an embedded container server

that runs within WebSphere Application Server, or it can be in a remote data grid.

- Splicing a session data grid application with the addObjectGridFilter script
  Use the **addObjectGridFilter** command-line script to splice an application with filter declarations and configuration in the form of servlet context initialization parameters.
- Editing the splicer.properties file
  After you have configured the appliance to store HTTP sessions, you can edit other aspects of the HTTP session configuration with the splicer.properties file.

Related concepts:
Interoperability with other products
Monitoring with vendor tools
Installation topologies
Tuning garbage collection with WebSphere Real Time
HTTP session management
Related tasks:
Troubleshooting cache integration
Configuring HTTP session managers
Configuring WebSphere Application Server HTTP session persistence to a data grid
Configuring HTTP session manager with WebSphere Portal
Configuring the HTTP session manager for various application servers
Related reference:
XML files for HTTP session manager configuration
Servlet context initialization parameters
splicer.properties file
Related information:
Configure WebSphere Commerce to use WebSphere eXtreme Scale for dynamic cache to improve performance and scale
WebSphere Business Process Management and Connectivity integration

Configuring WebSphere Application Server HTTP session persistence to a data grid

You can configure your WebSphere® Application Server application to persist sessions to a data grid. This data grid can be in an embedded container server that runs within WebSphere Application Server, or it can be in a remote data grid.

Before you begin

Before you change the configuration in WebSphere Application Server, you must have:

- The name of the session data grid that you want to use. See Configuring the HTTP session manager with WebSphere Application Server for information about creating a session data grid.
- If the catalog service to manage your sessions is outside of the cell in which you are installing your session application, you must create a catalog service domain. For more information, see Creating catalog service domains in WebSphere Application Server.
- If you are configuring a catalog service domain, you might must enable client security on the catalog service domain if the container servers require authentication. These settings inform the run time which CredentialGenerator implementation to use. This implementation generates a credential to pass to the remote data grid. For more information, see Configuring client security on a catalog service domain.
- Enable global security in the WebSphere Application Server administrative console, if you support one of these scenarios:
  - The catalog servers within your catalog service domain have Secure Sockets Layer (SSL) enabled.
  - You want to use SSL for a catalog service domain with SSL supported.

  You require SSL for a catalog server by setting the transportType attribute to `SSL-Required` in the Server properties file. For more information about configuring global security, see Global security settings.
- If you are using Version 7.1.0.3 or later, you can persist sessions that use URL rewriting or cookies as a session tracking mechanism to the data grid. For releases before Version 7.1.0.3, you cannot persist sessions that use URL rewriting as a session tracking mechanism. To enable the persistence of sessions that use URL rewriting, set the useURLEncoding property to `true` in the splicer.properties file after you automatically splice the application.
- When you are automatically splicing applications for HTTP session management in WebSphere Application Server , all of the application servers that host the web application have the HttpSessionIdReuse web container custom property that is set to `true`. This property enables sessions that fail over from one application server to another, or are invalidated from the in-memory session cache in a remote scenario to preserve its session ID across requests. If you do not want this behavior, set the web container custom property to `false` on all of the applicable application servers before you configure session management for the applications. For more information about this custom property, see Troubleshooting cache integration.

Procedure

- **To configure session management when you are installing the application, complete the following steps:**
  1. In the WebSphere Application Server administrative console, click Applications > New application > New Enterprise Application. Choose the Detailed path for creating the application and complete the initial wizard steps.
  2. In the eXtreme Scale session management settings step of the wizard, configure the data grid that you want to use. Choose either the Remote eXtreme Scale data grid or the Embedded eXtreme Scale data grid.
     - For the Remote eXtreme Scale data grid option, choose the catalog service domain that manages the session data grid, and choose a data grid from the list of active session data grids.
     - For the Embedded eXtreme Scale data grid option, choose either the default ObjectGrid configuration or specify the specific location of your ObjectGrid configuration files.
  3. Complete the wizard steps to finish installing your application.

You can also install the application with a wsadmin script. In the following example, the -SessionManagement parameter creates the same configuration that you can in the administrative console:
For the remote eXtreme Scale data grid configuration:

```
AdminApp.install('C:/A.ear', '[  -nopreCompileJSPs -distributeApp
-nouseMetaDataFromBinary -nodeployejb -appname A -edition 8.0
-createMBeansForResources -noreloadEnabled -nodeployws -validateinstall
off -noprocessEmbeddedConfig -filepermission .*\.dll=755#.*\.so=755#.*\.a=755#.*\.sl=755
-buildVersion Unknown -noallowDispatchRemoteInclude -noallowServiceRemoteInclude
-asyncRequestDispatchType DISABLED -nouseAutoLink -SessionManagement [[true
XSRemoteSessionManagement cs0:!:grid0]]
-MapWebModToVH [[MicroWebApp microwebapp.war,WEB-INF/web.xml default_host] [MicroSipApp
microsipapp.war,WEB-INF/web.xml default_host] [MicroDG1App microdg1app.war,WEB-INF/web.xml
default_host] [MicroDG2App microdg2app.war,WEB-INF/web.xml default_host] [MicroSip2App
microsip2app.war,WEB-INF/web.xml default_host]]]')
```

For the eXtreme Scale embedded scenario with default configuration:

```
AdminApp.install('C:/A.ear', '[  -nopreCompileJSPs -distributeApp
-nouseMetaDataFromBinary -nodeployejb -appname A -edition 8.0
-createMBeansForResources -noreloadEnabled -nodeployws -validateinstall
off -noprocessEmbeddedConfig -filepermission .*\.dll=755#.*\.so=755#.*\.a=755#.*\.sl=755
-buildVersion Unknown -noallowDispatchRemoteInclude -noallowServiceRemoteInclude
-asyncRequestDispatchType DISABLED -nouseAutoLink -SessionManagement [[true
XSRemoteSessionManagement  :!: :!:default]]  -MapWebModToVH [[MicroWebApp microwebapp.war,
WEB-INF/web.xml default_host] [MicroSipApp
microsipapp.war,WEB-INF/web.xml default_host] [MicroDG1App microdg1app.war,WEB-INF/web.xml
default_host] [MicroDG2App microdg2app.war,WEB-INF/web.xml default_host] [MicroSip2App
microsip2app.war,WEB-INF/web.xml default_host]]]')
```

For the eXtreme Scale embedded scenario with a custom configuration:

```
AdminApp.install('C:/A.ear', '[  -nopreCompileJSPs -distributeApp
-nouseMetaDataFromBinary -nodeployejb -appname A -edition 8.0
-createMBeansForResources -noreloadEnabled -nodeployws -validateinstall
off -noprocessEmbeddedConfig -filepermission .*\.dll=755#.*\.so=755#.*\.a=755#.*\.sl=755
-buildVersion Unknown -noallowDispatchRemoteInclude -noallowServiceRemoteInclude
-asyncRequestDispatchType DISABLED -nouseAutoLink -SessionManagement [[true
XSRemoteSessionManagement  :!: :!:custom:!:c:\XS\objectgrid.xml:!:c:\XS\objectgriddeployment.xml]]
-MapWebModToVH [[MicroWebApp microwebapp.war,WEB-INF/web.xml default_host] [MicroSipApp
microsipapp.war,WEB-INF/web.xml default_host] [MicroDG1App microdg1app.war,WEB-INF/web.xml
default_host] [MicroDG2App microdg2app.war,WEB-INF/web.xml default_host] [MicroSip2App
microsip2app.war,WEB-INF/web.xml default_host]]]')
```

- **To configure session management on an existing application in the WebSphere Application Server administrative console:**
  Note: The Override session management box is checked when the application is set to use WebSphere eXtreme Scale. This means any server level session settings that were made to WebSphere Application Server configuration are overwritten by the application-level session settings. If you do not want to override settings, you can enable WebSphere eXtreme Scale at the server level.
  1. In the WebSphere Application Server administrative console, click Applications > Application Types > WebSphere enterprise applications > *application_name* > Web Module properties > Session management > eXtreme Scale session management settings.
  2. Update the fields to enable session persistence to a data grid.

  You can also update the application with a wsadmin script. In the following example, the -SessionManagement parameter creates the same configuration that you can in the administrative console:

  For the remote eXtreme Scale data grid configuration:

  ```
  AdminApp.edit('DefaultApplication','[-SessionManagement[[true
  XSRemoteSessionManagement cs0:!:grid0]]]')
  ```

  The `:!:` characters that are passed are used as delimiters. The values that are passed are:

  catalogServiceName*:!:*gridName

  For the eXtreme Scale embedded scenario with default configuration:

  ```
  AdminApp.edit('DefaultApplication','[-SessionManagement[[true
  XSEmbeddedSessionManagement :!:!:!:default]]]')
  ```

  The `:!:` characters that are passed are used as delimiters. The values that are passed are:

  ***catalogServiceName*:!:*gridName*:!:default:!:**
  absolutePath_to_objectGridXmlfile*:!:*absolutePath_to_DeploymentXmlfile

  For the eXtreme Scale embedded scenario with a custom configuration:

  ```
  AdminApp.edit('DefaultApplication','[-SessionManagement[[true
  XSEmbeddedSessionManagement
  :!:!:!:custom:!:c:\XS\objectgrid.xml:!:c:\XS\objectgriddeployment.xml]]]')
  ```

  The `:!:` characters that are passed are used as delimiters. The values that are passed are:

  ***catalogServiceName*:!:*gridName*:!:custom:!:**
  absolutePath_to_objectGridXmlfile*:!:*absolutePath_to_DeploymentXmlfile

  When you save the changes, the application uses the configured data grid for session persistence on the appliance.
- **To configure session management on an existing server:**
  1. In the WebSphere Application Server administrative console, click Servers > Server Types > WebSphere application servers > *server_name* > Session management > eXtreme Scale session management settings.

2. Update the fields to enable session persistence.

You can also configure session management on an existing server with the following wsadmin tool commands:

For the remote eXtreme Scale data grid configuration:

```
AdminTask.configureServerSessionManagement('[-nodeName IBM-C77EE220EB6Node01 -serverName server1
-enableSessionManagement true -sessionManagementType XSRemoteSessionManagement -XSRemoteSessionManagement
[-catalogService cs0 -csGridName grid0]]')
```

For the eXtreme Scale embedded configuration:

- The default configuration, if you are using the default XML files:

```
AdminTask.configureServerSessionManagement('[-nodeName IBM-C77EE220EB6Node01 -serverName server1
-enableSessionManagement true -sessionManagementType XSEmbeddedSessionManagement
-XSEmbeddedSessionManagement [-embeddedGridType default -objectGridXML -objectGridDeploymentXML ]]')
```

- The custom configuration, if you are using customized XML files:

```
AdminTask.configureServerSessionManagement('[-nodeName IBM-C77EE220EB6Node01 -serverName server1
-enableSessionManagement true -sessionManagementType XSEmbeddedSessionManagement
-XSEmbeddedSessionManagement
[-embeddedGridType custom -objectGridXML c:\XS\objectgrid.xml -objectGridDeploymentXML
c:\XS\objectgriddeployment.xml]]')
```

When you save the changes, the server now uses the configured data grid for session persistence with any applications that are running on the server.

Results

You configured HTTP session manager to persist the sessions to a data grid. Entries are removed from the data grid when the sessions time out. See Session management settings for more information about updating the session timeout value in the WebSphere Application Server administrative console.

- eXtreme Scale session management settings
  You can configure your WebSphere Application Server applications to use WebSphere eXtreme Scale or a WebSphere DataPower® XC10 Appliance for session persistence.

Related concepts:
HTTP session management
Related tasks:
Troubleshooting cache integration
Configuring HTTP session managers
Configuring the HTTP session manager with WebSphere Application Server
Configuring HTTP session manager with WebSphere Portal
Configuring the HTTP session manager for various application servers
Related reference:
XML files for HTTP session manager configuration
Servlet context initialization parameters
splicer.properties file

eXtreme Scale session management settings

You can configure your WebSphere® Application Server applications to use WebSphere eXtreme Scale or a WebSphere DataPower® XC10 Appliance for session persistence.

You can edit these settings in the enterprise application installation wizard, or on the application or server detail pages:

- Version 6.1: Applications > Install new application
- Version 6.1: Applications > Enterprise Applications > *application_name*
- Version 6.1: Servers > Application servers > *server_name* > Web container settings > Session management
- Version 7.0: Applications > New application > New Enterprise Application, and choose the Detailed path for creating the application.
- Version 7.0: Applications > Application Types > WebSphere enterprise applications > *application_name* > Web Module properties > Session management > Session management settings
- Version 7.0: Servers > Server Types > WebSphere application servers > *server_name* > Container settings > Session management settings

Enable session management

Enables session management to use WebSphere eXtreme Scale embedded or remote data grid or a WebSphere DataPower XC10 Appliance for session persistence.

Manage session persistence by

Specifies how session persistence is managed. You can choose one of the following options:

- WebSphere DataPower XC10 Appliance
- Remote eXtreme Scale data grid
- Embedded eXtreme Scale data grid

The remaining settings that you configure depend on the session persistence mechanism you choose.

WebSphere DataPower XC10 Appliance specific settings

The following settings are specific to configuring the WebSphere DataPower XC10 Appliance for session persistence.

IP or host name of the WebSphere DataPower XC10 Appliance

Specifies the IP or host name of the appliance to use for persisting sessions.

IBM® WebSphere DataPower XC10 Appliance administrative credentials

Specifies the User name and Password that you use to log in to the DataPower XC10 Appliance user interface. Click Test Connection... to test the connection to your appliance.

Session persistence preference

Specifies the data grid on which sessions are persisted. You can choose one of the following options:

- Persist sessions in a new data grid on the IBM WebSphere DataPower XC10 Appliance. You can then specify a Data grid name.
- Persist sessions in an existing data grid on the IBM WebSphere DataPower XC10 Appliance. You can then enter or browse for an Existing data grid name.

Remote eXtreme Scale data grid configuration

The following settings are specific to configuring the remote eXtreme Scale grid for session persistence.

Catalog service domain that manages the remote session data grid

Specifies the catalog service domain that you want to use to manage your sessions.

If no catalog service domains are displayed, or you want to create a new catalog service domain, click System administration > WebSphere eXtreme Scale > Catalog service domains.

Remote data grid in which to store session information

Specifies the name of the data grid in the catalog service domain in which you want to store your session information. The list of active remote grids is populated when you select a catalog service. The remote data grid must already exist in the eXtreme Scale configuration.

Embedded eXtreme Scale data grid configuration

The following settings are specific to configuring an embedded eXtreme Scale configuration. In the embedded eXtreme Scale scenario, the eXtreme Scale processes are hosted by WebSphere Application Server processes.

eXtreme Scale embedded data grid configuration

- Use default ObjectGrid configuration
- Specify custom ObjectGrid configuration files

Full path to objectgrid.xml file to copy into configuration
: Specifies the full path to the objectgrid.xml file for the configuration that you want to use.

Full path to objectgriddeployment.xml file to copy into configuration
: Specifies the full path to the objectgriddeployment.xml file for the configuration that you want to use.

Configuring HTTP session manager with WebSphere Portal

You can persist HTTP sessions from WebSphere® Portal into a data grid.

Before you begin

Your WebSphere eXtreme Scale and WebSphere Portal environment must meet the following requirements:

- How you install WebSphere eXtreme Scale depends on your deployment scenario. You can run the container servers, which host the data grids, either inside or outside of the WebSphere Application Server cell:
  - If you are running container servers in the WebSphere Application Server cell **(embedded scenario)**: Install both the WebSphere eXtreme Scale client and server on your WebSphere Application Server and WebSphere Portal nodes.
  - If you are running container servers outside of the WebSphere Application Server cell **(remote scenario)**: Install WebSphere eXtreme Scale Client on your WebSphere Application Server and WebSphere Portal nodes.

  See Installing WebSphere eXtreme Scale or WebSphere eXtreme Scale Client with WebSphere Application Server for more information.
- WebSphere Portal Version 7 or later.
- Custom portlets must be configured within WebSphere Portal. The administrative portlets that come with WebSphere Portal cannot currently be integrated with data grids.

About this task

Introducing WebSphere eXtreme Scale into a WebSphere Portal environment can be beneficial in the following scenarios:
Important: Although the following scenarios introduce benefits, increased processor usage in the WebSphere Portal tier can result from introducing WebSphere eXtreme Scale into the environment.

- **When session persistence is required.**
  For example, if the session data from your custom portlets must stay available during a WebSphere Portal Server failure, you can persist the HTTP sessions to the WebSphere eXtreme Scale data grid. Data replicates among many servers, increasing data availability.
- **In a multiple data center topology.**
  If your topology spans multiple data centers across different physical locations, you can persist the WebSphere Portal HTTP sessions to the WebSphere eXtreme Scale data grid. The sessions replicate across data grids in the data centers. If a data center fails, the sessions are rolled over to another data center that has a copy of the data grid data.
- **To lower memory requirements on the WebSphere Portal Server tier.**
  By offloading session data to a remote tier of container servers, a subset of sessions are on the WebSphere Portal servers. This offload of data reduces the memory requirements on the WebSphere Portal Server tier.

Procedure

1. Splice the wps WebSphere Portal application and any custom portlets to enable the sessions to be stored in the data grid.
   See Configuring WebSphere Application Server HTTP session persistence to a data grid for more information. This action results in the splicing of the custom portlets to enable session persistance to your data grid.

   You can splice the application by configuring HTTP session management when you deploy the application, or you can use custom properties to automatically splice your applications. See Configuring the HTTP session manager with WebSphere Application Server for more information about splicing the application.

2. If you are using the remote scenario, where your container servers are outside of the WebSphere Application Server, explicitly start remote eXtreme Scale containers for remote HTTP session persistence scenarios. Start the containers with the XS/ObjectGrid/session/samples/objectGridStandAlone.xml and objectGridDeploymentStandAlone.xml configuration files. For example, you might use the following command: UNIX Linux

   ```
   startOgServer.sh xsContainer1 -catalogServiceEndPoints <host>:<port>
   -objectgridFile XS/ObjectGrid/session/samples/objectGridStandAlone.xml -deploymentPolicyFile
   XS/ObjectGrid/session/samples/objectGridDeploymentStandAlone.xml
   ```

   For more information about starting container servers, see Starting container servers . If you are using an embedded scenario, see Configuring container servers in WebSphere Application Server for more information about configuring and starting container servers.
3. Some versions of WebSphere Portal server can have runtime errors when cookies are added to an HTTP response. Since adds cookies for failover and other purposes, these cookies need to be added to WebSphere Portal server cookie ignore list. For more information, see the cookie.ignore.regex parameter section of Caching pages shared by multiple users on the IBM WebSphere Portal wiki. The two cookies that need to be added to the list are IBMID.* and IBMSessionHandle.*. The updated list may look like this for example `"digest\\.ignore.*|LtpaToken|LtpaToken2|JSESSIONID|IBMID.*|IBMSessionHandle.*"`. For more information, see Caching pages shared by multiple users on the IBM WebSphere Portal wiki.
4. Restart the WebSphere Portal servers. See WebSphere Portal Version 7: Starting and stopping servers, deployment managers, and node agents for more information.

Results

You can access the WebSphere Portal Server, and HTTP session data for the configured custom portlets is persisted to the data grid.
If the entire data grid that is hosting the application session data is unreachable from the web container client, the client instead uses the base web container in WebSphere Application Server for session management. The data grid might be unreachable in the following scenarios:

- A network problem between the Web container and the remote container servers.
- The remote container server processes have been stopped.

The number of session references kept in memory, specified by sessionTableSize parameter, is still maintained when the sessions are stored in the base web container. The least recently used sessions are invalidated from the web container session cache when the sessionTableSize value is exceeded. If the remote data grid becomes available, sessions that were invalidated from the web container cache can retrieve data from the remote data grid and load the data into a new session. If the entire remote data grid is not available and the session is invalidated from the session cache, the user's session data is lost. Because of this issue, you should not shut down the entire production remote data grid when the system is running under load.
Related concepts:
Interoperability with other products
Monitoring with vendor tools
Installation topologies
Tuning garbage collection with WebSphere Real Time
HTTP session management
Related tasks:
Troubleshooting cache integration
Configuring HTTP session managers
Configuring the HTTP session manager with WebSphere Application Server
Configuring WebSphere Application Server HTTP session persistence to a data grid
Configuring the HTTP session manager for various application servers
Related reference:
XML files for HTTP session manager configuration

Servlet context initialization parameters
splicer.properties file
Related information:
Configure WebSphere Commerce to use WebSphere eXtreme Scale for dynamic cache to improve performance and scale
WebSphere Business Process Management and Connectivity integration

Configuring the HTTP session manager for various application servers

WebSphere® eXtreme Scale is bundled with a session management implementation that overrides the default session manager for a web container. This implementation provides session replication, high availability, better scalability, and configuration options. You can enable the WebSphere eXtreme Scale session replication manager and generic embedded ObjectGrid container startup for various application servers such as Tomcat.

About this task

You can use the HTTP session manager with other application servers that are not running WebSphere Application Server, such as WebSphere Application Server Community Edition. To configure other application servers to use the data grid, you must splice your application and incorporate WebSphere eXtreme Scale Java archive (JAR) files into your application.

Procedure

1. Splice your application so that it can use the session manager. To use the session manager, you must add the appropriate filter declarations to the web deployment descriptors for the application. In addition, session manager configuration parameters are passed in to the session manager in the form of servlet context initialization parameters in the deployment descriptors. There are three ways in which you can introduce this information into your application:
   - **addObjectGridFilter** script: For more information, see Splicing a session data grid application with the addObjectGridFilter script.
   - Ant build script:
     WebSphere eXtreme Scale ships with a build.xml file that can be used by Apache Ant, which is included in the *was_root*/bin folder of a WebSphere Application Server installation. You can modify the build.xml file to change the session manager configuration properties. The configuration properties are identical to the property names in the splicer.properties file. After the build.xml file has been modified, invoke the Ant process by running ant.sh, ws_ant.sh (UNIX) or ant.bat, ws_ant.bat (Windows).

   - Update the web descriptor manually:
     Edit the web.xml file that is packaged with the web application to incorporate the filter declaration, its servlet mapping, and servlet context initialization parameters. Do not use this method because it is prone to errors.

   For a list of the parameters that you can use, see Servlet context initialization parameters.
2. Incorporate the WebSphere eXtreme Scale session replication manager JAR files into your application. You can embed the files into the application module WEB-INF/lib directory or in the application server classpath. The required JAR files vary depending on the type of containers that you are using:
   - Remote container servers: ogclient.jar and sessionobjectgrid.jar
   - Embedded container servers: objectgrid.jar and sessionobjectgrid.jar
3. Optional: If you use remote container servers, start the container servers. See Starting a stand-alone catalog service for details.
4. Deploy the application. Deploy the application with your normal set of steps for a server or cluster. After you deploy the application, you can start the application.
5. Access the application. You can now access the application, which interacts with the session manager and WebSphere eXtreme Scale.

What to do next

You can change a majority of the configuration attributes for the session manager when you instrument your application to use the session manager. These attributes include variations to the replication type (synchronous or asynchronous), in-memory session table size, and so on. Apart from the attributes that can be changed at application instrumentation time, the only other configuration attributes that you can change after the application deployment are the attributes that are related to the WebSphere eXtreme Scale server cluster topology and the way that their clients (session managers) connect to them.
Remote scenario behavior: If the entire data grid that is hosting the application session data is unreachable from the web container client, the client instead uses the base web container of the application server for session management. The data grid might be unreachable in the following scenarios:

- A network problem between the Web container and the remote container servers.
- The remote container server processes have been stopped.

The number of session references kept in memory, specified by sessionTableSize parameter, is still maintained when the sessions are stored in the base web container. The least recently used sessions are invalidated from the web container session cache when the sessionTableSize value is exceeded. If the remote data grid becomes available, sessions that were invalidated from the web container cache can retrieve data from the remote data grid and load the data into a new session. If the entire remote data grid is not available and the session is invalidated from the session cache, the user session data is lost. Because of this issue, do not shut down the entire production remote data grid when the system is running under load.
Related concepts:
Interoperability with other products
Monitoring with vendor tools
Installation topologies
Tuning garbage collection with WebSphere Real Time
HTTP session management
Related tasks:
Troubleshooting cache integration
Configuring HTTP session managers
Configuring the HTTP session manager with WebSphere Application Server

Configuring WebSphere Application Server HTTP session persistence to a data grid
Configuring HTTP session manager with WebSphere Portal
Related reference:
XML files for HTTP session manager configuration
Servlet context initialization parameters
splicer.properties file
Related information:
Configure WebSphere Commerce to use WebSphere eXtreme Scale for dynamic cache to improve performance and scale
WebSphere Business Process Management and Connectivity integration

XML files for HTTP session manager configuration

When you start a container server that stores HTTP session data, you can either use the default XML files or you can specify customized XML files. These files create specific ObjectGrid names, number of replicas, and so on.

Sample files location

These XML files are packaged in *wxs_install_root*/ObjectGrid/session/samples for a stand-alone installation or *was_root*/optionalLibraries/ObjectGrid/session/samples for WebSphere® eXtreme Scale installed in a WebSphere Application Server cell.

Embedded XML package

If you are configuring an embedded scenario, the container server starts in the web container tier. Use the objectGrid.xml file and objectGridDeployment.xml file, which are provided by default. You can update these files to customize the behavior of the HTTP session manager.
Figure 1. 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="session" txTimeout="30">
                        <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxssessxml_ObjectGridEventListener"
className="com.ibm.ws.xs.sessionmanager.SessionHandleManager"/>
                        <backingMap name="objectgridSessionMetadata" pluginCollectionRef="objectgridSessionMetadata"
readOnly="false"
                                    lockStrategy="PESSIMISTIC" ttlEvictorType="LAST_ACCESS_TIME" timeToLive="3600"
copyMode="NO_COPY"/>
                        <backingMap name="objectgridSessionAttribute.*" template="true" readOnly="false"
lockStrategy="PESSIMISTIC"
                                    ttlEvictorType="NONE" copyMode="NO_COPY"/>
       <backingMap name="objectgridSessionTTL.*" template="true" readOnly="false" lockStrategy="PESSIMISTIC"
                                    ttlEvictorType="LAST_ACCESS_TIME" timeToLive="3600" copyMode="NO_COPY"/>
                </objectGrid>
        </objectGrids>
    <backingMapPluginCollections>
        <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxssessxml_objectgridSessionMetadata"
>
            <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxssessxml_MapEventListener"
className="com.ibm.ws.xs.sessionmanager.MetadataMapListener"/>
        </backingMapPluginCollection>
    </backingMapPluginCollections>
</objectGridConfig>
```

Values you can change:

ObjectGrid name attribute
The value must match the following values in other configuration files:

- The objectGridName property in the splicer.properties file that is used to splice the web application.
- The objectgridName attribute in the objectGridDeployment.xml file.

If you have multiple applications, and you want the session data to be stored in different data grids, those applications must have different ObjectGrid name attribute values.

ObjectGrid txTimeout attribute
This value determines how many seconds a transaction can be open before the container server triggers the transaction to time out. The default is 30 seconds, and can be changed depending on the environment. If the HTTP session persistence is configured with the replicationInterval servlet context initialization parameter value set greater than zero, transactions are batched on a thread. If the replicationInterval property is set to `0`, a transaction typically starts when a web application retrieves a valid HttpSession object. The transaction commits at the end of the web application request. If your environment has requests that take longer than 30 seconds, set this value accordingly.

Values you cannot change:

ObjectGridEventListener
The ObjectGridEventListener line cannot be changed and is used internally.
objectgridSessionMetadata

The objectgridSessionMetadata line refers to the map where the HTTP session metadata is stored. There is one entry for every HTTP session stored in the data grid in this map.

objectgridSessionTTL.*

This value cannot be changed and is for future use.

objectgridSessionAttribute.*

The `objectgridSessionAttribute.*` text defines a dynamic map. This value is used to create the map in which HTTP session attributes are stored when the fragmentedSession parameter is set to `true` in the splicer.properties file. This dynamic map is called `objectgridSessionAttribute`. Another map is created based on this template called `objectgridSessionAttributeEvicted`, which stores sessions that have timed out, but the web container has not invalidated.

A time to live policy (TTL) is defined for the objectgridSessionMetadata map definition. The other map, objectgridSessionAttribute is dependant on this map and does not require a TTL parameter. For each active HTTP session, an entry gets created in the objectgridSessionMetadata map, and one entry in the objectgridSessionAttribute map for every session attribute. If an in-memory session does not exist due to an application server failure or a session is removed from the in-memory cache on the application server, the grid must initiate the session invalidation using the TTL eviction policy. At the time of eviction, the attributes are removed from the objectgridSessionAttribute map and inserted into a dynamically created map called objectgridSessionAttributeEvicted map. The data is stored in this map until an application server can remove the session and complete session invalidation. Therefore, the TTL parameter is only required in the objectgridSessionMetadata map definition.

Note: The objectgridSessionTTL is not used by WebSphere eXtreme Scale in the current release.

The MapEventListener line is internal and cannot be modified.

Figure 2. objectGridDeployment.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy ../deploymentPolicy.xsd"
        xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">

        <objectgridDeployment objectgridName="session">
                <mapSet name="sessionMapSet" numberOfPartitions="5" minSyncReplicas="0" maxSyncReplicas="0"
                        maxAsyncReplicas="1" developmentMode="false" placementStrategy="PER_CONTAINER">
            <map ref="objectgridSessionMetadata"/>
            <map ref="objectgridSessionAttribute.*"/>
            <map ref="objectgridSessionTTL.*"/>
    </mapSet>
   </objectgridDeployment>
</deploymentPolicy>
```

Values you can change:

ObjectGrid name attribute

The value must match the following values in other configuration files:

- The objectGridName property in the splicer.properties file that is used to splice the web application.
- The ObjectGrid name attribute in the objectGrid.xml file.

If you have multiple applications, and you want the session data to be stored in different data grids, those applications must have different ObjectGrid name attribute values.

mapSet element attributes

You can change all mapSet properties except for the placementStrategy attribute.

Name

Can be updated to any value.

numberOfPartitions

Specifies the number of primary partitions that are started in each server that is hosting the web application. As you add partitions, the data becomes more spread out in the event of a failover. The default value is `5` partitions, and is fine for most applications.

minSyncReplicas, maxSyncReplicas, and maxAsyncReplicas

Specifies the number and type of replicas that store the HTTP session data. The default is `1` asynchronous replica, which is fine for most applications. Synchronous replication occurs during the request path, which can increase the response times for your web application.

developmentMode

Informs the eXtreme Scale placement service whether the replica shards for a partition can be placed on the same node as its primary shard. You can set the value to true in a development environment, but disable this function in a production environment because a node failure could cause the loss of session data.

placementStrategy

Do not change the value of this attribute.

The rest of the file refers to the same map names as in the objectGrid.xml file. These names cannot be changed.

Values you cannot change:

- The placementStrategy attribute on the mapSet element.

Remote XML package

When you are using the remote mode, where the containers run as stand-alone processes, you must use the objectGridStandAlone.xml file and the objectGridDeploymentStandAlone.xml file to start the processes. You can update these files to modify the configuration.

Figure 3. objectGridStandAlone.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="session" txTimeout="30">
```

```
                        <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxssessxml_ObjectGridEventListener"
className="com.ibm.ws.xs.sessionmanager.SessionHandleManager"/>
            <backingMap name="objectgridSessionMetadata" pluginCollectionRef="objectgridSessionMetadata"
                                                         readOnly="false" lockStrategy="PESSIMISTIC"
ttlEvictorType="LAST_ACCESS_TIME" timeToLive="3600"
                                                         copyMode="COPY_TO_BYTES"/>
            <backingMap name="objectgridSessionAttribute.*" template="true" readOnly="false" lockStrategy="PESSIMISTIC"
                                                         ttlEvictorType="NONE" copyMode="COPY_TO_BYTES"/>
            <backingMap name="objectgridSessionTTL.*" template="true" readOnly="false" lockStrategy="PESSIMISTIC"
                                                         ttlEvictorType="LAST_ACCESS_TIME" timeToLive="3600"
copyMode="COPY_TO_BYTES"/>
                </objectGrid>
        </objectGrids>
    <backingMapPluginCollections>
        <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxssessxml_objectgridSessionMetadata"
>
            <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxssessxml_MapEventListener"
className="com.ibm.ws.xs.sessionmanager.MetadataMapListener"/>
        </backingMapPluginCollection>
    </backingMapPluginCollections>
</objectGridConfig>
```

Values you can change:

ObjectGrid name attribute

The value must match the following values in other configuration files:

- The objectGridName property in the splicer.properties file that is used to splice the web application.
- The objectgridName attribute in the objectGridStandAlone.xml file.

If you have multiple applications, and you want the session data to be stored in different data grids, those applications must have different ObjectGrid name attribute values.

ObjectGrid txTimeout attribute

This value determines how many seconds a transaction can be open before the container server triggers the transaction to time out. The default is 30 seconds, and can be changed depending on the environment. If the HTTP session persistence is configured with the replicationInterval servlet context initialization parameter value set greater than zero, transactions are batched on a thread. If the replicationInterval property is set to `0`, a transaction typically starts when a web application retrieves a valid HttpSession object. The transaction commits at the end of the web application request. If your environment has requests that take longer than 30 seconds, set this value accordingly.

Values you cannot change:

ObjectGridEventListener

The ObjectGridEventListener line cannot be changed and is used internally.

objectgridSessionMetadata

The objectgridSessionMetadata line refers to the map where the HTTP session metadata is stored. There is one entry for every HTTP session stored in the data grid in this map.

objectgridSessionTTL.*

This value cannot be changed and is for future use.

objectgridSessionAttribute.*

The `objectgridSessionAttribute.*` text defines a dynamic map. This value is used to create the map in which HTTP session attributes are stored when the fragmentedSession parameter is set to `true` in the splicer.properties file. This dynamic map is called `objectgridSessionAttribute`. Another map is created based on this template called `objectgridSessionAttributeEvicted`, which stores sessions that have timed out, but the web container has not invalidated.

A time to live policy (TTL) is defined for the objectgridSessionMetadata map definition. The other map, objectgridSessionAttribute is dependant on this map and does not require a TTL parameter. For each active HTTP session, an entry gets created in the objectgridSessionMetadata map, and one entry in the objectgridSessionAttribute map for every session attribute. If an in-memory session does not exist due to an application server failure or a session is removed from the in-memory cache on the application server, the grid must initiate the session invalidation using the TTL eviction policy. At the time of eviction, the attributes are removed from the objectgridSessionAttribute map and inserted into a dynamically created map called objectgridSessionAttributeEvicted map. The data is stored in this map until an application server can remove the session and complete session invalidation. Therefore, the TTL parameter is only required in the objectgridSessionMetadata map definition.

Note: The objectgridSessionTTL is not used by WebSphere eXtreme Scale in the current release.

The MetadataMapListener line is internal and cannot be modified.

Figure 4. objectGridDeploymentStandAlone.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy ../deploymentPolicy.xsd"
        xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">

        <objectgridDeployment objectgridName="session">
                <mapSet name="sessionMapSet" numberOfPartitions="47" minSyncReplicas="0" maxSyncReplicas="0"
                        maxAsyncReplicas="1" developmentMode="false" placementStrategy="FIXED_PARTITIONS">
            <map ref="objectgridSessionMetadata"/>
            <map ref="objectgridSessionAttribute.*"/>
            <map ref="objectgridSessionTTL.*"/>
            </mapSet>
    </objectgridDeployment>
</deploymentPolicy>
```

Values you can change:

objectgridName attribute

The value must match the following values in other configuration files:

- The objectGridName property in the splicer.properties file that is used to splice the web application.
- The ObjectGrid name attribute in the objectGrid.xml file.

If you have multiple applications, and you want the session data to be stored in different data grids, those applications must have different ObjectGrid name attribute values.

mapSet element attributes
: You can change all mapSet properties.

Name
: Can be updated to any value.

numberOfPartitions
: The default fixed partition placement strategy is enabled with the FIXED_PARTITIONS setting. This setting specifies the number of total partitions that are spread across all running grid containers. The default value is `47` partitions, and is fine for most applications. If a per container placement strategy is used with the PER_CONTAINER setting, then this specifies the number of primary partitions started in each grid container. As you add partitions, the data becomes more spread out in the event of a failover. The recommended value is `5` for the per container strategy.

minSyncReplicas, maxSyncReplicas, and maxAsyncReplicas
: Specifies the number of primary partitions that are started in each server that is hosting the web application. As you add partitions, the data becomes more spread out in the event of a failover. The default value is `5` partitions, and is fine for most applications.

developmentMode
: Informs the eXtreme Scale placement service whether the replica shards for a partition can be placed on the same node as its primary shard. You can set the value to true in a development environment, but disable this function in a production environment because a node failure could cause the loss of session data.

placementStrategy
: You can change this attribute to one of the following:

- FIXED_PARTITIONS This is the default value and is the preferred approach for using a remote HTTP Session topology. It is required if you are using Multi-Master replication
- PER_CONTAINER This is still a supported configuration in a remote topology.

Related concepts:
HTTP session management
Related tasks:
Configuring HTTP session managers
Configuring the HTTP session manager with WebSphere Application Server
Configuring WebSphere Application Server HTTP session persistence to a data grid
Configuring HTTP session manager with WebSphere Portal
Configuring the HTTP session manager for various application servers
Troubleshooting cache integration

Servlet context initialization parameters

The following list of servlet context initialization parameters can be specified in the splicer properties file as required in the chosen splicing method.

Parameters

applicationQualifiedCookies
: A string value of either `true` or `false`. Set to `true` if your environment contains multiple applications that use unique cookie names. Default is `false`, which assumes all applications are using the same cookie name.

authenticationRetryCount
: Specifies the retry count for authentication if the credential is expired. If the value is set to 0, there will not be any authentication retry.

catalogHostPort
: The catalog server can be contacted to obtain a client side ObjectGrid instance. The value must be of the form `host:port<,host:port>`. The host is the listener host on which the catalog server is running. The port is the listener port for that catalog server process. This list can be arbitrarily long and is used for bootstrapping only. The first viable address is used. It is optional inside WebSphere® Application Server if the catalog.services.cluster property is configured.

credentialAuthentication
: Specifies the client credential authentication support.The possible values are:

- Never- The client does not support credential authentication.
- Supported - The client supports the credential authentication if and only if the server supports too.
- Required - The client requires the credential authentication. The default value is Supported.

cookieDomain
: Specifies if you require sessions to be accessible across hosts. Set the value to the name of the common domain between the hosts.

cookiePath
: Specifies the name of the class that implements the com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator interface. This class is used to get credentials for clients.

credentialGeneratorClass
: The name of the class that implements the com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator interface. This class is used to obtain credentials for clients.

credentialGeneratorProps
: The properties for the CredentialGenerator implementation class. The properties are set to the object with the setProperties(String) method. The credentialGeneratorProps value is used only if the value of the credentialGeneratorClass property is not null.

enableSessionStats
: A string value of either `true` or `false`. Enables eXtreme Scale client HTTP Sessions statistics tracking.

fragmentedSession
: A string value of either `true` or `false`. The default value is `true`. Use this setting to control whether the product stores session data as a whole entry, or stores each attribute separately.

  Set the fragmentedSession parameter to `true` if the web application session has many attributes or attributes with large sizes. Set fragmentedSession to `false` if a session has few attributes, because all the attributes are stored in the same key in the data grid.

  In the previous, filter-based implementation, this property was referred to as persistenceMechanism, with the possible values of ObjectGridStore (fragmented) and ObjectGridAtomicSessionStore (not fragmented).

objectGridType
: A string value of either `REMOTE` or `EMBEDDED`. The default is `REMOTE`.

  If it is set to `REMOTE`, the session data is stored outside of the server on which the web application is running.

  If it is set to `EMBEDDED`, an embedded eXtreme Scale container starts in the application server process on which the web application is running.

objectGridName
: A string value that defines the name of the ObjectGrid instance used for a particular web application. The default name is session.

  This property must reflect the objectGridName in both the ObjectGrid XML and deployment XML files used to start the eXtreme Scale container servers.

objectGridXML
: The file location of the objectgrid.xml file. The built-in XML file packaged in the eXtreme Scale library is loaded automatically if `objectGridType=EMBEDDED` and the objectGridXML property is not specified.

objectGridDeploymentXML
: Specifies the location of the objectGrid deployment policy XML file. The built-in XML file packaged in the eXtreme Scale library is loaded automatically if `objectGridType=EMBEDDED` and the objectGridDeploymentXML property is not specified.

replicationInterval
: An integer value (in seconds) that defines the time between writing of updated sessions to ObjectGrid. The default is `10` seconds. Possible values are from 0 to 60. 0 means that updated sessions are written to the ObjectGrid at the end of servlet service method call for each request. A higher replicationInterval value improves performance because fewer updates are written to the data grid. However, a higher value makes the configuration less fault tolerant.

  This setting applies only when objectGridType is set to `REMOTE`.

reuseSessionID
: A string value of either `true` or `false`. The default is `false`. Set to `true` if the underlying web container reuses session IDs across requests to different hosts. The value of this property must be the same as the value in the web container. If you are using WebSphere Application Server and configuring eXtreme Scale HTTP session persistence using the administrative console or **wsadmin** tool scripting, the web container custom property `HttpSessionIdReuse=true` is added by default. The reuseSessionID is also set to `true`. If you do not want the session IDs to be reused, set the `HttpSessionIdReuse=false` custom property on the web container custom property before you configure eXtreme Scale session persistence.

sessionIdOverrideClass
: The name of the class that implements the com.ibm.websphere.xs.sessionmanager.SessionIDOverride interface. This class is used to override the unique session identifier retrieved with the HttpSession.getId() method so that all applications have the same ID. The default is to use the user ID derived from the HttpSession.getId().

sessionStatsSpec = session.all = enabled
: A string of eXtreme Scale client HTTP statistics specification.

shareSessionsAcrossWebApps
: A string value of either `true` or `false`. The default is `false`. Specifies if sessions are shared across web applications, specified as a string value of either `true` or `false`. The servlet specification states that HTTP Sessions cannot be shared across web applications. An extension to the servlet specification is provided to allow this sharing.

sessionTableSize
: An integer value that defines the number of session references kept in memory. The default is `1000`.

  This setting pertains only to a `REMOTE` topology because the `EMBEDDED` topology already has the session data in the same tier as the web container.

  Sessions are evicted from the in-memory table based on least recently used (LRU) logic. When a session is evicted from the in-memory table, it is invalidated from the web container. However, the data is not removed from the grid, so subsequent requests for that session can still retrieve the data. This value must be set higher than the web container maximum thread pool value, which reduces contention on the session cache.

securityEnabled
: A string value of either `true` or `false`. The default value is `false`. This setting enables eXtreme Scale client security. It must match the securityEnabled setting in the eXtreme Scale server properties file. If the settings do not match, an exception occurs.

sessionIdOverrideClass
: Overrides the retrieved session ID of an application. The default is to use the ID derived from the HttpSession.getId() method. Enables eXtreme Scale client HTTP Sessions to override the unique session ID of an application so that all applications are retrieved with the same ID. Set to the implementation of the com.ibm.websphere.xs.sessionmanager.SessionIDOverride interface. This interface determines the HttpSession ID based on the HttpServletRequest object.

traceSpec
: Specifies the IBM® WebSphere trace specification as a string value. Use this setting for application servers other than WebSphere Application Server.

traceFile
: Specifies the trace file location as a string value. Use this setting for application servers other than WebSphere Application Server.

useURLEncoding
: A string value of either `true` or `false`. The default is `false`. Set to `true` if you want to enable URL rewriting. The default value is `false`, which indicates that cookies are used to store session data. The value of this parameter must be the same as the web container settings for session management.

useCookies
: A string value of either `true` or `false`. Set to true if the underlying web container will reuse session ID's across requests to different hosts. The default is `false`. The value of this should be the same as what is set in the web container.

Related concepts:
HTTP session management
Related tasks:
Configuring HTTP session managers
Configuring the HTTP session manager with WebSphere Application Server
Configuring WebSphere Application Server HTTP session persistence to a data grid
Configuring HTTP session manager with WebSphere Portal
Configuring the HTTP session manager for various application servers
Troubleshooting cache integration

splicer.properties file

The splicer.properties file contains all of the configuration options for configuring a servlet-filter-based session manager.

Sample splicer properties

If you choose to use any of the additional properties that are described in this file, be sure to uncomment the lines for the properties that you want to enable.

```
# Properties file that contains all the configuration
# options that the servlet filter based ObjectGrid session
# manager can be configured to use.
#
# This properties file can be made to hold all the default
# values to be assigned to these configuration settings, and
# individual settings can be overridden using ANT Task
# properties, if this properties file is used in conjunction
# with the filtersplicer ANT task.

# A string value of either "REMOTE" or "EMBEDDED".  The default is REMOTE.
# If it is set to "REMOTE", the session data will be stored outside of
# the server on which the web application is running. If it is set to
# "EMBEDDED", an embedded WebSphere eXtreme Scale container will start
# in the application server process on which the web application is running.

objectGridType = REMOTE

# A string value that defines the name of the ObjectGrid
# instance used for a particular web application. The default name
# is session. This property must reflect the objectGridName in both
# the objectgrid xml and deployment xml files used to start the eXtreme
# Scale containers.

objectGridName = session

# Catalog Server can be contacted to obtain a client side
# ObjectGrid instance.  The value needs to be of the
# form "host:port<,host:port>", where the host is the listener host
# on which the catalog server is running, and the port is the listener
# port for that catalog server process.
# This list can be arbitrarily long and is used for bootstrapping only.
# The first viable address will be used.  It is optional inside WebSphere
# if the catalog.services.cluster property is configured.

# catalogHostPort = host:port<,host:port>

# An integer value (in seconds) that defines the time in seconds between
# writing of updated sessions to ObjectGrid. The default is 10. This property
# is only used when objectGridType is set to REMOTE. Possible values are
# from 0 to 60. 0 means updated sessions are written to the ObjectGrid
# at the end of servlet service method call for each request.

replicationInterval = 10

# An integer value that defines the number of session references
# kept in memory. The default is 1000. This property is only used when
# objectGridType is set to REMOTE. When the number of sessions stored
# in memory in the web container exceeds this value, the least recently
# accessed session is invalidated from the web container. If a request
# comes in for that session after it's been invalidated, a new session
```

```
# will be created (with a new session ID if reuseSessionId=false),
# populated with the invalidated session's attributes. This value should
# always be set to be higher than the maximum size of the web container
# thread pool to avoid contention on this session cache.

sessionTableSize = 1000

# A string value of either "true" or "false", default is "true".
# It is to control whether we store session data as a whole entry
# or store each attribute separately.
# This property was referred to as persistenceMechanism in the
# previous filter-based implementation, with the possible values
# of ObjectGridStore (fragmented) and ObjectGridAtomicSessionStore
# (not fragmented).

fragmentedSession = true

# A string value of either "true" or "false", default is "false".
# Enables eXtreme Scale client security. This setting needs to match
# the securityEnabled setting in the eXtreme Scale server properties
# file. If the settings do not match, an exception occurs.

securityEnabled = false

# Specifies the client credential authentication support.
#   The possible values are:
#   Never      - The client does not support credential authentication.
#   Supported* - The client supports the credential authentication if and only if the server
#                supports too.
#   Required   - The client requires the credential authentication.
#   The default value is Supported.

# credentialAuthentication =

# Specifies the retry count for authentication if the credential
# is expired. If the value is set to 0, there will not be
# any authentication retry.

# authenticationRetryCount =

# Specifies the name of the class that implements the
# com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator
# interface. This class is used to get credentials for clients.

# credentialGeneratorClass =

# Specifies the properties for the CredentialGenerator implementation
# class. The properties are set to the object with the setProperties(String)
# method. The credentialGeneratorProps value is used only if the value of the
# credentialGeneratorClass property is not null.

# credentialGeneratorProps =

# The file location of the objectgrid xml file.
# The built-in xml file packaged in the eXtreme Scale library
# will automatically be loaded if this property
# is not specified and if objectGridType=EMBEDDED

# objectGridXML =

# The file location of the objectGrid deployment policy xml file.
# The built-in xml file packaged in the eXtreme Scale library
# will automatically be loaded if this property
# is not specified and if objectGridType=EMBEDDED

# objectGridDeploymentXML =

# A string of IBM WebShere trace specification,
# useful for all other application servers besides WebSphere.

# traceSpec =

# A string of trace file location.
# useful for all other application servers besides WebSphere.

# traceFile=

# This property should be set if you require sessions to be
# accessible across hosts. The value will be the name of the
# common domain between the hosts.

# cookieDomain=

# This property should be set to the same path you have configured
# for your application server cookie settings. The default path
# is /.

# cookiePath

# Set to true if the underlying web container will reuse
# session ID's across requests to different hosts. Default
```

```
# is false. The value of this should be the same as what is
# set in the web container.

# reuseSessionId=

# A string value of either "true" or "false", the default is
# "false". Per the servlet specification, HTTP Sessions cannot
# be shared across web applications. An extension to the servlet
# specification is provided to allow this sharing.

# shareSessionsAcrossWebApps = false

# A string value of either "true" or "false", default is "false".
# Set to true if you want to enable urlRewriting.  Default is
# false. The value of this should reflect what is set in the
# web container settings for session management.

# useURLEncoding = false

# Set to false if you want to disable cookies as the session tracking
# mechanism.  Default is true. The value of this should reflect what
# is set in the web container settings for session management.

# useCookies = true


# A string of eXtreme Scale client Http session statistics specification,

# sessionStatsSpec = session.all=enabled

# Set to true if your environment contains multiple applications that
# use unique cookie names. Default is false, which assumes all applications
# are using the same cookie name.

# applicationQualifiedCookies = false



# The prefix of the two cookies that are added to the HTTP response that
# represent the ID of the session object in the data grid and the session
# handle that contains the session's data. Default is IBM

# cookieNamePrefix = IBM

# When listenerMode = true (default), use the web container to generate sessions.
# if it is set to false, the web container will not be used.
# Setting listenerMode=false is only supported when
# reuseSessionId = true, sessionTableSize > 0, and when installed on WebSphere
# Application Server.
# listenerMode = true


# Only applies when listenerMode=false. When this property is set to true, all
# listeners configured for this web application will get the HttpSessionListener.sessionCreated
# call whenever a session is created, or a session is retrieved from the remote grid.
# Examples of this would be when an application server fails, or the
# sessionTableSize is exceeded and a session has to be brought back into the
# application server from the remote grid. HttpSessionListener.sessionDestroyed will also be
# called when a session is invalidated from the in-memory session cache if the sessionTableSize
# limit is exceeded.

# sessionCreatedOnFailover = false
```

Related concepts:
HTTP session management
Related tasks:
Configuring HTTP session managers
Configuring the HTTP session manager with WebSphere Application Server
Configuring WebSphere Application Server HTTP session persistence to a data grid
Configuring HTTP session manager with WebSphere Portal
Configuring the HTTP session manager for various application servers
Troubleshooting cache integration
8.5

Configuring the dynamic cache provider for WebSphere eXtreme Scale

Installing and configuring the dynamic cache provider for eXtreme Scale depends on what your requirements are and the environment you have set up.

Before you begin

- To use the dynamic cache provider, 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 for more information.

- Global security must be enabled in the WebSphere Application Server administrative console, 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. You require SSL for a catalog server by setting the transportType attribute to `SSL-Required` in the Server properties file. For more information about configuring global security, see Global security settings.

About this task

For information about using the eXtreme Scale dynamic cache provider with IBM® WebSphere Commerce, see the following topics in the IBM WebSphere Commerce documentation:

- Enabling the dynamic cache service and servlet caching
- Enabling WebSphere Commerce data cache

If you are not specifically directing your caching to a defined Object Cache or Servlet Cache instance, then it is likely that the Dynamic Cache API calls are being serviced by the baseCache. If you want to use the eXtreme Scale dynamic cache provider for JSP, Web services or command caching, then you must set the baseCache instance to use the eXtreme Scale dynamic cache provider. The same configuration properties are used to configure the baseCache instance. Remember that these configuration properties need to be set as Java™ Virtual Machine (JVM) custom properties. This caveat applies to any cache configuration property discussed in this section except for servlet caching. To use eXtreme Scale with the dynamic cache provider for servlet caching, be sure to configure enablement in system properties rather than custom properties.

Procedure

1. Enable the eXtreme Scale dynamic cache provider.
   - **WebSphere Application Server Version 7.0 and later:**
     You can configure the dynamic cache service to use the eXtreme Scale dynamic cache provider with the administrative console. After you install eXtreme Scale, the eXtreme Scale dynamic cache provider is immediately available as a Cache Provider option in the administrative console. For more information, see WebSphere Application Server Version 7.0 information center: Selecting a cache service provider.

   - **WebSphere Application Server Version 6.1:**
     Use a custom property to configure the dynamic cache service to use the eXtreme Scale dynamic cache provider. You can also use these custom properties in WebSphere Application Server Version 7.0 and later. To create a custom property on a cache instance, click Resources > Cache instances > cache_instance_type > cache_instance_name > Custom properties > New. If you are using the base cache instance, create the custom properties on the JVM.

     com.ibm.ws.cache.CacheConfig.cacheProviderName
     To use the eXtreme Scale dynamic cache provider, set the value to `com.ibm.ws.objectgrid.dynacache.CacheProviderImpl`. You can create this custom property on a dynamic cache instance, or the base cache instance. If you choose to set the custom property on the base cache instance, then all other cache instances on the server use the eXtreme Scale provider by default. Any eXtreme Scale dynamic cache provider configuration properties set for the baseCache are the default configuration properties for all cache instances backed by eXtreme Scale. To override the base cache instance and make a particular dynamic cache instance use the default dynamic cache provider, create the com.ibm.ws.cache.CacheConfig.cacheProviderName custom property on the dynamic cache instance and set the value to `default`.

2. Optional: If you are using replicated cache instances, configure the replication setting for the cache.
   With the eXtreme Scale dynamic cache provider, you can have local cache instances or replicated cache instances. If you are only using local cache instances, you can skip this step.

   Use one of the following methods to configure the replicated cache:

   - Enable cache replication with the administrative console. You can enable cache replication at any time in WebSphere Application Server Version 7.0. In WebSphere Application Server Version 6.1, you must create a DRS replication domain.
   - Enable cache replication with the com.ibm.ws.cache.CacheConfig.enableCacheReplication custom property to force the cache to report that it is a replicated cache, even though a DRS replication domain has not been assigned to it. Set the value of this custom property to `true`. Set this custom property on the cache instance if you are using an object cache or servlet cache, or on the JVM if you are using the baseCache instance.
3. Optional: If you are using eXtreme Scale as a JSP fragment cache, set the com.ibm.ws.cache.CacheConfig.disableTemplateInvalidation custom property to `true` to disable template-based invalidations during JSP reloads.
4. Configure the topology for the dynamic cache service.
   The only required configuration parameter for the eXtreme Scale dynamic cache provider is the cache topology. Set the custom property on the cache instance or for the dynamic cache service if you are using baseCache instance. Enter the name of the custom property as: `com.ibm.websphere.xs.dynacache.topology`.

   The three possible values for this property follow. You must use one of the allowed values:
   - `embedded`
   - `embedded_partitioned`
   - `remote`

   If you are using embedded or embedded partitioned topologies, consider setting the `com.ibm.ws.cache.CacheConfig.ignoreValueInInvalidationEvent` custom property to `true` to save some serialization costs. Set this custom property on the cache instance or the JVM if you are using the baseCache instance.

5. Optional: If you are using an embedded partitioned topology, configure the number of initial containers for the dynamic cache service.
   You can maximize the performance of caches that are using the embedded partitioned topology by configuring the number of initial containers. Set the variable as a system property on the WebSphere Application Server Java virtual machine.

   Enter the name of the property as: `com.ibm.websphere.xs.dynacache.num_initial_containers`.

   The recommended value of this configuration property is an integer that is equal to or slightly less than the total number of WebSphere Application Server instances that are accessing this distributed cache instance. For example, if a dynamic cache service is shared between data grid members, then the value should be set to the number of data grid members.

For embedded or embedded_partitioned topologies, you must be using Version 7.0 of WebSphere Application Server. Set the following custom property on the JVM process to ensure that the initial containers are available right away.

```
com.ibm.ws.cache.CacheConfig.createCacheAtServerStartup=true
```

6. Configure the eXtreme Scale catalog service grid.
   When you are using eXtreme Scale as the dynamic cache provider for a distributed cache instance, you must configure an eXtreme Scale catalog service domain.

   A single catalog service domain can service multiple dynamic cache service providers backed by eXtreme Scale.

   A catalog service can run inside or outside of WebSphere Application Server processes. Starting with eXtreme Scale Version 7.1, when you use the administrative console to configure catalog service domains, the dynamic cache uses these settings. It is not necessary to take additional steps to set up a catalog service. For more information, see Creating catalog service domains in WebSphere Application Server.

7. Configure custom key objects.
   When you are using custom objects as keys the objects must implement the Serializable or Externalizable interface. When you are using the embedded or embedded partitioned topologies, you must place objects on the WebSphere shared library path, just like if they were being used with the default dynamic cache provider. See Using the DistributedMap and DistributedObjectCache interfaces for the dynamic cache in the WebSphere Application Server Network Deployment information center for more details.

   If you are using the remote topology, you must place the custom key objects on the CLASSPATH for the standalone eXtreme Scale containers. See Starting container servers for more information.

8. Optional: If you are using a remote topology, configure the eXtreme Scale container servers.
   - **Embedded or embedded partitioned topology:**
     The cached data is stored in WebSphere eXtreme Scale container servers. Container servers can run inside or outside of WebSphere Application Server processes. The eXtreme Scale provider automatically creates containers inside the WebSphere process when you are using embedded or embedded partitioned topologies for a cache instance. No further configuration is needed for these topologies.

   - **Remote topology:**
     When you are using the remote topology, you must start up stand-alone eXtreme Scale container servers before the WebSphere Application Server instances that access the cache instance start. To start stand-alone container servers, see Starting and stopping stand-alone servers. Verify that all the container servers for a specific dynamic cache service point to the same catalog service endpoints.

     The XML configuration files for the stand-alone eXtreme Scale dynamic cache provider containers are in either the *was_root*/optionalLibraries/ObjectGrid/dynacache/etc directory for installations on top of WebSphere Application Server, or the *wxs_install_root*/ObjectGrid/dynacache/etc directory for stand-alone installations. The files are named dynacache-remote-objectgrid.xml and dynacache-remote-definition.xml. Make copies of these files to edit and use when you are starting stand-alone containers for the eXtreme Scale dynamic cache provider. The **numInitialContainers** parameter in the **dynacache-remote-deployment.xml** file must match the number of container processes that are running. Note that the numberOfPartitions attribute in the `dynacache-remote-objectgrid.xml` file has a default value of `47`.

     Note: The set of container server processes must have enough free memory to service all the dynamic cache instances that are configured to use the remote topology. Any WebSphere Application Server process that shares the same or equivalent values for the catalog.services.cluster custom property must use the same set of stand-alone containers. The number of containers and number of servers on which they reside must be sized appropriately. See Dynamic cache capacity planning for additional details.
     A command line entry that starts a stand-alone container for the eXtreme Scale dynamic cache provider follows:

     UNIX

     ```
     startOgServer.sh container1 -objectGridFile
     ../dynacache/etc/dynacache-remote-objectgrid.xml -deploymentPolicyFile
     ../dynacache/etc/dynacache-remote-deployment.xml -catalogServiceEndPoints
     MyServer1.company.com:2809
     ```

9. For distributed or embedded topologies, enable the sizing agent to improve memory consumption estimates.
   The sizing agent estimates memory consumption (usedBytes statistic). The agent requires a Java 5 or higher JVM.

   Load the agent by adding the following argument to the JVM command line:

   ```
   -javaagent:WXS lib directory/wxssizeagent.jar
   ```

   For an embedded topology, add the argument to the command line of the WebSphere Application Server process.

   For a distributed topology, add the argument to command line of the eXtreme Scale processes (containers) and the WebSphere Application Server process.

Related concepts:
Dynamic cache capacity planning
Dynamic cache provider overview
Data invalidation

JPA level 2 (L2) cache plug-in

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 both OpenJPA and Hibernate Version 3.0.

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.

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 or Intra-domain topology.
   - If the data cannot fit into a single JVM heap, use the Embedded, partitioned topology, or Remote topology
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: local read, remote write
   - Intra-domain topology: local read, local write
   - Embedded, partitioned topology: Partitioned: remote read, remote write
   - Remote topology: 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 or Intra-domain topology.

   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.

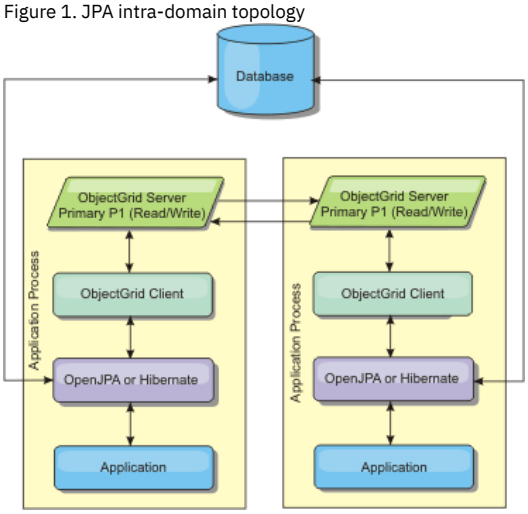
4. **What is the tolerated staleness level of the data?**
   In a system with few JVMs, data replication latency exists for write operations. The goal of the cache is to maintain an ultimate synchronized data view across all JVMs. When you are using the intra-domain topology, a data replication delay 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 1. 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.

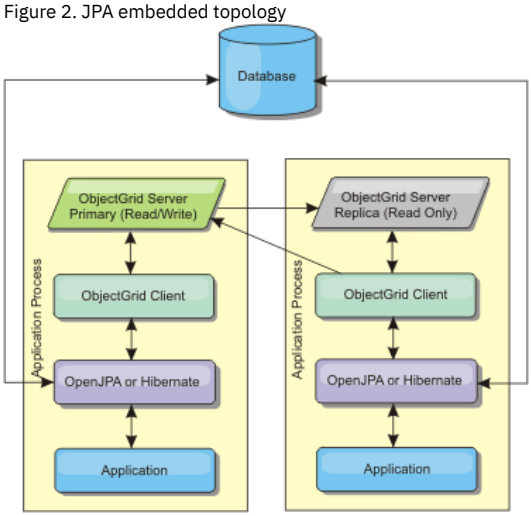
Limitations:

- 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 2. 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.

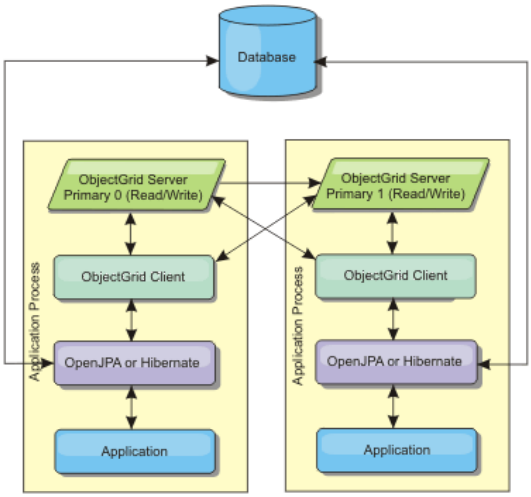
Limitations:

- 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 3. 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.

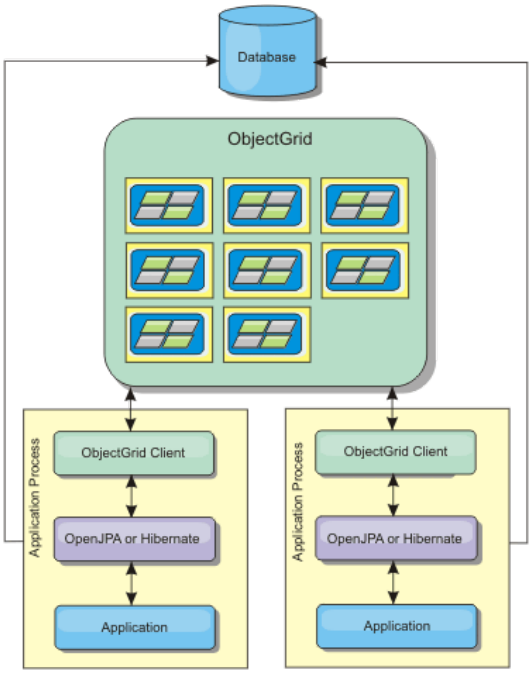
Read about Sizing memory and partition count calculation to maximize the performance of your chosen configuration.

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 4. 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.

- JPA cache configuration properties for both OpenJPA and Hibernate Version 3.0
  WebSphere eXtreme Scale includes level 2 cache plug-ins for both OpenJPA and Hibernate Java Persistence API (JPA) providers. To configure the L2 cache plug-in, you must update properties in the persistence.xml file.
- Configuring the OpenJPA cache plug-in
  You can configure both DataCache and QueryCache implementations for OpenJPA.
- Configuring the Hibernate cache plug-in
  You can enable the cache to use the Hibernate cache plug-in by specifying properties files.

Related tasks:
Configuring the OpenJPA cache plug-in
Troubleshooting multiple data center configurations
Configuring the Hibernate cache plug-in
Related reference:
JPA cache configuration properties for both OpenJPA and Hibernate Version 3.0
Example: OpenJPA ObjectGrid XML files
Example: Hibernate ObjectGrid XML files
Related information:
com.ibm.websphere.objectgrid.openJPA package
com.ibm.websphere.objectgrid.hibernate.cache package

JPA cache configuration properties for both OpenJPA and Hibernate Version 3.0

WebSphere® eXtreme Scale includes level 2 cache plug-ins for both OpenJPA and Hibernate Java™ Persistence API (JPA) providers. To configure the L2 cache plug-in, you must update properties in the persistence.xml file.

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 Configuring JPA loaders.

Properties location

You can configure these properties in the persistence.xml file. The syntax for specifying the properties in this file varies depending if you are using OpenJPA or Hibernate Version 3.0:

- **OpenJPA:** You can set the properties on the DataCache or QueryCache:

  ```
  <property name="openjpa.DataCache"
   value="<object_grid_datacache_class(<property>=<value>,...)"/>
  ```

  or

  ```
  <property name="openjpa.QueryCache"
   value="<object_grid_querycache_class(<property>=<value>,...)"/>
  ```

- **Hibernate Version 3.0:**

  ```
  <property name="objectgrid.configuration"
   value="<property>=<value>,..." />
  ```

Default topology and properties

The following default property values are used if you do not specify any values in the configuration:

- **ObjectGridName**: persistence unit name
- **ObjectGridType**: `EMBEDDED`
- **NumberOfPartitions**: `1` (cannot be changed when ObjectGrid type is `EMBEDDED`)
- **ReplicaMode**: `SYNC`
- **ReplicaReadEnabled**: `TRUE` (cannot be changed when ObjectGrid type is `EMBEDDED`)
- **MaxUsedMemory**: `TRUE`
- **MaxNumberOfReplicas**: `47` (must be less than or equal to the number of Java virtual machines in a distributed system)

Properties

You can configure JPA cache plug-ins with the following properties.

ObjectGridName
: Specifies the unique ObjectGrid name. The default value is the defined persistence unit name. If the persistence unit name is not available from the JPA provider, a generated name is used.

ObjectGridType
: Specifies the type of ObjectGrid.
Valid values:

  `EMBEDDED`
  : The default and recommended configuration type. Its default settings include: `NumberOfPartitions=1`, `ReplicaMode=SYNC`, `ReplicaReadEnabled=true` and `MaxNumberOfReplicas=47`. Use the ReplicaMode parameter to set the replication mode and the MaxNumberOfReplicas parameter to set the maximum number of replicas. If a system has more than 47 Java virtual machines, set the MaxNumberOfReplicas value to be equal to the number of Java virtual machines.

  `EMBEDDED_PARTITION`
  : The type to use when the system needs to cache a large amount of data in a distributed system. The default number of partitions is `47` with a replica mode of `NONE`. In a small system that has only a few Java virtual machines, set the NumberOfPartitions value to be equal or less than the number of Java virtual machines. You can specify the ReplicaMode, NumberOfPartitions, and ReplicaReadEnabled values to tune the system.

  `REMOTE`
  : The cache tries to connect to a remote, distributed ObjectGrid from the catalog service.

MaxNumberOfReplicas
: Specifies the maximum number of replicas to be used for the cache. This value applies to the `EMBEDDED` type only. This number must be equal to or greater than the number of Java virtual machines in a system. The default value is `47`.
Valid values: greater than or equal to `1`

MaxUsedMemory
: **Valid values:** `TRUE` or `FALSE`

  Enables eviction of cache entries when memory becomes constrained. The default value is `TRUE` and evicts data when the JVM heap utilization threshold exceeds 70 percent. You can modify the default JVM heap utilization threshold percentage by setting the memoryThresholdPercentage property in the objectGridServer.properties file and placing this file in the class path. For more information about evictors, see Plug-ins for evicting cache objectsthe information about evictors in the *Product Overview*. For more information about the server properties file, see Server properties file.

NumberOfPartitions
: **Valid values:** greater than or equal to `1`

  Specifies the number of partitions to be used for the cache. This property applies when the ObjectGridType value is set to `EMBEDDED_PARTITION`. The default value is `47`. For the `EMBEDDED` type, the **NumberOfPartitions** value is always `1`.

PlacementScope
: Indicates the granularity of a single instance of a map set.
Valid values:

DOMAIN_SCOPE
(Default) Places one primary shard for each partition on a container server within the catalog service domain. Replica shards for each partition are placed on the other container servers within the catalog service domain.
CONTAINER_SCOPE
Places a primary shard on each container server in the catalog service domain.

PlacementScopeTopology
Defines the linking topology of the container servers within the catalog service domain. This value is only used when the PlacementScope value is set to something other than `DOMAIN_SCOPE`.
Valid values:

HUB
(Default) If the hub topology is selected, then a single data grid is selected to be the hub. Every other data grid connects to the hub. This topology is fairly scalable because the spokes have a single connection. The hub can become a bottleneck and temporary single point of failure. The hub is relocated to another container server when it fails. The advantage to this configuration is more complex arbitration code can be written that allows a single point, the hub, to handle all collisions.
RING
If the ring topology is selected, each data grid is linked with two other data grids. The ordering of the links is not guaranteed. However, each container that starts is likely linked to the first container and last container to be added to the ring. This topology is the most scalable, but only two links can fail before being temporarily cut off. If the container servers fail, links are established among the survivors after the failure has been discovered.

ReplicaMode
Valid values: `SYNC`/`ASYNC`/`NONE`

Specifies the method that is used to copy the cache to the replicas. This property applies when you have the ObjectGridType value set to `EMBEDDED` or `EMBEDDED_PARTITION`. The default value is `NONE` for the `EMBEDDED_PARTITION` type and `SYNC` for the `EMBEDDED` type. If the **ReplicaMode** value is set to `NONE` for the `EMBEDDED` ObjectGridType, the `EMBEDDED` type still uses a **ReplicaMode** of `SYNC`.
ReplicaReadEnabled
Valid values: `TRUE` or `FALSE`

When enabled, clients read from replicas. This property applies to the `EMBEDDED_PARTITION` type. The default value is FALSE for the `EMBEDDED_PARTITION` type. The `EMBEDDED` type always sets the **ReplicaReadEnabled** value to `TRUE`.
writeBehind
For Hibernate providers only: When writeBehind is enabled, updates are temporarily stored in a JVM scope data storage until either the writeBehindInterval or writeBehindMaxBatchSize conditions are met.

Attention: Unless writeBehind is enabled, the other write behind configuration settings are disregarded.
Important: Take care when using the write behind function. Write behind configurations introduce longer latency of data synchronization across all JVMs and a higher chance of lost updates. In a system that has write behind configuration enabled with four or more JVMs, the update performed on one JVM has an approximate 15 second delay before the update becomes available to other JVMs. If any two JVMs update the same entry, the one that flushes the update first loses its update.
Valid values: TRUE or FALSE

Default value: FALSE

writeBehindInterval
For Hibernate providers only: Specifies the time interval in milliseconds to flush updates to the cache.

Valid values: greater than or equal to 1

Default value: 5000 (5 seconds)

writeBehindPoolSize
For Hibernate providers only: Specifies the maximum size of the thread pool used in flushing updates to the cache.

Valid values: greater than or equal to 1

Default value: 5

writeBehindMaxBatchSize
For Hibernate providers only: Specifies the maximum batch size per region cache in flushing updates to the cache. For example, if the size is set to 1000, and the updates stored in the write behind storage of a region cache exceeds 1000 entries, the updates are flushed to the cache, even the specified writeBehindInterval condition is not met. Updates flush to the cache either approximately at the number of seconds specified by the writeBehindInterval value or whenever the size of write behind storage of each region cache exceeds 1000 entries. Note, in the case of the writeBehindMaxBatchSize condition met; only the region cache that meets this condition flushes its updates in write behind storage to cache. A region cache usually corresponds to an entity or a query.

Valid values: greater than or equal to 1

Default value: 1000

Related concepts:
JPA level 2 (L2) cache plug-in
Related tasks:
Configuring the OpenJPA cache plug-in
Troubleshooting multiple data center configurations
Configuring the Hibernate cache plug-in
Related information:
com.ibm.websphere.objectgrid.openJPA package

Configuring the OpenJPA cache plug-in

You can configure both DataCache and QueryCache implementations for OpenJPA.

Before you begin

- You must determine the JPA cache plug-in topology that you want to use. See JPA level 2 (L2) cache plug-in for more information about the different configurations and the properties to set for each topology.
- You must have an application that uses the JPA APIs. If you want to use WebSphere eXtreme Scale APIs to access data with JPA, use the JPA loader. For more information, see Configuring JPA loaders.

Procedure

1. Set properties in your persistence.xml file to configure the OpenJPA cache plug-in: You can set these properties on either the DataCache or Query cache implementation.
   DataCache and QueryCache configurations are independent of one another. You can enable either configuration. However, if both configurations are enabled, the QueryCache configuration uses the same configuration as the DataCache configuration, and its configuration properties are discarded.

   ```
   <property name="openjpa.DataCache"
             value="<object_grid_datacache_class(<property>=<value>,...)"/>
   ```

   or

   ```
   <property name="openjpa.QueryCache"
             value="<object_grid_querycache_class(<property>=<value>,...)"/>
   ```

   Note: You can enable the QueryCache configuration for embedded and embedded-intradomain topologies only.
   You can specify the ObjectGridName property, the ObjectGridType property, and other simple deployment policy-related properties in the property list of the ObjectGrid cache class to customize cache configuration. An example follows:

   ```
   <property name="openjpa.DataCache"
             value="com.ibm.websphere.objectgrid.openjpa.ObjectGridDataCache(
             ObjectGridName=BasicTestObjectGrid,ObjectGridType=EMBEDDED,
             maxNumberOfReplicas=4)"/>
   <property name="openjpa.QueryCache"
             value="com.ibm.websphere.objectgrid.openjpa.ObjectGridQueryCache()"/>
   <property name="openjpa.RemoteCommitProvider" value="sjvm"/>
   ```

   See JPA cache configuration properties for both OpenJPA and Hibernate Version 3.0 for a list of the properties that you can set.
2. In the persistence.xml file, you also must set the openjpa.RemoteCommitProvider property to `sjvm`.

   ```
   <property name="openjpa.RemoteCommitProvider" value="sjvm"/>
   ```

3. Optional: To further customize the data grid used by the cache, you can provide additional settings with XML files.
   For most scenarios, setting cache properties should be sufficient. To further customize the ObjectGrid used by the cache, you can provide OpenJPA ObjectGrid configuration XML files in the META-INF directory similarly to the persistence.xml file. During initialization, the cache tries to locate these XML files and process them if found.

   There are three types of OpenJPA ObjectGrid configuration XML files:
   - openjpa-objectGrid.xml (ObjectGrid configuration)
     File path: META-INF/openjpa-objectGrid.xml

     This file is used to customize ObjectGrid configuration for both the EMBEDDED and EMBEDDED_PARTITION type. With the REMOTE type, this file is ignored. By default, each entity class is mapped to its own BackingMap configuration named as an entity class name within the ObjectGrid configuration. For example, com.mycompany.Employee entity class is mapped to com.mycompany.Employee BackingMap. The default BackingMap configuration is `readOnly="false"`, `copyKey="false"`, `lockStrategy="NONE"`, and `copyMode="NO_COPY"`. You can customize some BackingMaps with your chosen configuration. You can use the `ALL_ENTITY_MAPS` reserved keyword to represent all maps excluding other customized maps listed in the openjpa-objectGrid.xml file. BackingMaps that are not listed in this openjpa-objectGrid.xml file use the default configuration. If customized BackingMaps do not specify the BackingMaps attribute or properties and these attributes are specified in the default configuration, the attribute values from the default configuration are applied. For example, if an entity class is annotated with `timeToLive=30`, the default BackingMap configuration for that entity has a `timeToLive=30`. If the custom openjpa-objectGrid.xml file also includes that BackingMap but does not specify timeToLive value, then the customize BackingMap has a `timeToLive=30` value by default. The openjpa-objectGrid.xml file intends to override or extend the default configuration.

   - openjpa-objectGridDeployment.xml (deployment policy)
     File path: META-INF/openjpa-objectGridDeployment.xml

     This file is used to customize deployment policy. When you are customizing deployment policy, if the openjpa-objectGridDeployment.xml file is provided, the default deployment policy is discarded. All deployment policy attribute values are from the provided openjpa-objectGridDeployment.xml file.

   - openjpa-objectGrid-client-override.xml (client ObjectGrid override configuration)
     File path: META-INF/openjpa-objectGrid-client-override.xml

This file is used to customize a client-side ObjectGrid. By default, the ObjectGrid cache applies a default client override ObjectGrid configuration that disables a near cache. You can enable the near cache by providing the openjpa-objectGrid-client-override.xml file that overrides this configuration. For more information about the settings to change in this file to enable near cache, see Configuring the near cache. The way that the openjpa-objectGrid-client-override.xml file works is similar to the openjpa-objectGrid.xml file. It overrides or extends the default client ObjectGrid override configuration.

Depending on the configured eXtreme Scale topology, you can provide any one of these three XML files to customize that topology.
For both the EMBEDDED and EMBEDDED_PARTITION types, you can provide any one of the three XML files to customize the ObjectGrid, deployment policy, and client ObjectGrid override configuration.

For a REMOTE ObjectGrid, the ObjectGrid cache does not create a dynamic ObjectGrid. Instead, the cache only obtains a client-side ObjectGrid from the catalog service. You can only provide the openjpa-objectGrid-client-override.xml file to customize the client ObjectGrid override configuration.

4. Optional: (Remote configurations only) Set up an external eXtreme Scale system if you want to configure a cache with a REMOTE ObjectGrid type.
You must set up an external eXtreme Scale system if you want to configure a cache with a REMOTE ObjectGrid type. You need both ObjectGrid and ObjectGridDeployment configuration XML files that are based on the persistence.xml file to set up an external system. For examples of these configuration files, see Example: OpenJPA ObjectGrid XML files.

Results

EMBEDDED, EMBEDDED_PARTITION, or intra-domain configuration:

When an application starts, the plug-in automatically detects or starts a catalog service, starts a container server, and connect the container servers to the catalog service. The plug-in then communicates with the ObjectGrid container and its peers that are running in other application server processes using the client connection.

REMOTE configuration:

The deployment policy is specified separately from the JPA application. An external ObjectGrid system has both catalog service and container server processes. You must start a catalog service before starting container servers. See Starting stand-alone servers and Starting container servers for more information.

What to do next

- Develop an OpenJPA application that uses the configuration. For more information, see Example: Using the Hibernate plug-in to preload data into the ObjectGrid cache.
- In a production environment, create catalog service domains for your automatically created processes for your EMBEDDED or EMBEDDED_PARTITION configuration.
  - Stand-alone environment:
    If you are not running your servers inside a WebSphere Application Server process, the catalog service domain hosts and ports are specified using properties file named objectGridServer.properties. This file must be stored in the class path of the application and have the **catalogServiceEndPoints** property defined. The catalog service domain is started independently from the application processes and must be started before the application processes are started.

    The format of the objectGridServer.properties file follows:

    ```
    catalogServiceEndPoints=<hostname1>:<port1>,<hostname2>:<port2>
    ```

  - WebSphere Application Server environment:
    If you are running inside a WebSphere Application Server process, the JPA cache plug-in automatically connects to the catalog service or catalog service domain that is defined for the WebSphere Application Server cell.

- When you are using the `EMBEDDED` or `EMBEDDED_PARTITION` ObjectGridType value in a Java SE environment, use the System.exit(0) method at the end of the program to stop the embedded eXtreme Scale server. Otherwise, the program can become unresponsive.

- Example: OpenJPA ObjectGrid XML files
  OpenJPA ObjectGrid XML files should be created based on the configuration of the persistence unit.

Related concepts:
JPA level 2 (L2) cache plug-in
Related tasks:
Troubleshooting multiple data center configurations
Related reference:
JPA cache configuration properties for both OpenJPA and Hibernate Version 3.0
Example: OpenJPA ObjectGrid XML files
Related information:
com.ibm.websphere.objectgrid.openJPA package

Example: OpenJPA ObjectGrid XML files

OpenJPA ObjectGrid XML files should be created based on the configuration of the persistence unit.

persistence.xml file

A persistence.xml file that is an example that represents the configuration of a persistence unit follows:

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  version="1.0">
  <persistence-unit name="AnnuityGrid">
    <provider>org.apache.openjpa.persistence.PersistenceProviderImpl</provider>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.AnnuityPersistebleObject</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.Annuity</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.FixedAnnuity</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.EquityAnnuity</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.Payout</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.Rider</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.Payor</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.Person</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.AnnuityHolder</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.Contact</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.Address</class>
    <exclude-unlisted-classes>true</exclude-unlisted-classes>

    <properties>
    <!-- Database setting -->


    <!--  enable cache -->
      <property name="openjpa.DataCache"
        value="com.ibm.websphere.objectgrid.openjpa.ObjectGridDataCache(objectGridName=Annuity,
               objectGridType=EMBEDDED, maxNumberOfReplicas=4)" />
      <property name="openjpa.RemoteCommitProvider" value="sjvm" />
      <property name="openjpa.QueryCache"
                value="com.ibm.websphere.objectgrid.openjpa.ObjectGridQueryCache()" />
    </properties>
  </persistence-unit>
</persistence>
```

openjpa-objectGrid.xml file

The openjpa-objectGrid.xml file is used to customize ObjectGrid configuration for both the EMBEDDED and EMBEDDED_PARTITION type. The openjpa-objectGrid.xml file that matches the persistence.xml file follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<?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="Annuity">
      <backingMap name="com.ibm.wssvt.acme.annuity.common.bean.jpa.Annuity" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="com.ibm.wssvt.acme.annuity.common.bean.jpa.Annuity" />
      <backingMap name="com.ibm.wssvt.acme.annuity.common.bean.jpa.Address" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="com.ibm.wssvt.acme.annuity.common.bean.jpa.Address" />
      <backingMap name="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payor" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payor" />
      <backingMap name="com.ibm.wssvt.acme.annuity.common.bean.jpa.Person" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="com.ibm.wssvt.acme.annuity.common.bean.jpa.Person" />
      <backingMap name="com.ibm.wssvt.acme.annuity.common.bean.jpa.Contact" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="com.ibm.wssvt.acme.annuity.common.bean.jpa.Contact" />
      <backingMap name="com.ibm.wssvt.acme.annuity.common.bean.jpa.AnnuityPersistebleObject"
                  readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="com.ibm.wssvt.acme.annuity.common.bean.jpa.AnnuityPersistebleObject" />
      <backingMap name="com.ibm.wssvt.acme.annuity.common.bean.jpa.Rider" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="com.ibm.wssvt.acme.annuity.common.bean.jpa.Rider" />
      <backingMap name="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payout" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payout" />
      <backingMap name="ObjectGridQueryCache" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" pluginCollectionRef="ObjectGridQueryCache"
                  evictionTriggers="MEMORY_USAGE_THRESHOLD"  />
    </objectGrid>
  </objectGrids>
  <backingMapPluginCollections>
    <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsopenjpaxml_com.ibm.wssvt.acme.annu
ity.common.bean.jpa.Annuity">
      <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsopenjpaxml_ObjectTransformer"
            className="com.ibm.ws.objectgrid.openjpa.ObjectGridPCDataObjectTransformer" />
      <bean id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsopenjpaxml_Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsopenjpaxml_com.ibm.wssvt.acme.annu
ity.common.bean.jpa.Address">
```

```
      <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsopenjpaxml_ObjectTransformer"
            className="com.ibm.ws.objectgrid.openjpa.ObjectGridPCDataObjectTransformer" />
      <bean id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsopenjpaxml_Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsopenjpaxml_com.ibm.wssvt.acme.annu
ity.common.bean.jpa.Payor">
      <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsopenjpaxml_ObjectTransformer"
            className="com.ibm.ws.objectgrid.openjpa.ObjectGridPCDataObjectTransformer" />
      <bean id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsopenjpaxml_Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsopenjpaxml_com.ibm.wssvt.acme.annu
ity.common.bean.jpa.Person">
      <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsopenjpaxml_ObjectTransformer"
            className="com.ibm.ws.objectgrid.openjpa.ObjectGridPCDataObjectTransformer" />
      <bean id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsopenjpaxml_Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsopenjpaxml_com.ibm.wssvt.acme.annu
ity.common.bean.jpa.Contact">
      <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsopenjpaxml_ObjectTransformer"
            className="com.ibm.ws.objectgrid.openjpa.ObjectGridPCDataObjectTransformer" />
      <bean id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsopenjpaxml_Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection

id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsopenjpaxml_com.ibm.wssvt.acme.annu
ity.common.bean.jpa.AnnuityPersistebleObject">
      <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsopenjpaxml_ObjectTransformer"
            className="com.ibm.ws.objectgrid.openjpa.ObjectGridPCDataObjectTransformer" />
      <bean id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsopenjpaxml_Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsopenjpaxml_com.ibm.wssvt.acme.annu
ity.common.bean.jpa.Rider">
      <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsopenjpaxml_ObjectTransformer"
            className="com.ibm.ws.objectgrid.openjpa.ObjectGridPCDataObjectTransformer" />
      <bean id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsopenjpaxml_Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsopenjpaxml_com.ibm.wssvt.acme.annu
ity.common.bean.jpa.Payout">
      <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsopenjpaxml_ObjectTransformer"
            className="com.ibm.ws.objectgrid.openjpa.ObjectGridPCDataObjectTransformer" />
      <bean id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsopenjpaxml_Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsopenjpaxml_ObjectGridQueryCache">
      <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsopenjpaxml_MapIndexPlugin"
className="com.ibm.websphere.objectgrid.plugins.index.HashIndex" >
          <property name="Name" type="java.lang.String"
                    value="QueryCacheKeyIndex" description="name of index"/>
          <property name="POJOKeyIndex" type="boolean" value="true" description="POJO Key Index "/>
      </bean>
      <bean id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsopenjpaxml_Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
  </backingMapPluginCollections>
</objectGridConfig>
```

Important:

1. Each entity is mapped to a BackingMap named as the fully qualified entity class name.
   By default, entities are part of the second level cache. In the Entity classes which needs to be excluded from caching, You can include the `@DataCache(enabled=false)` annotation on the entity class that you want to exclude from L2 cache:

```
import org.apache.openjpa.persistence.DataCache;
@Entity
@DataCache(enabled=false)
public class OpenJPACacheTest { ... }
```

2. If entity classes are in an inheritance hierarchy, child classes map to the parent BackingMap. The inheritance hierarchy shares a single BackingMap.
3. The ObjectGridQueryCache map is required to support QueryCache.
4. The backingMapPluginCollection for each entity map must have the ObjectTransformer using the com.ibm.ws.objectgrid.openjpa.ObjectGridPCDataObjectTransformer class.
5. The backingMapPluginCollection for ObjectGridQueryCache map must have the key index named as QueryCacheKeyIndex as shown in the sample.
6. The evictor is optional for each map.

openjpa-objectGridDeployment.xml file

The openjpa-objectGridDeployment.xml file is used to customize deployment policy. The openjpa-objectGridDeployment.xml file that matches the persistence.xml file follows:

```
openjpa-objectGridDeployment.xml

<?xml version="1.0" encoding="UTF-8" ?>
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy ../deploymentPolicy.xsd"
  xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">
  <objectgridDeployment objectgridName="Annuity">
    <mapSet name="MAPSET_Annuity" numberOfPartitions="1" numInitialContainers="1"
            minSyncReplicas="0" maxSyncReplicas="4" maxAsyncReplicas="0"
            replicaReadEnabled="true">
      <map ref="com.ibm.wssvt.acme.annuity.common.bean.jpa.Annuity" />
      <map ref="com.ibm.wssvt.acme.annuity.common.bean.jpa.Address" />
      <map ref="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payor" />
      <map ref="com.ibm.wssvt.acme.annuity.common.bean.jpa.Person" />
      <map ref="com.ibm.wssvt.acme.annuity.common.bean.jpa.Contact" />
      <map ref="com.ibm.wssvt.acme.annuity.common.bean.jpa.AnnuityPersistebleObject" />
      <map ref="com.ibm.wssvt.acme.annuity.common.bean.jpa.Rider" />
      <map ref="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payout" />
      <map ref="ObjectGridQueryCache" />
    </mapSet>
  </objectgridDeployment>
</deploymentPolicy>
```

Note: The ObjectGridQueryCache map is required to support QueryCache.

Related concepts:
JPA level 2 (L2) cache plug-in

Related tasks:
Configuring the OpenJPA cache plug-in
Troubleshooting multiple data center configurations

Related information:
com.ibm.websphere.objectgrid.openJPA package

Configuring the Hibernate cache plug-in

You can enable the cache to use the Hibernate cache plug-in by specifying properties files.

Before you begin

- You must determine the JPA cache plug-in topology that you want to use. See JPA level 2 (L2) cache plug-in for more information about the different configurations.
- You must have an application that uses the JPA APIs. If you want to use WebSphere eXtreme Scale APIs to access data with JPA, use the JPA loader. For more information, see Configuring JPA loaders.

Procedure

1. If you are using WebSphere Application Server, place the Java Archive (JAR) files in the appropriate locations for your configuration. 8.5
   The Hibernate cache plug-in is packaged in the oghibernate-cache.jar file and is installed in the *was_root*/optionalLibraries/ObjectGrid directory. To use the Hibernate cache plug-in, you have to include the oghibernate-cache.jar file in the Hibernate library. For example, if you include the Hibernate library in your application, also must include the oghibernate-cache.jar file. If you define a shared library to include Hibernate library, you must add the oghibernate-cache.jar file into the shared library directory.

   eXtreme Scale does not install the cglib.jar file in the WebSphere Application Server environment. If you have existing applications or shared libraries, such as hibernate, which depend on the cglib.jar, locate the cglib.jar file and include it in the classpath. For example, if your application includes all hibernate library JAR files, but excludes the cglib.jar file available with hibernate, you must include the cglib.jar file that comes from Hibernate in your application.

2. Set properties in your persistence.xml file to configure the Hibernate cache plug-in
   The syntax for setting properties in the persistence.xml file follows:

```
<property name="hibernate.cache.provider_class"
        value="com.ibm.websphere.objectgrid.hibernate.cache.ObjectGridHibernateCacheProvider" />
<property name="hibernate.cache.use_query_cache" value="true"/>
<property name="objectgrid.configuration" value="<property>=<value>,..." />
<property name="objectgrid.hibernate.regionNames" value="<regionName>,.." />
```

- hibernate.cache.provider_class : The value of the provider_class property is the com.ibm.websphere.objectgrid.hibernate.cache.ObjectGridHibernateCacheProvider class.
- hibernate.cache.use_query_cache: To enable query cache, set the value to `true` on the use_query_cache property.
  Note: You can enable the query cache for embedded and embedded-intradomain topologies only.
- objectgrid.configuration: Use the objectgrid.configuration property to specify eXtreme Scale cache configuration properties, including the ObjectGridType attribute that specifies how to place the shards on the data grid.
  You must specify a unique ObjectGridName property value to avoid potential naming conflicts. The other eXtreme Scale cache configuration properties are optional.

  To enable write-behind caching, use the following write-behind attributes on the objectgrid.configuration property. When write-behind caching is enabled, updates are temporarily stored in a JVM scope data storage until either the writeBehindInterval or writeBehindMaxBatchSize conditions are met, when the data is flushed to the cache.

  ```
  writeBehind=true, writeBehindInterval=5000, writeBehindPoolSize=10, writeBehindMaxBatchSize=1000
  ```

  Attention: Unless writeBehind is enabled, the other write behind configuration settings are disregarded.
  For more information about the values that you can set in the objectgrid.configuration property, see JPA cache configuration properties for both OpenJPA and Hibernate Version 3.0.

- objectgrid.hibernate.regionNames: The objectgrid.hibernate.regionNames property is optional and should be specified when the regionNames values are defined after the eXtreme Scale cache is initialized. Consider the example of an entity class that is mapped to a regionName with the entity class unspecified in the persistence.xml file or not included in the Hibernate mapping file. Further, say it does have Entity annotation. Then, the regionName for this entity class is resolved at class loading time when the eXtreme Scale cache is initialized. Another example is the Query.setCacheRegion(String regionName) method that runs after the eXtreme Scale cache initialization. In these situations, include all possible dynamic determined regionNames in the objectgrid.hibernate.regionNames property so that the eXtreme Scale cache can prepare BackingMaps for all regionNames.

3. Optional: To further customize the data grid used by the cache, you can provide additional settings with XML files.
   For most scenarios, setting cache properties should be sufficient. To further customize the ObjectGrid used by the cache, you can provide Hibernate ObjectGrid configuration XML files in the META-INF directory similarly to the persistence.xml file. During initialization, the cache tries to locate these XML files and process them if found.

   There are three types of Hibernate ObjectGrid configuration XML files:
   - hibernate-objectGrid.xml (ObjectGrid configuration)
     File path: META-INF/hibernate-objectGrid.xml

     By default, each entity class has an associated regionName (default to entity class name) that is mapped to a BackingMap configuration named as regionName within the ObjectGrid configuration. For example, the com.mycompany.Employee entity class has an associated regionName default to com.mycompany.Employee BackingMap. The default BackingMap configuration is readOnly="false", copyKey="false", lockStrategy="NONE", and `copyMode="NO_COPY"`. You can customize some BackingMaps with a chosen configuration. The reserved key word "ALL_ENTITY_MAPS" can be used to represent all maps excluding other customized maps listed in the hibernate-objectGrid.xmlfile. BackingMaps that are not listed in this hibernate-objectGrid.xml file use the default configuration.

   - hibernate-objectGridDeployment.xml (deployment policy)
     File path: META-INF/hibernate-objectGridDeployment.xml

     This file is used to customize deployment policy. When you are customizing deployment policy, if the hibernate-objectGridDeployment.xml is provided, the default deployment policy is discarded. All deployment policy attribute values will come from the provided hibernate-objectGridDeployment.xml file.

   - hibernate-objectGrid-client-override.xml (client ObjectGrid override configuration)
     File path: META-INF/hibernate-objectGrid-client-override.xml

     This file is used to customize a client-side ObjectGrid. By default, the ObjectGrid cache applies a default client override configuration that disables the near cache. You can enable the near cache by providing the hibernate-objectGrid-client-override.xml file that overrides this configuration. For more information about the settings to change in this file to enable near cache, see Configuring the near cache. The way that the hibernate-objectGrid-client-override.xml file works is similar to the hibernate-objectGrid.xml file. It overrides or extends the default client ObjectGrid override configuration.

   Depending on the configured eXtreme Scale topology, you can provide any one of these three XML files to customize that topology.
   For both the EMBEDDED and EMBEDDED_PARTITION type, you can provide any one of the three XML files to customize the ObjectGrid, deployment policy, and client ObjectGrid override configuration.

   For a REMOTE ObjectGrid, the cache does not create a dynamic ObjectGrid. The cache only obtains a client-side ObjectGrid from the catalog service. You can only provide a hibernate-objectGrid-client-override.xml file to customize client ObjectGrid override configuration.

4. Optional: (Remote configurations only) Set up an external eXtreme Scale system if you want to configure a cache with a REMOTE ObjectGrid type. You also need to specify the libraries and their dependencies in the classpath for the eXtreme Scale container servers.
   You must set up an external eXtreme Scale system if you want to configure a cache with a REMOTE ObjectGrid type. You need both ObjectGrid and ObjectGridDeployment configuration XML files that are based on the persistence.xml file to set up an external system. For examples of these configuration files, seeExample: Hibernate ObjectGrid XML files.

Results

EMBEDDED or EMBEDDED_PARTITION configuration:

When an application starts, the plug-in automatically detects or starts a catalog service, starts a container server, and connect the container servers to the catalog service. The plug-in then communicates with the ObjectGrid container and its peers that are running in other application server processes using the client connection.

Each JPA entity has an independent backing map assigned using the class name of the entity. Each BackingMap has the following attributes.

- `readOnly="false"`
- `copyKey="false"`
- `lockStrategy="NONE"`
- `copyMode="NO_COPY"`

REMOTE configuration:

The deployment policy is specified separately from the JPA application. An external ObjectGrid system has both catalog service and container server processes. You must start a catalog service before starting container servers. See Starting stand-alone servers and Starting container servers for more information.

What to do next

- Develop a Hibernate application that uses the configuration. For more information, see Example: Using the Hibernate plug-in to preload data into the ObjectGrid cache.
- In a production environment, create catalog service domains for your automatically created processes for your EMBEDDED or EMBEDDED_PARTITION configuration.
  - Stand-alone environment:
    If you are not running your servers inside a WebSphere Application Server process, the catalog service domain hosts and ports are specified using properties file named objectGridServer.properties. This file must be stored in the class path of the application and have the **catalogServiceEndPoints** property defined. The catalog service domain is started independently from the application processes and must be started before the application processes are started.

    The format of the objectGridServer.properties file follows:

    ```
    catalogServiceEndPoints=<hostname1>:<port1>,<hostname2>:<port2>
    ```

  - WebSphere Application Server environment:
    If you are running inside a WebSphere Application Server process, the JPA cache plug-in automatically connects to the catalog service or catalog service domain that is defined for the WebSphere Application Server cell. However, using an objectGridServer.properties file with defined `catalogServiceEndPoints` will cause problems because it will try to establish a connection to that catalog server instead of the one in WebSphere Application Server.
- When you are using the `EMBEDDED` or `EMBEDDED_PARTITION` ObjectGridType value in a Java SE environment, use the System.exit(0) method at the end of the program to stop the embedded eXtreme Scale server. Otherwise, the program can become unresponsive.
- Example: Hibernate ObjectGrid XML files
  Create Hibernate ObjectGrid XML files based on the configuration of a persistence unit.

Related concepts:
JPA level 2 (L2) cache plug-in
Related tasks:
Troubleshooting multiple data center configurations
Related reference:
JPA cache configuration properties for both OpenJPA and Hibernate Version 3.0
Example: Hibernate ObjectGrid XML files
Example: Hibernate ObjectGrid XML files
Related information:
com.ibm.websphere.objectgrid.hibernate.cache package

Example: Hibernate ObjectGrid XML files

Create Hibernate ObjectGrid XML files based on the configuration of a persistence unit.

persistence.xml file

An example persistence.xml file that represents the configuration of a persistence unit follows:

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  version="1.0">
  <persistence-unit name="AnnuityGrid">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>

    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.AnnuityPersistebleObject</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.Annuity</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.FixedAnnuity</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.EquityAnnuity</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.Payout</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.Rider</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.Payor</class>
```

```
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.Person</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.AnnuityHolder</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.Contact</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.Address</class>

    <exclude-unlisted-classes>true</exclude-unlisted-classes>

    <properties>
      <property name="hibernate.show_sql" value="false" />
      <property name="hibernate.connection.url" value="jdbc:db2:Annuity" />
      <property name="hibernate.connection.driver_class" value="com.ibm.db2.jcc.DB2Driver" />
      <property name="hibernate.default_schema" value="EJB30" />

      <!-- Cache -->
      <property name="hibernate.cache.provider_class"
        value="com.ibm.websphere.objectgrid.hibernate.cache.ObjectGridHibernateCacheProvider" />
      <property name="hibernate.cache.use_query_cache" value="true" />
      <property name="objectgrid.configuration" value="ObjectGridType=EMBEDDED,
                ObjectGridName=Annuity, MaxNumberOfReplicas=4" />
    </properties>
  </persistence-unit>

</persistence>
```

hibernate-objectGridDeployment.xml file

Use the hibernate-objectGridDeployment.xml file to optionally customize the deployment policy. If you provide this file in the META-INF/hibernate-objectGridDeployment.xml directory, the default deployment policy is overridden by the configuration in this file.

```
<?xml version="1.0" encoding="UTF-8" ?>
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy ../deploymentPolicy.xsd"
  xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">
  <objectgridDeployment objectgridName="Annuity">
    <mapSet name="MAPSET_Annuity" numberOfPartitions="1" numInitialContainers="1" minSyncReplicas="0"
          maxSyncReplicas="4" maxAsyncReplicas="0" replicaReadEnabled="true">
      <map ref="com.ibm.wssvt.acme.annuity.common.bean.jpa.Annuity" />
      <map ref="defaultCacheMap" />
      <map ref="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payor" />
      <map ref="com.ibm.wssvt.acme.annuity.common.bean.jpa.Contact" />
      <map ref="com.ibm.wssvt.acme.annuity.common.bean.jpa.Person" />
      <map ref="com.ibm.wssvt.acme.annuity.common.bean.jpa.Rider" />
      <map ref="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payout" />
      <map ref="org.hibernate.cache.UpdateTimestampsCache" />
      <map ref="org.hibernate.cache.StandardQueryCache" />
    </mapSet>
  </objectgridDeployment>
</deploymentPolicy>
```

hibernate-objectGrid.xml file

If you are not using Hibernate with the Java Persistence API (JPA), use the following example hibernate-objectGrid.xml to create your Hibernate configuration:

```
<?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="Annuity">
      <backingMap name="com.ibm.wssvt.acme.annuity.common.bean.jpa.Annuity" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="com.ibm.wssvt.acme.annuity.common.bean.jpa.Annuity" />
      <backingMap name="defaultCacheMap" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="defaultCacheMap" />
      <backingMap name="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payor" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payor" />
      <backingMap name="com.ibm.wssvt.acme.annuity.common.bean.jpa.Contact" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="com.ibm.wssvt.acme.annuity.common.bean.jpa.Contact" />
      <backingMap name="com.ibm.wssvt.acme.annuity.common.bean.jpa.Person" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="com.ibm.wssvt.acme.annuity.common.bean.jpa.Person" />
      <backingMap name="com.ibm.wssvt.acme.annuity.common.bean.jpa.Rider" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="com.ibm.wssvt.acme.annuity.common.bean.jpa.Rider" />
      <backingMap name="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payout" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payout" />
      <backingMap name="org.hibernate.cache.UpdateTimestampsCache" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="org.hibernate.cache.UpdateTimestampsCache" />
      <backingMap name="org.hibernate.cache.StandardQueryCache" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="org.hibernate.cache.StandardQueryCache" />
    </objectGrid>
```

```
  </objectGrids>
  <backingMapPluginCollections>
    <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxshibernatexml_com.ibm.wssvt.acme.an
nuity.common.bean.jpa.Annuity">
      <bean id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxshibernatexml_Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxshibernatexml_defaultCacheMap">
      <bean id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxshibernatexml_Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxshibernatexml_com.ibm.wssvt.acme.an
nuity.common.bean.jpa.Payor">
      <bean id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxshibernatexml_Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxshibernatexml_com.ibm.wssvt.acme.an
nuity.common.bean.jpa.Contact">
      <bean id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxshibernatexml_Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxshibernatexml_com.ibm.wssvt.acme.an
nuity.common.bean.jpa.Person">
      <bean id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxshibernatexml_Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxshibernatexml_com.ibm.wssvt.acme.an
nuity.common.bean.jpa.Rider">
      <bean id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxshibernatexml_Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxshibernatexml_com.ibm.wssvt.acme.an
nuity.common.bean.jpa.Payout">
      <bean id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxshibernatexml_Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxshibernatexml_org.hibernate.cache.U
pdateTimestampsCache">
      <bean id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxshibernatexml_Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>

    </backingMapPluginCollection>
    <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxshibernatexml_org.hibernate.cache.S
tandardQueryCache">
      <bean id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxshibernatexml_Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
  </backingMapPluginCollections>
</objectGridConfig>
```

Note: The org.hibernate.cache.UpdateTimestampsCache, org.hibernate.cache.StandardQueryCache and defaultCacheMap maps are required.

Related concepts:
JPA level 2 (L2) cache plug-in

Related tasks:
Configuring the Hibernate cache plug-in
Troubleshooting multiple data center configurations
Configuring the Hibernate cache plug-in

Related information:
com.ibm.websphere.objectgrid.hibernate.cache package

Configuring a Spring cache provider

8.5+ 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.

Before you begin

- You must have an application that is using Spring Framework Version 3.1 or later.
- Your application must declare the methods to cache by using annotations. For more information about updating your application for cache abstraction, see Spring Framework Reference Documentation : Cache abstraction.
- Ensure that the ogclient.jar file is in the classpath for the Spring application.
- If the JVM on which your application is running is not the JVM that is installed by WebSphere eXtreme Scale Client, you must add the following JVM argument so that the IBM Object Request Broker (ORB) is used:

  ```
  -Djava.endorsed.dirs=wxs_root/lib/endorsed
  ```

- You must start a catalog server. For more information, see Starting a stand-alone catalog service.

About this task

By using the cache abstraction in the Spring framework, you can reduce the number of times that your Spring application methods run. When configured, the results of a particular method are placed in the cache. When the method is run again, the abstraction checks the cache to see if the method results are already in the cache. If the results are in the cache, the results are returned from the cache and the method does not run again. Therefore, you can reduce the number of times that expensive methods run, also decreasing the average response time of your application.

Procedure

1. Configure your container servers to use the configuration files for Spring.
   You must start container servers before the Spring application that accesses the cache starts. To start container servers, see Starting stand-alone servers.

   The default XML configuration files for starting a container server for the eXtreme Scale Spring cache provider are in one of the following locations:
   - Stand-alone installations: *wxs_install_root*/ObjectGrid/spring/etc
   - WebSphere Application Server installations: *was_root*/optionalLibraries/ObjectGrid/spring/etc

   The files are named spring-remote-objectgrid.xml and spring-remote-deployment.xml. You can use these files as-is, customize these files, or create your own configuration files.
   Run the following command to start a stand-alone container server for the eXtreme Scale Spring cache provider. Run the following command from the *wxs_home*/ObjectGrid/bin directory: Windows

   ```
   startOgServer.bat container1 -objectGridFile ../spring/etc/spring-remote-objectgrid.xml
   -deploymentPolicyFile ../spring/etc/spring-remote-deployment.xml
   ```

   UNIX

   ```
   startOgServer.sh container1 -objectGridFile ../spring/etc/spring-remote-objectgrid.xml
   -deploymentPolicyFile ../spring/etc/spring-remote-deployment.xml
   ```

2. Configure Spring Inversion of Control (IoC) container to use WebSphere eXtreme Scale as the cache provider. The WebSphere eXtreme Scale cache implementation resides under the com.ibm.websphere.objectgrid.spring package. Define the following beans in your Spring IoC container configuration.

   ```
   <bean
   id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsspringprovide_wxsCSDomain"
   class="com.ibm.websphere.objectgrid.spring.ObjectGridCatalogServiceDomainBean"
     p:catalog-service-endpoints="CATALOG_SERVICE_ENDPOINTS"
     p:client-override-xml="CLIENT_OVERRIDE_XML (optional)"
     p:client-security-config="CLIENT_SECURITY_CONFIG (optional)" />

   <bean
   id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsspringprovide_wxsGridClient"
   class="com.ibm.websphere.objectgrid.spring.ObjectGridClientBean"
     p:object-grid-name="OBJECT_GRID_NAME(optional)"
     p:catalog-service-domain-ref="wxsCSDomain" />

   <bean
   id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsspringprovide_cacheManager"
   class="org.springframework.cache.support.SimpleCacheManager">
     <property name="caches">
       <set>
         <bean class="com.ibm.websphere.objectgrid.spring.ObjectGridCache"
            p:name="CACHE_NAME"
            p:map-name="MAP_NAME (optional)"
                  p:object-grid-client-ref="wxsGridClient" />
       </set>
     </property>
   </bean>
   ```

   CATALOG_SERVICE_ENDPOINTS
   Specifies the Object Request Broker (ORB) host and port number.
   CLIENT_OVERRIDE_XML (optional)
   Specifies an absolute or relative path to an ObjectGrid XML file on which to alter settings on the client side as a Spring resource. For information about specifying resources in Spring, see Spring Framework Reference Documentation: Resources.
   Example:`p:client-override-xml="file:/path/to/objectgrid.xml"`

   Example:`p:client-override-xml="classpath:com/example/app/override-objectgrid.xml"`

   Example:`p:client-override-xml="http://myserver/override-objectgrid.xml"`

   Example:`p:client-override-xml="ftp://myserver/override-objectgrid.xml"`

CLIENT_SECURITY_CONFIG (optional)
Specifies an absolute or relative path to a client.properties file as a Spring resource. For information about specifying resources in Spring, see Spring Framework Reference Documentation: Resources.
Example: `p:client-security-config="file:/path/to/client.properties"`

OBJECT_GRID_NAME (optional)
Specifies the ObjectGrid name. This parameter is not needed if the container servers are started with the provided XML configuration files. This parameter must be consistent with the XML configuration files that are used to start the container servers.

CACHE_NAME
Specifies the name of the cache that is specified in your Spring caching application.

MAP_NAME (optional)
Specifies the name of the backing map for a cache. This parameter is not needed if the container servers are started with the provided XML configuration files. This parameter must be consistent with the XML configuration files that are used to start the container servers. If you use the provided XML configuration files, the *MAP_NAME* value is not needed. The maps for the data grid are created automatically when the Spring application runs. The dynamic map name starts with `IBM_SPRING_PARTITIONED_`. For example: `IBM_SPRING_PARTITIONED_1`, `IBM_SPRING_PARTITIONED_2`, and so on.

Example

The following snippet creates two caches, named `default` and `books` hosted by the catalog service domain at `localhost:2809`.

```
<bean id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsspringprovide_wxsCSDomain"
class="com.ibm.websphere.objectgrid.spring.ObjectGridCatalogServiceDomainBean"
        p:catalog-service-endpoints ="localhost:2809" />
<bean id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsspringprovide_wxsGridClient"
class="com.ibm.websphere.objectgrid.spring.ObjectGridClientBean"
        p:catalog-service-domain-ref="wxsCSDomain" />
<bean id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsspringprovide_cacheManager"
class="org.springframework.cache.support.SimpleCacheManager">
        <property name="caches">
                <set>
                        <bean class="com.ibm.websphere.objectgrid.spring.ObjectGridCache"
                                p:name="default"
                                p:object-grid-client-ref="wxsGridClient" />
                        <bean class="com.ibm.websphere.objectgrid.spring.ObjectGridCache"
                                p:name="books"
                                p:object-grid-client-ref="wxsGridClient" />
                </set>
        </property>
</bean>
```

Related concepts:
Spring framework overview
Related reference:
ObjectGrid descriptor XML file
Deployment policy descriptor XML file

Configuring database integration

You can use WebSphere® eXtreme Scale to lower the load on databases. You can use a Java Persistence API (JPA) between WebSphere eXtreme Scale and the database to integrate changes as a loader.

Before you begin

For a summary of the various topologies that you can create with a database, see Database integration: Write-behind, in-line, and side caching.

- Configuring JPA loaders
  A Java™ Persistence API (JPA) Loader is a plug-in implementation that uses JPA to interact with the database.

Configuring JPA loaders

A Java™ Persistence API (JPA) Loader is a plug-in implementation that uses JPA to interact with the database.

Before you begin

- You must have a JPA implementation, such as Hibernate or OpenJPA.
- Your database can be any back end that is supported by the chosen JPA provider.
- Decide whether you are going to use the JPALoader plug-in or the JPAEntityLoader plug-in. 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.
  Note: If you are using the JPA APIs to access the JPA data source, use the JPA L2 cache plug-in. The cache plug-in introduces the data grid between your application and the JPA data source, while still using a JPA application. For more information, see JPA level 2 (L2) cache plug-in.

About this task

For more information about how the Java Persistence API (JPA) Loader works, see JPA Loaders.

Procedure

1. Configure the necessary parameters that JPA requires to interact with a database.
   The following parameters are required. These parameters are configured in the JPALoader or JPAEntityLoader bean, and JPATxCallback bean.

   - persistenceUnitName: Specifies the persistence unit name. This parameter is required for two purposes: for creating a JPA EntityManagerFactory, and for locating the JPA entity metadata in the persistence.xml file. This attribute is set on the JPATxCallback bean.
   - JPAPropertyFactory: Specifies the factory to create a persistence property map to override the default persistence properties. This attribute is set on the JPATxCallback bean. To set this attribute, Spring style configuration is required.
   - entityClassName: Specifies the entity class name that is required to use JPA methods, such as EntityManager.persist, EntityManager.find, and so on. The JPALoader plug-in requires this parameter, but the parameter is optional for JPAEntityLoader. For the JPAEntityLoader plug-in, if an entityClassName parameter is not configured, the entity class configured in the ObjectGrid entity map is used. You must use the same class for the eXtreme Scale EntityManager and for the JPA provider. This attribute is set on the JPALoader or JPAEntityLoader bean.
   - preloadPartition: Specifies the partition at which the map preload is started. If the preload partition is less than zero, or greater than the total number of partitions minus 1, the map preload is not started. The default value is -1, which means the preload does not start by default. This attribute is set on the JPALoader or JPAEntityLoader bean.

   Other than the four JPA parameters to be configured in eXtreme Scale, JPA metadata are used to retrieve the key from the JPA entities. The JPA metadata can be configured as annotation, or as an orm.xml file specified in the persistence.xml file. It is not part of the eXtreme Scale configuration.

2. Configure XML files for the JPA configuration.
   To configure a JPALoader or JPAEntityLoader, see Plug-ins for communicating with databases.

   Configure a JPATxCallback transaction callback along with the loader configuration. The following example is an ObjectGrid XML descriptor file (objectgrid.xml), that has a JPAEntityLoader and JPATxCallback configured:

```
configuring a loader including callback - XML example
<?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="JPAEM" entityMetadataXMLFile="jpaEMD.xml">
        <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsjpaload_TransactionCallback"
           className="com.ibm.websphere.objectgrid.jpa.JPATxCallback">
           <property
              name="persistenceUnitName"
              type="java.lang.String"
              value="employeeEMPU" />
       </bean>
       <backingMap name="Employee" pluginCollectionRef="Employee" />
    </objectGrid>
  </objectGrids>

  <backingMapPluginCollections>
    <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsjpaload_Employee">
        <bean id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsjpaload_Loader"
          className="com.ibm.websphere.objectgrid.jpa.JPAEntityLoader">
            <property
             name="entityClassName"
             type="java.lang.String"
             value="com.ibm.ws.objectgrid.jpa.test.entity.Employee"/>
        </bean>
    </backingMapPluginCollection>
  </backingMapPluginCollections>
</objectGridConfig>
```

   If you want to configure a JPAPropertyFactory, you have to use a Spring style configuration. The following is an XML configuration file sample,JPAEM_spring.xml which configures a Spring bean to be used for eXtreme Scale configurations.

```
configuring a loader including JPA property factory - XML example
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"

xmlns:objectgrid="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsjpaload_http://
www.ibm.com/schema/objectgrid"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

  <objectgrid:JPAEntityLoader
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsjpaload_jpaLoader"
                entityClassName="com.ibm.ws.objectgrid.jpa.test.entity.Employee"/>
  <objectgrid:JPATxCallback
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsjpaload_jpaTxCallback"
persistenceUnitName="employeeEMPU" />
</beans>
```

The Objectgrid.xml configuration XML file follows. Notice the ObjectGrid name is `JPAEM`, which matches the ObjectGrid name in the JPAEM_spring.xml Spring configuration file.

```
JPAEM loader configuration - XML example
<?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="JPAEM" entityMetadataXMLFile="jpaEMD.xml">
      <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsjpaload_TransactionCallback"
           className="{spring}jpaTxCallback"/>
       <backingMap name="Employee" pluginCollectionRef="Employee"
                   writeBehind="T4"/>
     </objectGrid>
  </objectGrids>

  <backingMapPluginCollections>
    <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsjpaload_Employee">
      <bean id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsjpaload_Loader"
className="{spring}jpaLoader" />
    </backingMapPluginCollection>
  </backingMapPluginCollections>
</objectGridConfig>
```

An entity can be annotated with both the JPA annotations and eXtreme Scale entity manager annotations. Each annotation has an XML equivalent that can be used. Thus, eXtreme Scale added the Spring namespace. You can also configure these using the Spring namespace support. For more information, see Spring framework overview.

- Configuring a JPA time-based data updater
  You can configure a time-based database update using XML for a local or distributed eXtreme Scale configuration. You can also configure a local configuration programmatically.

Related concepts:
Programming for JPA integration
Configuring cache integration
Related tasks:
Troubleshooting loaders

Configuring a JPA time-based data updater

You can configure a time-based database update using XML for a local or distributed eXtreme Scale configuration. You can also configure a local configuration programmatically.

About this task

For more information about how the Java™ Persistence API (JPA) time-based data updater works, see JPA time-based data updater.

Procedure

Create a timeBasedDBUpdate configuration.

- **With an XML file:**
  The following example shows an objectgrid.xml file that contains a timeBasedDBUpdate configuration:

```
JPA time-based updater - XML example
<?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="changeOG"
            entityMetadataXMLFile="userEMD.xml">
            <backingMap name="user" >
                <timeBasedDBUpdate timestampField="rowChgTs"
                      persistenceUnitName="userderby"
                      entityClass="com.test.UserClass"
                      mode="INVALIDATE_ONLY"
                />
            </backingMap>
        </objectGrid>
    </objectGrids>
    <backingMapPluginCollections>
</objectGridConfig>
```

In this example, the map "user" is configured with time-based database update. The database update mode is INVALIDATE_ONLY, and the timestamp field is rowChgTs.

When the distributed ObjectGrid "changeOG" is started in the container server, a time-based database update thread is automatically started in partition 0.

- **Programmatically:**
  If you create a local ObjectGrid, you can also create a TimeBasedDBUpdateConfig object and set it on the BackingMap instance:

  ```
  public void setTimeBasedDBUpdateConfig(TimeBasedDBUpdateConfig dbUpdateConfig);
  ```

  For more information about setting an object on the BackingMap instance, see BackingMap interface
  Alternatively, you can annotate the timestamp field in the entity class using the com.ibm.websphere.objectgrid.jpa.dbupdate.annotation.Timestamp annotation. By configuring the value in the class, you do not have to configure the timestampField in the XML configuration.

What to do next

Start the JPA time-based data updater. See Starting the JPA time-based updater for more information.

Configuring REST data services

You can use WebSphere® eXtreme Scale REST data service with WebSphere Application Server version 7.0,WebSphere Application Server Community Edition and Apache Tomcat.

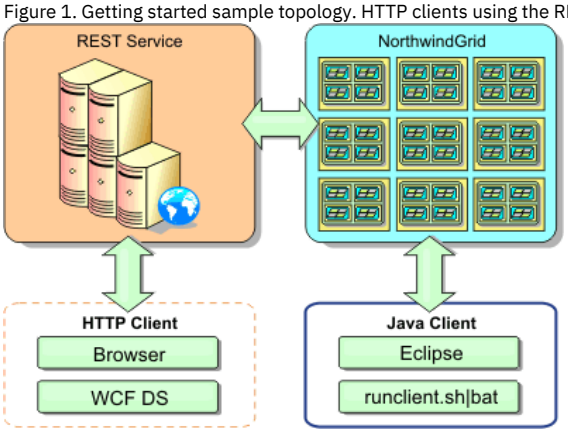
About this task

The included sample has source code and compiled binaries to run a partitioned data grid. This sample demonstrates how to create a simple data grid, model the data using entities and provides two command-line client applications that allow adding and querying entities using Java™ or C#.

The sample Java client uses the Java EntityManager API to persist and query data in the data grid. This client can be run in Eclipse or using a command-line script. Note that the sample Java client does not demonstrate the REST data service, but allows updating data in the grid, so a web browser or other clients can read the data.

The sample Microsoft WCF Data Services C# client communicates with the eXtreme Scale data grid through the REST data service using the .NET framework. The WCF Data Services client can be used to both update and query the data grid.

Figure 1. Getting started sample topology. HTTP clients using the REST data service and Java clients can access the same data grid.



Procedure

1. Configure and start the eXtreme Scale data grid. See Enabling the REST data service.
2. Configure and start the REST data service in a web server. See Configuring application servers for the REST data service.
3. Run a client to interact with the REST data service. Two options are available:
   a. Run the sample Java client to populate the grid with data using the EntityManager API and query the data in the grid using a web browser and the eXtreme Scale REST data service. See Using a Java client with REST data services.
   b. Run the sample WCF Data Services C# client. See Visual Studio 2008 WCF client with REST data service.

- Enabling the REST data service
  The REST data service can represent WebSphere eXtreme Scale entity metadata to represent each entity as an EntitySet.
- Configuring application servers for the REST data service
  You can configure various application servers to use the REST data service.
- Configuring Web browsers to access REST data service ATOM feeds
  The eXtreme Scale REST data service creates ATOM feeds by default when using a web browser. The ATOM feed format may not be compatible with older browsers or may be interpreted such that the data cannot be viewed as XML. You can configure Internet Explorer Version 8 and Firefox Version 3 to display the ATOM feeds and XML within the browser.
- Using a Java client with REST data services
  The Java client application uses the eXtreme Scale EntityManager API to insert data into the grid.
- Visual Studio 2008 WCF client with REST data service
  The eXtreme Scale REST data service getting started sample includes a WCF Data Services client that can interact with the eXtreme Scale REST data service. The sample is written as a command-line application in C#.

Enabling the REST data service

The REST data service can represent WebSphere® eXtreme Scale entity metadata to represent each entity as an EntitySet.

Starting a sample eXtreme Scale data grid

In general , before starting the REST data service, start the eXtreme Scale data grid. The following steps will start a single eXtreme Scale catalog service process and two container server processes.

WebSphere eXtreme Scale can be installed using three different methods:

- Trial install
- Stand-alone deployment
- WebSphere Application Server integrated deployment

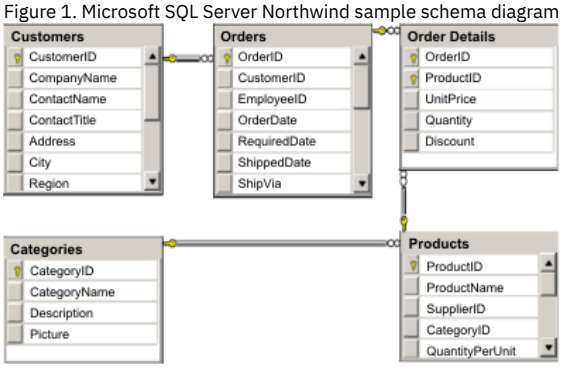
- Scalable data model in eXtreme Scale
  The Microsoft Northwind sample uses the Order Detail table to establish a many-to-many association between Orders and Products.
- Retrieving and updating data with REST
  The OData protocol requires that all entities can be addressed by their canonical form. This means that each entity must include the key of the partitioned, root entity, the schema root.
- Starting a stand-alone data grid for REST data services
  Follow these steps to start the WebSphere eXtreme Scale REST service sample data grid for a stand-alone eXtreme Scale deployment.
- Starting a data grid for REST data services in WebSphere Application Server
  Follow these steps to start a stand-alone WebSphere eXtreme Scale REST service sample data grid for a WebSphere eXtreme Scale deployment that is integrated with WebSphere Application Server. Although WebSphere eXtreme Scale is integrated with WebSphere Application Server, these steps start a stand-alone WebSphere eXtreme Scale catalog service process and container.

Scalable data model in eXtreme Scale

The Microsoft Northwind sample uses the Order Detail table to establish a many-to-many association between Orders and Products.

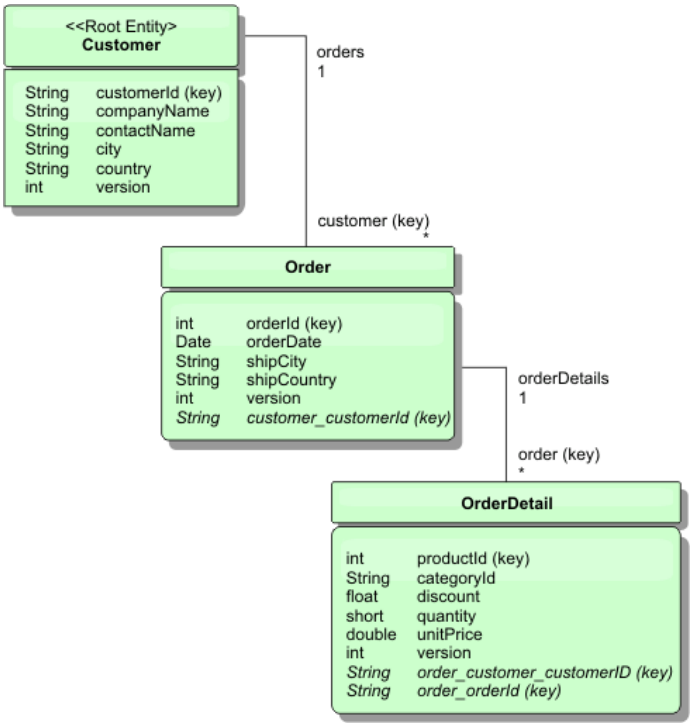
Object to relational mapping specifications (ORMs) such as the ADO.NET Entity Framework and Java™ Persistence API (JPA) can map the tables and relationships using entities. However, this architecture does not scale. Everything must be located on the same machine, or an expensive cluster of machines to perform well.

Figure 1. Microsoft SQL Server Northwind sample schema diagram



To create a scalable version of the sample, the entities must be modeled so each entity or group of related entities can be partitioned based off a single key. By creating partitions on a single key, requests can be spread out among multiple, independent servers. To achieve this configuration, the entities have been divided into two trees: the Customer and Order tree and the Product and Category tree. In this model, each tree can be partitioned independently and therefore can grow at different rates, increasing scalability.

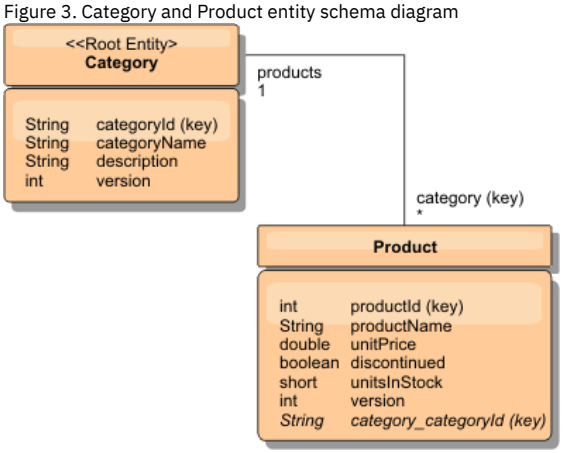
Figure 2. Customer and Order entity schema diagram



For example, both Order and Product have unique, separate integers as keys. In fact, the Order and Product tables are really independent of each other. For example, consider the effect of the size of a catalog, the number of products you sell, with the total number of orders. Intuitively, it might seem that having many products implies also having many orders, but this is not necessarily the case. If this were true, you could easily increase sales by just adding more products to your catalog. Orders and products have their own independent tables. You can further extend this concept so that orders and products each have their own separate, data grids. With independent data grids, you can control the number of partitions and servers, in addition to the size of each data grid separately so that your application can scale. If you double the size of your catalog, you must double the products data grid, but the order grid might be unchanged. The converse is true for an order surge, or expected order surge.

In the schema, a Customer has zero or more Orders, and an Order has line items (OrderDetail), each with one specific product. A Product is identified by ID (the Product key) in each OrderDetail. A single data grid stores Customers, Orders, and OrderDetails with Customer as the root entity of the data grid. You can retrieve Customers by ID, but you must get Orders starting with the Customer ID. So customer ID is added to Order as part of its key. Likewise, the customer ID and order ID are part of the OrderDetail ID.

Figure 3. Category and Product entity schema diagram



In the Category and Product schema, the Category is the schema root. With this schema, customers can query products by category. See Retrieving and updating data with REST for additional details on key associations and their importance.

Retrieving and updating data with REST

The OData protocol requires that all entities can be addressed by their canonical form. This means that each entity must include the key of the partitioned, root entity, the schema root.

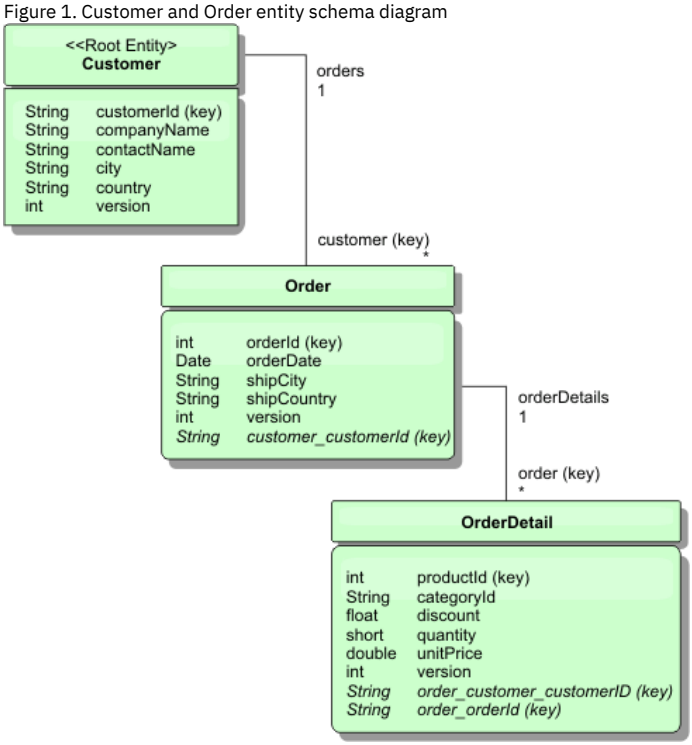
The following is an example of how to use the association from a root entity to address a child in :

```
/Customer('ACME')/order(100)
```

In WCF Data Services, the child entity must be directly addressable, meaning that the key in the schema root must be a part of the key of the child: `/Order(customer_customerId='ACME', orderId=100)`. This is achieved by creating an association to the root entity where the one-to-one or many-to-one

association to the root entity is also identified as a key. When entities are included as part of the key, the attributes of the parent entity are exposed as key properties.

Figure 1. Customer and Order entity schema diagram



The Customer/Order entity schema diagram illustrates how each entity is partitioned using the Customer. The Order entity includes the Customer as part of its key and is therefore directly accessible. The REST data service exposes all key associations as individual properties: Order has customer_customerId and OrderDetail has order_customer_customerId and order_orderId.

Using the EntityManager API, you can find the Order using the Customer and order id:

```
transaction.begin();
// Look-up the Order using the Customer.  We only include the Id
// in the Customer class when building the OrderId key instance.
Order order = (Order) em.find(Order.class,
    new OrderId(100, new Customer('ACME')));
...
transaction.commit();
```

When using the REST data service, the Order can be retrieved with either of the URLs:

- `/Order(orderId=100, customer_customerId='ACME')`
- `/Customer('ACME')/orders?$filter=orderId eq 100`

The customer key is addressed using the attribute name of the Customer entity, an underscore character and the attribute name of the customer id: customer_customerId.

An entity can also include a non-root entity as part of its key if all of the ancestors to the non-root entity have key associations to the root. In this example, OrderDetail has a key-association to Order and Order has a key-association to the root Customer entity. Using the EntityManager API:

```
transaction.begin();
// Construct an OrderDetailId key instance.  It includes
// The Order and Customer with only the keys set.
Customer customerACME = new Customer("ACME");
Order order100 = new Order(100, customerACME);
OrderDetailId orderDetailKey =
    new OrderDetailId(order100, "COMP");
OrderDetail orderDetail = (OrderDetail)
    em.find(OrderDetail.class, orderDetailKey);
...
```

The REST data service allows addressing the OrderDetail directly:

```
/OrderDetail(productId=500, order_customer_customerId='ACME', order_orderId =100)
```

The association from the OrderDetail entity to the Product entity has been broken to allow partitioning the Orders and Product inventory independently. The OrderDetail entity stores the category and product id instead of a hard relationship. By decoupling the two entity schemas, only one partition is accessed at a time.

The Category and Product schema illustrated in the diagram shows that the root entity is Category and each Product has an association to a Category entity. The Category entity is included in the Product identity. The REST data service exposes a key property: category_categoryId which allows directly addressing the Product.

Because Category is the root entity, in a partitioned environment, the Category must be known in order to find the Product. Using the EntityManager API, the transaction must be pinned to the Category entity prior to finding the Product.

Using the EntityManager API:

```
transaction.begin();
// Create the Category root entity with only the key.  This
// allows us to construct a ProductId without needing to find
// The Category first.  The transaction is now pinned to the
// partition where Category "COMP" is stored.
Category cat = new Category("COMP");
Product product = (Product) em.find(Product.class,
    new ProductId(500, cat));
...
```

The REST data service allows addressing the Product directly:

```
/Product(productId=500, category_categoryId='COMP')
```

Starting a stand-alone data grid for REST data services

Follow these steps to start the WebSphere® eXtreme Scale REST service sample data grid for a stand-alone eXtreme Scale deployment.

Before you begin

Install the WebSphere eXtreme Scale Trial or full product:

- Install the stand-alone version of the product and apply any subsequent fixes.
- Download and extract the WebSphere eXtreme Scale Version 7.1 or later trial, which includes the WebSphere eXtreme Scale REST data service.

About this task

Start the WebSphere eXtreme Scale sample data grid.

Procedure

1. Start the catalog service process. Open a command-line or terminal window and set the JAVA_HOME environment variable:
   - Linux UNIX `export JAVA_HOME=java_home`
   - Windows `set JAVA_HOME=java_home`
2. `cd restservice_home/gettingstarted`
3. Start the catalog service process. To start the service *without* eXtreme Scale security, use the following commands.
   - Linux UNIX `./runcat.sh`
   - Windows `runcat.bat`

   To start the service witheXtreme Scale security, use the following commands.
   - Linux UNIX `./runcat_secure.sh`
   - Windows `runcat_secure.bat`
4. Start two container server processes. Open another command-line or terminal window and set the JAVA_HOME environment variable:
   - Linux UNIX `export JAVA_HOME=java_home`
   - Windows `set JAVA_HOME=java_home`
5. `cd restservice_home/gettingstarted`
6. Start a container server process:
   To start the server without eXtreme Scale security, use the following commands:
   - Linux UNIX `./runcontainer.sh container0`
   - Windows `runcontainer.bat container0`

   To start the server witheXtreme Scale security, use the following commands.
   - Linux UNIX `./runcontainer_secure.sh container0`
   - Windows `runcontainer_secure.bat container0`
7. Open another command-line or terminal window and set the JAVA_HOME environment variable:
   - Linux UNIX `export JAVA_HOME=java_home`
   - Windows `set JAVA_HOME=java_home`
8. `cd restservice_home/gettingstarted`
9. Start a second container server process.
   To start the server without eXtreme Scale security, use the following commands.
   - Linux UNIX `./runcontainer.sh container1`
   - Windows `runcontainer.bat container1`

   To start the server with eXtreme Scale security, use the following commands.
   - Linux UNIX `./runcontainer_secure.sh container1`
   - Windows `runcontainer_secure.bat container1`

Results

Wait until the eXtreme Scale containers are ready before proceeding with the next steps. The container servers are ready when the following message is displayed in the terminal window:
`CWOBJ1001I: ObjectGrid Server container_name is ready to process requests.`

Where *container_name* is the name of the container that was started.

Starting a data grid for REST data services in WebSphere Application Server

Follow these steps to start a stand-alone WebSphere® eXtreme Scale REST service sample data grid for a WebSphere eXtreme Scale deployment that is integrated with WebSphere Application Server. Although WebSphere eXtreme Scale is integrated with WebSphere Application Server, these steps start a stand-alone WebSphere eXtreme Scale catalog service process and container.

Before you begin

Install the product into a WebSphere Application Server Version 7.0.0.5 or later installation directory with security disabled. Augment at least one application server profile.

About this task

Start the WebSphere eXtreme Scale sample data grid.

Procedure

1. Start the catalog service process. Open a command-line or terminal window and set the JAVA_HOME environment variable:
   - Linux UNIX `export JAVA_HOME=java_home`
   - Windows `set JAVA_HOME=java_home`

   `cd restservice_home/gettingstarted`
2. Start the catalog service process.
   To start the server without eXtreme Scale security, use the following commands.
   - Linux UNIX `./runcat.sh`
   - Windows `runcat.bat`

   To start the server witheXtreme Scale security, use the following commands.
   - Linux UNIX `./runcat_secure.sh`
   - Windows `runcat_secure.bat`
3. Start two container server processes. Open another command-line or terminal window and set the JAVA_HOME environment variable:
   - Linux UNIX `export JAVA_HOME=java_home`
   - Windows `set JAVA_HOME=java_home`
4. Start a container server process.
   To start the server without eXtreme Scale security, use the following commands.
   a. Open a command-line window.
   b. `cd restservice_home/gettingstarted`
   c. To start the server *without* eXtreme Scale security, use the following commands.
      - Linux UNIX `./runcontainer.sh container0`
      - Windows `runcontainer.bat container0`
   d. To start the server with eXtreme Scale security, use the following commands.
      - Linux UNIX `./runcontainer_secure.sh container0`
      - Windows `runcontainer_secure.bat container0`
5. Start a second container server process.
   a. Open a command-line window.
   b. `cd restservice_home/gettingstarted`
   c. To start the server *without* eXtreme Scale security, use the following commands.
      - Linux UNIX `./runcontainer.sh container1`
      - Windows `runcontainer.bat container1`
   d. To start the server *with* eXtreme Scale security, use the following commands.
      - Linux UNIX `./runcontainer_secure.sh container1`
      - Windows `runcontainer_secure.bat container1`

Results

Wait until the container servers are ready before proceeding with the next steps. The container servers are ready when the following message is displayed:
`CWOBJ1001I: ObjectGrid Server container_name is ready to process requests.`

Where *container_name* is the name of the container that was started in the previous step.

Configuring application servers for the REST data service

You can configure various application servers to use the REST data service.

- Deploying the REST data service on WebSphere Application Server
  This topic describes how to configure the WebSphere eXtreme Scale REST data service on WebSphere Application Server or WebSphere Application Server Network Deployment Version 6.1.0.25 or or later. These instructions also apply to deployments whereWebSphere eXtreme Scale is integrated with the WebSphere Application Server deployment.
- Deploying the REST data service on WebSphere Application Server Community Edition
  You can configure the eXtreme Scale REST data service on WebSphere Application Server Community Edition Version 2.1.1.3 or later.
- Deploying the REST data service on Apache Tomcat
  This topic describes how to configure theWebSphere eXtreme Scale REST data service on Apache Tomcat Version 5.5 or later.

Deploying the REST data service on WebSphere Application Server

This topic describes how to configure the WebSphere® eXtreme Scale REST data service on WebSphere Application Server or WebSphere Application Server Network Deployment Version 6.1.0.25 or or later. These instructions also apply to deployments whereWebSphere eXtreme Scale is integrated with the WebSphere Application Server deployment.

Before you begin

You must have one of the following environments on your system to configure and deploy the REST data service for WebSphere eXtreme Scale.

- WebSphere Application Server with the stand-alone WebSphere eXtreme Scale client:
  - The WebSphere eXtreme Scale Trial Version 7.1 with the REST data service is downloaded and extracted or theWebSphere eXtreme Scale Version 7.1.0.0 with cumulative fix 2 product is installed into a stand-alone directory.
  - WebSphere Application Server Version 6.1.0.25 or 7.0.0.5 or later is installed and running.
- WebSphere Application Server integrated with WebSphere eXtreme Scale:
  WebSphere eXtreme Scale Version 7.1.0.0 with cumulative fix 2 or later is installed on top of WebSphere Application Server Version 6.1.0.25 or 7.0 or later.

  Tip: The WebSphere eXtreme Scale REST data service only requires that the WebSphere eXtreme Scale client option be installed. The profile does not need to be augmented.
  Read about how to enable Java™ 2 security in the WebSphere Application Server information center.

Procedure

1. Configure and start a data grid.
   a. For details on configuring a data grid for use with the REST data service, see Starting a data grid for REST data services in WebSphere Application Server.
   b. Verify that a client can connect to and access entities in the data grid. For an example, see Tutorial: Getting started with WebSphere eXtreme Scale.
2. Build the eXtreme Scale REST service configuration JAR or directory. See the information about packaging and deploying the REST service in Installing the REST data service.
3. Add the REST data service configuration JAR or directory to the application server classpath:
   a. Open the WebSphere Application Server administrative console
   b. Navigate to Environment > Shared libraries
   c. Click **New**
   d. Add the following entries into the appropriate fields:
      - Name: extremescale_rest_configuration
      - Classpath: <REST service configuration jar or directory>
   e. Click **OK**
   f. Save the changes to the master configuration
4. Add the WebSphere eXtreme Scale client runtime JAR, wsogclient.jar, and the REST data service configuration JAR or directory to the application server classpath. This step is not necessary if WebSphere eXtreme Scale is integrated with the WebSphere Application Server installation.

   a. Open theWebSphere Application Server administrative console.
   b. Navigate to Environment > Shared libraries.
   c. Click **New**.
   d. Add the following entries into the fields:
      - Name: extremescale_client_v71
      - Classpath: wxs_home/lib/wsogclient.jar

      Remember: Add each path on a separate line.
   e. Click OK.
   f. Save the changes to the master configuration.
5. Install the REST data service EAR file, wxsrestservice.ear, to the WebSphere Application Server using the administrative console:
   a. Open the WebSphere Application Server administrative console.
   b. Click Applications > New application.
   c. Browse to the /lib/wxsrestservice.ear file on the file system and select it and click **Next**.
      - If using WebSphere Application Server Version 7.0, click Next.
      - If using WebSphere Application Server Version 6.1, enter a Context Root value with the name: /wxsrestservice and continue to the next step.
   d. Choose the detailed installation option, and click Next.
   e. On the application security warnings screen, click Continue.
   f. Choose the default installation options, and click Next.
   g. Choose a server to map the application to, and click Next.
   h. On the JSP reloading page, use the defaults, and click Next.
   i. On the shared libraries page, map the wxsrestservice.war module to the shared libraries that you defined:

- extremescale_rest_configuration
- extremescale_client_v71

Tip: This shared library is required only if WebSphere eXtreme Scale is not integrated with WebSphere Application Server.

j. On the map shared library relationship page, use the defaults, and click Next.
k. On the map virtual hosts page, use the defaults, and click Next.
l. On the map context roots page, set the context root to: wxsrestservice
m. On the Summary screen, click Finish to complete the installation.
n. Save the changes to the master configuration.

6. Start the `wxsrestservice` REST data service application:
   a. Go to the application in the administrative console.
      - WebSphere Application Server Version 7.0: In the administrative console, click Applications > Application Types > WebSphere Applications.
      - WebSphere Application Server Version 6.1: In the administrative console, click Applications > Enterprise Applications.
   b. Check the check box next to the `wxsrestservice` application, and click **Start**.
   c. Review the SystemOut.log file for the application server profile. When the REST data service has started successfully, the following message is displayed in the SystemOut.log file for the server profile:
   `CWOBJ4000I: The WebSphere eXtreme Scale REST data service has been started.`

7. Verify the REST data service is working: The port number can be found in the SystemOut.log file within the application server profile logs directory by looking at the first port displayed for message identifier: SRVE0250I. The default port is 9080.
   For example:`http://localhost:9080/wxsrestservice/restservice/NorthwindGrid/` Result: The AtomPub service document is displayed.

   For example: `http://localhost:9080/wxsrestservice/restservice/NorthwindGrid/$metadata`. The Entity Model Data Extensions (EDMX) document is displayed.

8. To stop the data grid processes, use `CTRL+C` in the respective command window.

- Starting REST data services with WebSphere eXtreme Scale integrated in WebSphere Application Server 7.0
  This topic describes how to configure and start the eXtreme Scale REST data service using WebSphere Application Server version 7.0 that has been integrated and augmented with WebSphere eXtreme Scale.

Starting REST data services with WebSphere eXtreme Scale integrated in WebSphere Application Server 7.0

This topic describes how to configure and start the eXtreme Scale REST data service using WebSphere® Application Server version 7.0 that has been integrated and augmented with WebSphere eXtreme Scale.

Before you begin

Verify that the sample stand-alone eXtreme Scale data grid is started. See Enabling the REST data service for details on how to start the data grid.

About this task

To get started with the WebSphere eXtreme Scale REST data service using WebSphere Application Server, follow these steps:

Procedure

1. Add the WebSphere eXtreme Scale REST data service sample configuration JAR to the classpath:
   a. Open the WebSphere Administration Console
   b. Navigate to Environment -> Shared libraries
   c. Click New
   d. Add the following entries into the appropriate fields:
      i. Name: extremescale_gettingstarted_config
      ii. Classpath
         - restservice_home/gettingstarted/restclient/bin
         - restservice_home/gettingstarted/common/bin

      Remember: Each path must appear on a different line.
   e. Click **OK**
   f. Save the changes to the master configuration
2. Install the REST data service EAR file, wxsrestservice.ear, to the WebSphere Application Server using the WebSphere administration console:
   a. Open the WebSphere administration console
   b. Navigate to Applications -> New Application
   c. Browse to restservice_home/lib/wxsrestservice.ear file on the file system. Select the file and click **Next**.
   d. Choose the detailed installation options, and click **Next**.
   e. On the application security warnings screen, click **Continue**.
   f. Choose the default installation options, and click **Next**.
   g. Choose a server to map the wxsrestservice.war module to, and click **Next**.
   h. On the JSP reloading page, use the defaults, and click **Next**.
   i. On the shared libraries page, map the "wxsrestservice.war" module to the following shared libraries that were defined during step 1: extremescale_ gettingstarted _config
   j. On the map shared library relationship page, use the defaults, and click **Next**.
   k. On the map virtual hosts page, use the defaults, and click **Next**.

l. On the map context roots page, set the context root to: wxsrestservice.
m. On the Summary screen, click **Finish** to complete the installation.
n. Save the changes to the master configuration.

3. If the eXtreme Scale data grid was started with eXtreme Scale security enabled, set the following property in the restservice_home/gettingstarted/restclient/bin/wxsRestService.properties file.

```
ogClientPropertyFile=restservice_home/gettingstarted/security/security.ogclient.properties
```

4. Start the application server and the "wxsrestservice " eXtreme Scale REST data service application.
After the application is started review the SystemOut.log for the application server and verify that the following message appears: `CWOBJ4000I: The WebSphere eXtreme Scale REST data service has been started.`

5. Verify that the REST data service is working:
   a. Open a browser and navigate to:
   http://localhost:9080/wxsrestservice/restservice/NorthwindGrid

   The service document for the NorthwindGrid is displayed.

   b. Navigate to:
   http://localhost:9080/wxsrestservice/restservice/NorthwindGrid/$metadata

   The Entity Model Data Extensions (EDMX) document is displayed

6. To stop the data grid processes, use CTRL+C in the respective command window to stop the process.

Deploying the REST data service on WebSphere Application Server Community Edition

You can configure the eXtreme Scale REST data service on WebSphere® Application Server Community Edition Version 2.1.1.3 or later.

Before you begin

- An IBM® (recommended) or Oracle JRE or JDK, Version 5 or later is installed and the JAVA_HOME environment variable is set.
- Download and install WebSphere Application Server Community Edition Version 2.1.1.3 or later to the wasce_root directory, for example the /opt/IBM/wasce directory. Read the installation instructions for information on version 2.1.1 or other versions.

Procedure

1. Configure and start a data grid.
   a. For details on configuring an eXtreme Scale data grid for use with the REST data service, read about Starting a stand-alone data grid for REST data services.
   b. Verify that an eXtreme Scale client can connect to and access entities in the data grid. For an example, see Tutorial: Getting started with WebSphere eXtreme Scale.
2. Build the eXtreme Scale REST service configuration JAR or directory. See the packaging and deployment information in the Installing the REST data service topic for details.
3. Start the WebSphere Application Server Community Edition server:
   a. To start the server without Java™ SE security enabled, run the following command:
   UNIX Linux `wasce_root/bin/startup.sh`

   Windows `wasce_root/bin/startup.bat`

   b. To start the server with Java SE security enabled, follow these steps: UNIX Linux
      i. Open a command-line or terminal window and run the following copy command (or copy the contents of the specified policy file into your existing policy): `cp restservice_home/gettingstarted/wasce/geronimo.policy wasce_root/bin`
      ii. Edit the `wasce_root/bin/setenv.sh` file
      iii. After the line that contains "`WASCE_JAVA_HOME=`", add the following: `export JAVA_OPTS="-Djava.security.manager -Djava.security.policy=geronimo.policy"`

   Windows
      i. Open a command-line window and run the following copy command or copy the contents of the specified policy file into your existing policy:
      `copy restservice_home\gettingstarted\wasce\geronimo.policy\bin`

      ii. Edit the `wasce_root\bin\setenv.bat` file
      iii. After the line that contains "set WASCE_JAVA_HOME=", add the following:
      `set JAVA_OPTS="-Djava.security.manager -Djava.security.policy=geronimo.policy"`

4. Add the ObjectGrid client runtime JAR to the WebSphere Application Server Community Edition repository:
   a. Open the WebSphere Application Server Community Edition administration console and log in. The default URL is: `http://localhost:8080/console` and the default userid is `system` and password is `manager`.
   b. Click the Repository link on the left side of the console window, in the Services folder.
   c. In the Add Archive to Repository section, fill in the following into the input text boxes:

Table 1. Add Archive to Repository

Text box	Value

Text box	Value
File	wxs_home/lib/ogclient.jar
Group	com.ibm.websphere.xs
Artifact	ogclient
Version	7.1
Type	JAR

d. Click the Install button

See the following tech note for details on different ways class and library dependencies can be configured: Specifying external dependencies to applications running on WebSphere Application Server Community Edition.

5. Deploy and start the REST data service module, the wxsrestservice.war file, to the WebSphere Application Server Community Edition server.
   a. Copy and edit the sample deployment plan XML file: restservice_home/gettingstarted/wasce/geronimo-web.xml to include path dependencies to your REST data service configuration JAR or directory. See section for an example on setting the classpath to include your wxsRestService.properties file and other configuration files and metadata classes.
   b. Open the WebSphere Application Server Community Edition administration console and log in.
   Tip: The default URL is: `http://localhost:8080/console`. The default userid is `system` and password is `manager`.
   c. Click on the Deploy Newlink on the left side of the console window.
   d. On the Install New Applications page, enter the following values into the text boxes:

Table 2. Install New Applications

Text box	Value
Archive	restservice_home/lib/wxsrestservice.war
Plan	restservice_home/gettingstarted/wasce/geronimo-web.xml

   Tip: Use the path to the geronimo-web.xml file that you copied and edited in step 3.
   e. Click on the Install button. The console page then indicates that the application was successfully installed and started.
   f. Examine the WebSphere Application Server Community Edition system output log or console to verify that the REST data service has started successfully. The following message must appear:
   `CWOBJ4000I: The WebSphere eXtreme Scale REST data service has been started.`

6. Start the WebSphere Application Server Community Edition server by running the following command:
   - UNIX Linux `wasce_root/bin/startup.sh`
   - Windows `wasce_root/bin/startup.bat`
7. Install the eXtreme Scale REST data service and the provided sample into the WebSphere Application Server Community Edition server:
   a. Add the ObjectGrid client runtime JAR to the WebSphere Application Server Community Edition repository:
      i. Open the WebSphere Application Server Community Edition administration console and log in. The default URL is: `http://localhost:8080/console`. The default userid is `system` and password is `manager`.
      ii. Click the Repository link on the left side of the console window, in the Services folder.
      iii. In the Add Archive to Repository section, fill in the following into the input text boxes:

Table 3. Add Archive to Repository

Text box	Value
File	wxs_home/lib/ogclient.jar
Group	com.ibm.websphere.xs
Artifact	ogclient
Version	7.1
Type	JAR

      iv. Click the install button.
      Tip: See the following technote for details on different ways class and library dependencies can be configured: Specifying external dependencies to applications running on WebSphere Application Server Community Edition
   b. Deploy the REST data service module: wxsrestservice.war to the WebSphere Application Server Community Edition server.
      i. Edit the sample restservice_home/gettingstarted/wasce/geronimo-web.xml deployment XML file to include path dependencies to the getting started sample classpath directories:
         - Change the "classesDirs" for the two getting started client GBeans:

      The "classesDirs" path for the GettingStarted_Client_SharedLib GBean should be set to: restservice_home/gettingstarted/restclient/bin
      The "classesDirs" path for the GettingStarted_Common_SharedLib GBean should be set to: restservice_home/gettingstarted/common/bin

      ii. Open the WebSphere Application Server Community Edition administration console and log in.
      iii. Click on the Deploy New link on the left side of the console window.
      iv. On the Install New Applications page, enter the following values into the text boxes:

Table 4. Install New Applications

Text box	Value
Archive	*restservice_home*/lib/wxsrestservice.war
Plan	*restservice_home*/gettingstarted/wasce/geronimo-web.xml

      v. Click the Install button.
      The console page then indicates that the application has successfully installed and started.

      vi. Examine the WebSphere Application Server Community Edition system output log to verify that the REST data service has started successfully by verifying that the following message is present:

```
CWOBJ4000I: The WebSphere eXtreme Scale REST data service has been started.
```

8. Verify that the REST data service is working:
   Open a Web browser and navigate to the following URL: `http://<host>:<port>/<context root>/restservice/<Grid Name>`

   The default port for WebSphere Application Server Community Edition is 8080 and is defined using the "HTTPPort" property in the /var/config/config-substitutions.properties file.

   For example: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/

Results

The AtomPub service document is displayed.

- Starting the REST data service in WebSphere Application Server Community Edition
  This topic describes how to configure and start the eXtreme Scale REST data service using WebSphere Application Server Community Edition.

Starting the REST data service in WebSphere Application Server Community Edition

This topic describes how to configure and start the eXtreme Scale REST data service using WebSphere® Application Server Community Edition.

Before you begin

Verify that the sample data grid is started. See Enabling the REST data service for details on how to start the grid.

Procedure

1. Download and install WebSphere Application Server Community Edition Version 2.1.1.3 or later to wasce_root, such as: /opt/IBM/wasce
2. Start the WebSphere Application Server Community Edition server by running the following command:
   - Linux UNIX wasce_root/bin/startup.sh
   - Windows wasce_root/bin/startup.bat
3. If the eXtreme Scale grid was started with eXtreme Scale security enabled, set the following properties in the restservice_home/gettingstarted/restclient/bin/wxsRestService.properties file.

   ```
   ogClientPropertyFile=restservice_home/gettingstarted/security/security.ogclient.properties
   loginType=none
   ```

4. Install the eXtreme Scale REST data service and the provided sample into the WebSphere Application Server Community Edition server:
   a. Add the ObjectGrid client runtime JAR to the WebSphere Application Server Community Edition repository:
      i. Open the WebSphere Application Server Community Edition administration console and log in.
         Tip: The default URL is: `http://localhost:8080/console`. The default user ID is `system` and password is `manager`.
      ii. Click the Repository, in the Services folder.
      iii. In the Add Archive to Repository section, fill in the following into the input text boxes:

      Table 1. Archive to repository

Text box	Value
File	wxs_home/lib/ogclient.jar
Group	com.ibm.websphere.xs
Artifact	ogclient
Version	7.0
Type	jar

      iv. Click the Install button.
         Tip: See the following tech note for details on different methods of configuration class and library dependencies: Specifying external dependencies to applications running on WebSphere Application Server Community Edition.
   b. Deploy the REST data service module, which is the wxsrestservice.war file, to the WebSphere Application Server Community Edition server.
      i. Edit the sample restservice_home/gettingstarted/wasce/geronimo-web.xml deployment XML file to include path dependencies to the getting started sample classpath directories:
         Change the classesDirs paths for the two getting started client GBeans:
         - The "classesDirs" path for the GettingStarted_Client_SharedLib GBean should be set to: restservice_home/gettingstarted/restclient/bin
         - The "classesDirs" path for the GettingStarted_Common_SharedLib GBean should be set to: restservice_home/gettingstarted/common/bin

      ii. Open the WebSphere Application Server Community Edition administrative console and log in.
         Tip: The default URL is: `http://localhost:8080/console`. The default user ID is `system` and password is `manager`.
      iii. Click Deploy New.
      iv. On the Install New Applications page, enter the following values into the text boxes:

      Table 2. Installation values

Text box	Value
Archive	restservice_home/lib/wxsrestservice.war
Plan	restservice_home/gettingstarted/wasce/geronimo-web.xml

v. Click on the Install button.
The console page should indicate that the application was successfully installed and started.

vi. Examine the WebSphere Application Server Community Edition system output log or console to verify that the REST data service has started successfully by verify that the following message is present:
`CWOBJ4000I: The WebSphere eXtreme Scale REST data service has been started.`

5. Verify that the REST data service is working:
   a. Open the following link in a browser window: `http://localhost:8080/wxsrestservice/restservice/NorthwindGrid`. The service document for the NorthwindGrid grid is displayed.
   b. Open the following link in a browser window: `http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/$metadata`. The Entity Model Data Extensions (EDMX) document is displayed.
6. To stop the grid processes, use `CTRL+C` in the respective command window to stop the process.
7. To stop WebSphere Application Server Community Edition, use the following command:
   - UNIX Linux wasce_root/bin/shutdown.sh
   - Windows wasce_root\bin\shutdown.bat

   Tip: The default user ID is `system` and password is `manager`. If you are using a custom port, use the `-port` option.

Deploying the REST data service on Apache Tomcat

This topic describes how to configure theWebSphere® eXtreme Scale REST data service on Apache Tomcat Version 5.5 or later.

About this task

- An IBM® or Oracle JRE or JDK, Version 5 or later installed and a specified JAVA_HOME environment variable.
- Apache Tomcat Version 5.5 or later is installed. See Apache Tomcat for details on how to install Tomcat.
- A stand-alone installation of WebSphere eXtreme Scale.

Procedure

1. If using an Oracle JRE or JDK, install the IBM ORB into Tomcat:
   a. Tomcat version 5.5:
      Copy all of the JAR files from:

      the *wxs_home*/lib/endorsed directory

      to:

      the *tomcat_root*/common/endorsed directory

   b. Tomcat version 6.0:
      Create an "endorsed" directory:

      UNIX Linux mkdir tomcat_root/endorsed

      Windows md tomcat_root/endorsed

      Copy all of the JAR files from:

      wxs_home/lib/endorsed

      to:

      tomcat_root/common/endorsed

2. Configure and start a data grid.
   a. For details on configuring a data grid for use with the REST data service, see Configuring.
   b. Verify that an eXtreme Scale client can connect to and access entities in the grid. For an example, see Configuring REST data services.
3. Build the eXtreme Scale REST service configuration JAR or directory. See the packaging and deployment information in Installing the REST data service for details.
4. Deploy the REST data service module: wxsrestservice.war to the Tomcat server.
   Copy the wxsrestservice.war file from:

   restservice_home/lib

   to:

   tomcat_root/webapps

5. Add the ObjectGrid client runtime JAR and the application JAR to the shared classpath in Tomcat:
   a. Edit the *tomcat_root*/conf/catalina.properties file

b. Append the following path names to the end of the shared.loader property, separating each path name with a comma:
- *wxs_home*/lib/ogclient.jar
- *restservice_home*/gettingstarted/restclient/bin
- *restservice_home*/gettingstarted/common/bin

6. If you are using Java™ 2 security, add security permissions to the tomcat policy file:
   - If using Tomcat version 5.5:
     Merge the contents of the sample 5.5 catalina policy file found in

     restservice_home/gettingstarted/tomcat/catalina-5_5.policy with the *tomcat_root*/conf/catalina.policy file.

   - If using Tomcat version 6.0:
     Merge the contents of the sample 6.0 catalina policy file found in

     restservice_home/gettingstarted/tomcat/catalina-6_0.policy with the *tomcat_root*/conf/catalina.policy file.

7. Start the Tomcat server:
   - **If using Tomcat 5.5 on UNIX or Windows, or the Tomcat 6.0 ZIP distribution:**
     - `cd tomcat_root/bin`
     - Start the server:
       - Without Java 2 security enabled:
         UNIX Linux `./catalina.sh run`

         Windows `catalina.bat run`

       - With Java 2 security enabled:
         UNIX Linux `./catalina.sh run -security`

         Windows `catalina.bat run -security`

     - The Apache Tomcat logs are displayed to the console. When the REST data service has started successfully, the following message is displayed in the administrative console:
       `CWOBJ4000I: The WebSphere eXtreme Scale REST data service has been started.`

   - **If using Tomcat 6.0 on Windows using the Windows installer distribution:**
     - `cd /bin`
     - Start the Apache Tomcat 6 configuration tool:
       `tomcat6w.exe`

     - To enable Java 2 security (optional):
       Add the following entries to the Java Options in the Java tab in the Apache Tomcat 6 properties window:

       ```
       -Djava.security.manager

       -Djava.security.policy=\conf\catalina.policy
       ```

     - Click on the Start button on the Apache Tomcat 6 properties window to start the Tomcat server.
     - Review the following logs to verify that the Tomcat server has started successfully:
       - *tomcat_root*/bin/catalina.log
         Displays the status of the Tomcat server engine

       - *tomcat_root*/bin/stdout.log
         Displays the system output log

     - When the REST data service has started successfully, the following message is displayed in the system output log:
       `CWOBJ4000I: The WebSphere eXtreme Scale REST data service has been started.`

8. Verify the REST data service is working.
   Open a Web browser and navigate to the following URL:

   `http://host:port/context_root/restservice/grid_name`

   The default port for Tomcat is 8080 and is configured in the *tomcat_root*/conf/server.xml file in the <Connector> element.

   For example:

   `http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/`

Results

The AtomPub service document is displayed.

- Starting REST data services in Apache Tomcat
  This topic describes how to configure and start the eXtreme Scale REST data service using Apache Tomcat, version 5.5 or later.

Starting REST data services in Apache Tomcat

This topic describes how to configure and start the eXtreme Scale REST data service using Apache Tomcat, version 5.5 or later.

Before you begin

Verify that the sample eXtreme Scale data grid is started. See Enabling the REST data service for details on how to start the data grid.

Procedure

1. Download and install Apache Tomcat Version 5.5 or later to tomcat_root. For example: /opt/tomcat
2. Install the eXtreme Scale REST data service and the provided sample into the Tomcat server as follows:
   a. If you are using an Oracle JRE or JDK, you must install the IBM® ORB into Tomcat:
      - For Tomcat version 5.5
        Copy all of the JAR files from:

        wxs_home/lib/endorsed

        to

        tomcat_root/common/endorsed

      - For Tomcat version 6.0
        - Create an "endorsed" directory
          - UNIX Linux `mkdir tomcat_root/endorsed`
          - Windows `md tomcat_root/endorsed`
        - Copy all of the JAR files from:
          wxs_home/lib/endorsed

          to

          tomcat_root/endorsed

   b. Deploy the REST data service module: wxsrestservice.war to the Tomcat server.
      Copy the wxsrestservice.war file from:

      restservice_home/lib

      to:

      tomcat_root/webapps

   c. Add the ObjectGrid client runtime JAR and the application JAR to the shared classpath in Tomcat:
      i. Edit the tomcat_root/conf/catalina.properties file
      ii. Append the following path names to the end of the shared.loader property in the form of a comma-delimited list:
         - wxs_home/lib/ogclient.jar
         - restservice_home/gettingstarted/restclient/bin
         - restservice_home/gettingstarted/common/bin

      Important: The path separator must be a **forward** slash.
3. If the eXtreme Scale data grid was started with eXtreme Scale security enabled, set the following properties in the restservice_home/gettingstarted/restclient/bin/wxsRestService.properties file.

   ```
   ogClientPropertyFile=restservice_home/gettingstarted/security/security.ogclient.properties
   loginType=none
   ```

4. Start the Tomcat server with the REST data service:
   - If using Tomcat 5.5 on UNIX or Windows, or Tomcat 6.0 on UNIX:
     - `cd tomcat_root/bin`
     - Start the server:
       - UNIX Linux `./catalina.sh run`
       - Windows `catalina.bat run`
     - The console then displays the Apache Tomcat logs. When the REST data service has started successfully, the following message is displayed in the administration console:
       `CWOBJ4000I: The WebSphere eXtreme Scale REST data service has been started.`
   - If using Tomcat 6.0 on Windows:
     - `cd tomcat_root/bin`
     - Start the Apache Tomcat 6 configuration tool with the following command: `tomcat6w.exe`
     - Click on the Start button on the Apache Tomcat 6 properties window to start the Tomcat server.
     - Review the following logs to verify that the Tomcat server has started successfully:
       - tomcat_root/bin/catalina.log
         Displays the status of the Tomcat server engine
       - tomcat_root/bin/stdout.log
         Displays the system output log.
     - When the REST data service has started successfully, the following message is displayed in the system output log: `CWOBJ4000I: The WebSphere eXtreme Scale REST data service has been started.`
5. Verify that the REST data service is working:
   a. Open a browser and navigate to:
      http://localhost:8080/wxsrestservice/restservice/NorthwindGrid

The service document for the NorthwindGrid is displayed.

b. Navigate to:
http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/$metadata

The Entity Model Data Extensions (EDMX) document is displayed.

6. To stop the data grid processes, use CTRL+C in the respective command window.
7. To stop Tomcat, use CTRL +C in the window in which you started it.

Configuring Web browsers to access REST data service ATOM feeds

The eXtreme Scale REST data service creates ATOM feeds by default when using a web browser. The ATOM feed format may not be compatible with older browsers or may be interpreted such that the data cannot be viewed as XML. You can configure Internet Explorer Version 8 and Firefox Version 3 to display the ATOM feeds and XML within the browser.

About this task

The eXtreme Scale REST data service creates ATOM feeds by default when using a web browser. The ATOM feed format may not be compatible with older browsers or may be interpreted such that the data cannot be viewed as XML. For older browsers, you will be prompted to save the files to disk. Once the files are downloaded, use your favorite XML reader to look at the files. The generated XML is not formatted to be displayed, so everything will be printed on one line. Most XML reading programs, such as Eclipse, support reformatting the XML into a readable format.
For modern browsers, such as Microsoft Internet Explorer Version 8 and Firefox Version 3, the ATOM XML files can be displayed natively in the browser. The following topics provide details on how to configure Internet Explorer Version 8 and Firefox Version 3 to display the ATOM feeds and XML within the browser.

Procedure

Configure Internet Explorer Version 8

- To enable Internet Explorer to read the ATOM feeds that the REST data service generates use the following steps:
  1. Click Tools > Internet Options
  2. Select the **Content** tab
  3. Click the **Settings** button in the **Feeds and Web Slices** section
  4. Uncheck the box: "Turn on feed reading view"
  5. Click **OK** to return to the browser.
  6. Restart Internet Explorer.

Configure Firefox Version 3

- Firefox does not automatically display pages with content type: application/atom+xml. The first time a page is displayed, Firefox prompts you to save the file. To display the page, open the file itself with Firefox as follows:
  1. From the application chooser dialog box, select the "Open with" radio button and click the **Browse** button.
  2. Navigate to your Firefox installation directory. For example: C:\Program Files\Mozilla Firefox
  3. Select `firefox.exe` and hit the **OK** button.
  4. Check the “Do this automatically for files like this...” check box.
  5. Click the **OK** button.
  6. Next, Firefox displays the ATOM XML page in a new browser window or tab
- Firefox automatically renders ATOM feeds in readable format. However, the feeds that the REST data service creates include XML. Firefox cannot display the XML unless you disable the feed renderer. Unlike Internet Explorer, in Firefox, the ATOM feed rendering plug-in must be explicitly edited. To configure Firefox to read ATOM feeds as XML files, follow these steps:
  1. Open the following file in a text editor: <firefoxInstallRoot>\components\FeedConverter.js. In the path, <firefoxInstallRoot> is the root directory where Firefox is installed.
     For Windows operating systems, the default directory is: C:\Program Files\Mozilla Firefox.
  2. Search for the snippet that looks as follows:

     ```
     // show the feed page if it wasn't sniffed and we have a document,
     // or we have a document, title, and link or id
     if (result.doc && (!this._sniffed ||
         (result.doc.title && (result.doc.link || result.doc.id)))) {
     ```

  3. Comment out the two lines that begin with `if` and `result` by placing `//` (two forward slashes) in front of them.
  4. Append the following statement to the snippet: `if(0) {`.
  5. The resulting text should look as follows:

     ```
     // show the feed page if it wasn't sniffed and we have a document,
     // or we have a document, title, and link or id
     //if (result.doc && (!this._sniffed ||
     //    (result.doc.title && (result.doc.link || result.doc.id)))) {
     if(0) {
     ```

  6. Save the file.
  7. Restart Firefox
  8. Now Firefox can automatically display all feeds in the browser.
- Test your setup by trying some URLs.

Example

This section describes some example URLs that can be used to view the data that was added by the getting started sample provided with the REST data service. Before using the following URLs, add the default data set to the eXtreme Scale sample data grid using either the sample Java™ client or the sample Visual Studio WCF Data Services client.

The following examples assume the port is 8080 which can vary. See section for details on how to configure the REST data service on different application servers.

- View a single customer with the id of "ACME":

  ```
  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('ACME')
  ```

- View all of the orders for customer "ACME":

  ```
  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('ACME')/orders
  ```

- View the customer "ACME" and the orders:

  ```
  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('ACME')?$expand=orders
  ```

- View order 1000 for customer "ACME":

  ```
  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=1000,customer_customerId='ACME')
  ```

- View order 1000 for customer "ACME" and its associated Customer:

  ```
  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
  Order(orderId=1000,customer_customerId='ACME')?$expand=customer
  ```

- View order 1000 for customer "ACME" and its associated Customer and OrderDetails:

  ```
  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
  Order(orderId=1000,customer_customerId='ACME')?$expand=customer,orderDetails
  ```

- View all orders for customer "ACME" for the month of October, 2009 (GMT):

  ```
  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
  Customer(customerId='ACME')/orders?$filter=orderDate
  ge datetime'2009-10-01T00:00:00'
  and orderDate lt datetime'2009-11-01T00:00:00'
  ```

- View all the first 3 orders and orderDetails for customer "ACME" for the month of October, 2009 (GMT):

  ```
  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
  Customer(customerId='ACME')/orders?$filter=orderDate
  ge datetime'2009-10-01T00:00:00'
  and orderDate lt datetime'2009-11-01T00:00:00'
  &$orderby=orderDate&$top=3&$expand=orderDetails
  ```

Using a Java client with REST data services

The Java™ client application uses the eXtreme Scale EntityManager API to insert data into the grid.

About this task

The previous sections described how to create an eXtreme Scale data grid and configure and start the eXtreme Scale REST data service. The Java client application uses the eXtreme Scale EntityManager API to insert data into the grid. It does not demonstrate how to use the REST interfaces. The purpose of this client is to demonstrate how the EntityManager API is used to interact with the eXtreme Scale data grid, and allow modifying data in the grid. To view data in the grid using the REST data service, use a web browser or use the Visual Studio 2008 client application.

Procedure

To quickly add content to the eXtreme Scale data grid, run the following command:

1. Open a command-line or terminal window and set the JAVA_HOME environment variable:
   - Linux UNIX `export JAVA_HOME=java_home`
   - Windows `set JAVA_HOME=java_home`
2. `cd restservice_home/gettingstarted`
3. Insert some data into the grid. The data that is inserted will be retrieved later using a Web browser and the REST data service.
   If the data grid was started *without*eXtreme Scale security, use the following commands.
   - UNIX Linux `./runclient.sh load default`
   - Windows `runclient.bat load default`

   If the data grid was started *with*eXtreme Scale security, use the following commands.
   - UNIX Linux `./runclient_secure.sh load default`
   - Windows `runclient_secure.bat load default`

   For a Java client, use the following command syntax:

- UNIX Linux `runclient.sh command`
- Windows `runclient.bat command`

The following commands are available:

- `load default`
  Loads a predefined set of Customer, Category and Product entities into the data grid and creates a random set of Orders for each customer.
- `load category categoryId categoryName firstProductId num_products`
  Creates a product Category and a fixed number of Product entities in the data grid. The firstProductId parameter identifies the id number of the the first product and each subsequent product is assigned the next id until the specified number of products is created.
- `load customer companyCode contactNamecompanyName numOrders firstOrderIdshipCity maxItems discountPct`
  Loads a new Customer into the data grid and creates a fixed set of Order entities for any random product currently loaded in the grid. The number of Orders is determined by setting the <numOrders> parameter. Each Order will have a random number of OrderDetail entities up to <maxItems>
- `display customer companyCode`
  Display a Customer entity and the associated Order and OrderDetail entities.
- `display category categoryId`
  Display a product Category entity and the associated Product entities.

Results

- `runclient.bat load default`
- `runclient.bat load customer IBM "John Doe" "IBM Corporation" 5 5000 Rochester 5 0.05`
- `runclient.bat load category 5 "Household Items" 100 5`
- `runclient.bat display customer IBM`
- `runclient.bat display category 5`

Running and building the sample data grid and Java client with Eclipse

The REST data service getting started sample can be updated and enhanced using Eclipse. For details on how to setup your Eclipse environment see the text document: restservice_home/gettingstarted/ECLIPSE_README.txt.

After the WXSRestGettingStarted project is imported into Eclipse and is building successfully, the sample will automatically re-compile and the script files used to start the container server and client will automatically pick up the class files and XML files. The REST data service will also automatically detect any changes since the Web server is configured to read the Eclipse build directories automatically.
Important: When changing source or configuration files, both the eXtreme Scale container server and the REST data service application must be restarted. The eXtreme Scale container server must be started before the REST data service Web application.

Visual Studio 2008 WCF client with REST data service

The eXtreme Scale REST data service getting started sample includes a WCF Data Services client that can interact with the eXtreme Scale REST data service. The sample is written as a command-line application in C#.

Software requirements

The WCF Data Services C# sample client requires the following:

- Operating system
  - Microsoft Windows XP
  - Microsoft Windows Server 2003
  - Microsoft Windows Server 2008
  - Microsoft Windows Vista
- Microsoft Visual Studio 2008 with Service Pack 1

  Tip: See the previous link for additional hardware and software requirements.
- Microsoft .NET Framework 3.5 Service Pack 1
- Microsoft Support: An update for the .NET Framework 3.5 Service Pack 1 is available

Building and running the getting started client

The WCF Data Services sample client includes a Visual Studio 2008 project and solution and the source code for running the sample. The sample must be loaded into Visual Studio 2008 and compiled into a Windows runnable program before it can be run. To build and run the sample, see the text document: restservice_home/gettingstarted/VS2008_README.txt.

WCF Data Services C# client command syntax

Windows `WXSRESTGettingStarted.exe <service URL> <command>`

The <service URL> is the URL of the eXtreme Scale REST data service configured in section .

The following commands are available:

- `load default`
  Loads a predefined set of Customer, Category and Product entities into the data grid and creates a random set of Orders for each customer.
- `load category <categoryId> <categoryName> <firstProductId> <numProducts>`
  Creates a product Category and a fixed number of Product entities in the data grid. The firstProductId parameter identifies the id number of the the first product and each subsequent product is assigned the next id until the specified number of products is created.
- `load customer <companyCode> <contactName> <companyName> <numOrders> <firstOrderId> <shipCity> <maxItems> <discountPct>`
  Loads a new Customer into the data grid and creates a fixed set of Order entities for any random product currently loaded in the data grid. The number of Orders is determined by setting the <numOrders> parameter. Each Order will have a random number of OrderDetail entities up to <maxItems>
- `display customer <companyCode>`
  Display a Customer entity and the associated Order and OrderDetail entities.
- `display category <categoryId>`
  Display a product Category entity and the associated Product entities.
- `unload`
  Remove all entities that were loaded using the "default load" command.

The following examples illustrate various commands.

- `WXSRestGettingStarted.exe http://localhost:8080/wxsrestservice/restservice/NorthwindGrid load default`
- `WXSRestGettingStarted.exe http://localhost:8080/wxsrestservice/restservice/NorthwindGrid load customer`
- `IBM "John Doe" "IBM Corporation" 5 5000 Rochester 5 0.05`
- `WXSRestGettingStarted.exe http://localhost:8080/wxsrestservice/restservice/NorthwindGrid load category 5 "Household Items" 100 5`
- `WXSRestGettingStarted.exe http://localhost:8080/wxsrestservice/restservice/NorthwindGrid display customer IBM`
- `WXSRestGettingStarted.exe http://localhost:8080/wxsrestservice/restservice/NorthwindGrid display category 5`

Configuring OSGi-enabled plug-ins using the ObjectGrid descriptor XML file

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="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsconfigosgiplugs_2_myTranCallb
ack" 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="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsconfigosgiplugs_2_myPluginCol
lectionRef1">
            <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsconfigosgiplugs_2_MapSerializ
erPlugin" osgiService="mySerializerFactory"/>
        </backingMapPluginCollection>
        <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsconfigosgiplugs_2_myPluginCol
lectionRef2">
            <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsconfigosgiplugs_2_MapSerializ
erPlugin" osgiService="myOtherSerializerFactory"/>
            <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsconfigosgiplugs_2_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.

Related concepts:
Samples
System APIs and plug-ins
Related tasks:
Configuring eXtreme Scale plug-ins with OSGi Blueprint
Building eXtreme Scale dynamic plug-ins
Scenario: Using an OSGi environment to develop and run eXtreme Scale plug-ins
Related information:
Building OSGi applications with the Blueprint Container specification
OSGi Bundle Activator API documentation
Spring namespace schema

Configuring servers for OSGi

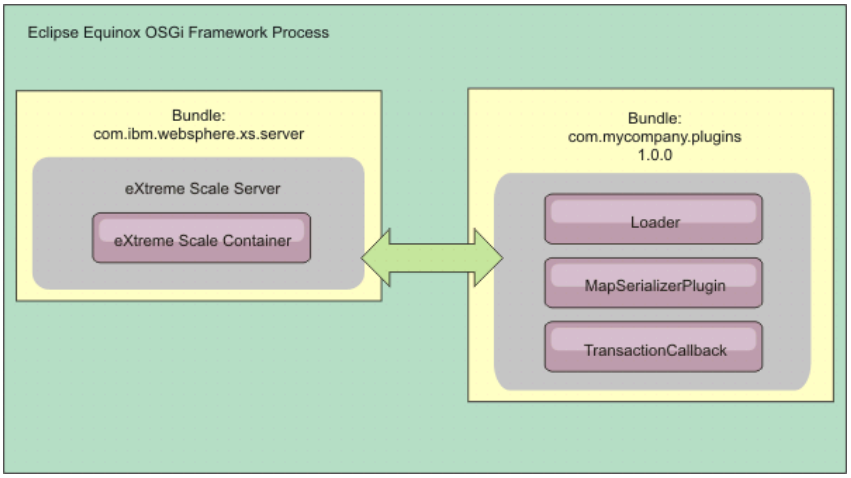
WebSphere® eXtreme Scale includes a server OSGi bundle, allowing starting and configuring servers and containers within an OSGi framework. The configuration topics describe how to use the eXtreme Scale server bundle, OSGi Blueprint service and eXtreme Scale configuration to run eXtreme Scale servers in an Eclipse Equinox OSGi framework.

About this task

the following tasks are required to start an eXtreme Scale server within Eclipse Equinox:

Procedure

1. Create an OSGi bundle that will store the eXtreme Scale plug-ins, exposing them as services and update the ObjectGrid descriptor XML file to reference the services.
2. Configure OSGi to start an eXtreme Scale container server.
3. Install and start the eXtreme Scale server bundle in the OSGi framework.
4. Install and start the OSGi bundle that contains the eXtreme Scale plug-ins.
   Figure 1. Eclipse Equinox process for installing and starting OSGi bundles with eXtreme Scale plug-ins



- Configuring eXtreme Scale plug-ins with OSGi Blueprint
  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.
- Configuring servers with OSGi Blueprint
  You can configure WebSphere eXtreme Scale container servers using an OSGi blueprint XML file, allowing simplified packaging and development of self-contained server bundles.
- Configuring servers with OSGI config admin
  You can use the OSGi configuration administration (config admin) service to configure WebSphere eXtreme Scale container servers.

Configuring eXtreme Scale plug-ins with OSGi Blueprint

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="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsblueprintosgi_2_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="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsblueprintosgi_2
_http://www.ibm.com/schema/objectgrid"

           xsi:schemaLocation="http://www.ibm.com/schema/objectgrid
                http://www.ibm.com/schema/objectgrid/objectgrid.xsd">

    <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsblueprintosgi_2_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="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsblueprintosgi_2_myPluginBeanF
actory"
    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.

Related concepts:
Samples
System APIs and plug-ins
Related tasks:
Configuring OSGi-enabled plug-ins using the ObjectGrid descriptor XML file
Building eXtreme Scale dynamic plug-ins
Scenario: Using an OSGi environment to develop and run eXtreme Scale plug-ins
Related information:
Building OSGi applications with the Blueprint Container specification
OSGi Bundle Activator API documentation
Spring namespace schema

Configuring servers with OSGi Blueprint

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="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsconfigblue_2_ht
tp://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="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsconfigblue_2_xsServer"
tracespec="ObjectGridOSGi=all=enabled"
tracefile="logs/osgi/wxsserver/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="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsconfigblue_2_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.

Configuring servers with OSGI config admin

You can use the OSGi configuration administration (config admin) service to configure WebSphere® eXtreme Scale container servers.

About this task

To configure a server, the ManagedService persistent identifier (PID), com.ibm.websphere.xs.server, is set to reference the ObjectGrid server properties file on the file system. To configure a container, the ManagedServiceFactory PID, com.ibm.websphere.xs.container, is set to reference the ObjectGrid deployment XML

file and ObjectGrid deployment policy XML file on the file system.
When the two PIDs are set in the config admin service, the eXtreme Scale server service automatically initializes the server and start the container with the specified configuration files. Config admin PIDs are persisted to the OSGi configuration directory. If the configuration is not cleared, the settings are retained between framework restarts.

Several third-party utilities exist for setting config admin properties. The following utilities are examples of tools that the product supports:

- The Luminis OSGi Configuration Admin command line client allows command line configuration.
- Apache Felix File Install allows specifying config admin PID settings in standard property files.

To configure eXtreme Scale container servers with the OSGi Configuration Administration command-line client for Luminis, complete the following steps

Procedure

1. Create a managed service PID for the ObjectGrid 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
```

2. Create a managed service factory persistence identifier PID for the ObjectGrid container in the OSGi console by running the following commands. Attention: Use the PID that is created with the **createf** config admin command. The PID that is used in the following code snippet is only an example.

```
osgi> cm createf com.ibm.websphere.xs.container
PID: com.ibm.websphere.xs.container-123456789-0
osgi> cm put com.ibm.websphere.xs.container-123456789-0 objectgridFile /mypath/objectGrid.xml
osgi> cm put com.ibm.websphere.xs.container-123456789-0 deploymentPolicyFile /mypath/deployment.xml
```

Results

eXtreme Scale container servers are now configured to start in an Eclipse Equinox OSGi framework.

What to do next

Container servers can also be programmatically created using the ServerFactory API and OSGi bundle activators. For details on using the ServerFactory API, see the API documentation.

Configuring eXtreme Scale servers to run in the Liberty profile

8.5+ You must create server and catalog service definitions to run WebSphere® eXtreme Scale in the Liberty profile.

Before you begin

Install the WebSphere Application Server Liberty profile and WebSphere eXtreme Scale. For more information, see Installing the Liberty profile.

Procedure

1. Create a server definition; for example:

```
<wlp install_root>/bin/server create <your server name>
```

2. Find the server.xml file under your server definition and open it in an editor. A commented feature manager stanza already exists. You can find the server definition in the following directory:

```
<wlp install_root>/usr/servers/<your server name>
```

3. Create a catalog service definition. To define a catalog service in a server, you need the WebSphere eXtreme Scale server feature and the server configuration file.
   a. Add the server feature to the server definition: 8.5

```
<featureManager>
     <feature>eXtremeScale_server-1.0</feature>
</featureManager>
```

   b. Add the configuration to tell the server feature that this server is a catalog service; for example:

```
<com.ibm.ws.xs.server.config isCatalog="true"/>
```

What to do next

After you configure eXtreme Scale servers, you can start yours servers in the Liberty profile. For more information, see Starting and stopping servers in the Liberty profile.

Scenario: Configuring HTTP session failover in the Liberty profile

8.5+ 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 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.

- 8.5 Enabling the eXtreme Scale web feature in the Liberty profile
  You can enable the web feature to use HTTP session failover in the Liberty profile.
- Configuring a web server plug-in to forward requests to multiple servers in the Liberty profile
  Use this task to configure the web server plug-in to distribute HTTP server requests between multiple servers in the Liberty profile.
- Merging plug-in configuration files for deployment to the application server plug-in
  Generate plug-in configuration files after you configure a unique clone ID in the Liberty server.xml configuration file.
- 8.5+ Enabling the eXtreme Scale web feature in the Liberty profile
  You can enable the web feature to use HTTP session failover in the Liberty profile.
- 8.5+ Configuring a web server plug-in to forward requests to multiple servers in the Liberty profile
  Use this task to configure the web server plug-in to distribute HTTP server requests between multiple servers in the Liberty profile.
- 8.5+ Merging plug-in configuration files for deployment to the application server plug-in
  Generate plug-in configuration files after you configure a unique clone ID in the Liberty server.xml configuration file.

Related tasks:
Scenario: Using an OSGi environment to develop and run eXtreme Scale plug-ins
8.5+ Scenario: Using JCA to connect transactional applications to eXtreme Scale clients
8.5+ Scenario: Running grid servers in the Liberty profile using Eclipse tools
Migrating a WebSphere Application Server memory-to-memory replication or database session to use WebSphere eXtreme Scale session management
8.5+

Enabling the eXtreme Scale web feature in the Liberty profile

8.5+ 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

8.5 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.
Related reference:
Liberty profile web feature properties
Liberty profile xsWebGrid feature properties
Liberty profile webApp feature properties

Configuring a web server plug-in to forward requests to multiple servers in the Liberty profile

8.5+ 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:

- 8.5 Enabling the eXtreme Scale web feature in the Liberty profile

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

8.5+ 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
- Configuring a web server plug-in to forward requests to multiple servers in the Liberty profile

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.

Administering

Administering and operating the product environment includes starting and stopping servers, managing the availability of the data grid, and recovering from data center failure scenarios. After you configure your catalog servers and container servers, you can start and stop the servers using various methods. The method that you use to start and stop servers depends on if you are using an embedded topology, a stand-alone topology, or a topology that is running within WebSphere® Application Server.

- Starting and stopping stand-alone servers
  You can start and stop stand-alone catalog and container servers with scripts or the embedded server API.
- Stopping servers gracefully with the xscmd utility
  You can use the **xscmd** utility with the -c teardown command to stop a list or group of catalog and container servers. This command simplifies shutting down all or portions of a data grid. It also prevents unnecessary placement and recovery catalog service actions that normally occur when processes are stopped ungracefully.
- Starting and stopping servers in a WebSphere Application Server environment
  Catalog and container servers can automatically start in a WebSphere Application Server or WebSphere Application Server Network Deployment environment.
- Starting and stopping servers in the Liberty profile
  Use the Liberty profile **server** command to start and stop WebSphere eXtreme Scale servers in the Liberty profile.
- Using the embedded server API to start and stop servers
  With WebSphere eXtreme Scale, you can use a programmatic API for managing the life cycle of embedded servers and containers. You can programmatically configure the server with any of the options that you can also configure with the command line options or file-based server properties. You can configure the embedded server to be a container server, a catalog service, or both.
- Administering with the xscmd utility
  With the **xscmd** utility, you can complete administrative tasks in the environment such as: establishing multi-master replication links, overriding quorum, and stopping groups of servers with the teardown command.
- Controlling placement
  You can use several different options to control when shards are placed on various container servers in the configuration. During startup, you might choose to delay the placement of shards. When you are running all of your container servers, you might need to suspend, resume, or change placement while you maintain servers.
- Managing ObjectGrid availability
  The availability state of an ObjectGrid instance determines which requests can be processed at any particular time. You can use the StateManager interface to set and retrieve the state of an ObjectGrid instance.
- Managing data center failures
  How you manage failures in your data centers depends on if you have enabled quorum in the catalog servers. If quorum is enabled, all placement activities are stopped when a failure is detected. You must override quorum and take further administrative actions.
- 8.5+ Querying and invalidating data
  You can use the query interfaces in the monitoring console and in the **xscmd** utility to retrieve small sets of keys from a map and invalidate sets of data.
- 8.5+ Retrieving eXtreme Scale environment information with the xscmd utility
  You can use the **xscmd** utility with the -c showinfo command to view important details about the servers running in your WebSphere eXtreme Scale environment, including: WebSphere eXtreme Scale servers, Java™ virtual machines, and (if applicable) servers running with WebSphere Application Server. Issue this command to retrieve name and version information, hostname and IP address, and the installation directories of these servers. Using -c showinfo command lets you retrieve these details without having to check log files, directories, or use third party applications.
- 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.
- 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.
- 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.
- Administering with Managed Beans (MBeans)
  You can use several different types of Java Management Extensions (JMX) MBeans to administer and monitor deployments. Each MBean refers to a specific entity, such as a map, data grid, server, or service.
- 8.5+ Administering J2C client connections
  The WebSphere eXtreme Scale connection factory includes an eXtreme Scale client connection that can be shared between applications and persisted through application restarts.

Related tasks:
Troubleshooting administration
Administering with the xscmd utility
Starting and stopping stand-alone servers
Starting a stand-alone catalog service
Configuring WebSphere eXtreme Scale with WebSphere Application Server

Starting and stopping stand-alone servers

You can start and stop stand-alone catalog and container servers with scripts or the embedded server API.

Before you begin

If you are starting or stopping servers in a stand-alone environment that is using an external client security provider, you must set the *CLIENT_AUTH_LIB* environment variable before you run the start and stop scripts. For more information about setting this environment variable, see Starting secure servers in a stand-alone environment.

- Starting stand-alone servers
  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.
- Stopping stand-alone servers
  You can use the stopOgServer script to stop eXtreme Scale server processes.

Related concepts:
Configuration considerations for multi-master topologies
8.5+ Installing fix packs using IBM Installation Manager
Installation topologies
Catalog service
Container servers, partitions, and shards
Example: Configuring catalog service domains
Administering
Related tasks:
Retrieving eXtreme Scale environment information with the xscmd utility
Updating eXtreme Scale servers
Configuring multiple data center topologies
Migrating to WebSphere eXtreme ScaleVersion 8.5
Starting and stopping servers in a WebSphere Application Server environment
Configuring catalog and container servers
Using the embedded server API to start and stop servers
Related reference:
startOgServer script
Server properties file
ObjectGrid descriptor XML file
Related information:
Getting started tutorial lesson 3.1: Starting catalog and container servers

Starting stand-alone servers

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.

- Starting a stand-alone catalog service
  You must start the catalog service manually when you are using a distributed WebSphere eXtreme Scale environment that is not running in WebSphere Application Server.
- Starting container servers
  You can start container servers from the command line using a deployment topology or using a server.properties file.
- startOgServer script
  The **startOgServer** script starts container and catalog servers . You can use a variety of parameters when you start your servers to enable trace, specify port numbers, and so on.

Related tasks:
Stopping stand-alone servers
Controlling placement
Administering with the xscmd utility
Enabling logging
Collecting trace
Related reference:
Deployment policy descriptor XML file
Server trace options
Messages
Related information:
Interface PlacementServiceMBean

Starting a stand-alone catalog service

You must start the catalog service manually when you are using a distributed WebSphere® eXtreme Scale environment that is not running in WebSphere Application Server.

Before you begin

- If you are using WebSphere Application Server, the catalog service automatically starts within the existing processes. For more information, see Starting and stopping servers in a WebSphere Application Server environment.

About this task

Start the catalog service with the **startOgServer** script. When you call the start command, use the **startOgServer.sh** script on UNIX platforms or **startOgServer.bat** on Windows.
The catalog service can run in a single process or can include multiple catalog servers to form a catalog service domain. A catalog service domain is required in a production environment for high availability. For more information, see High availability catalog service. You can also specify additional parameters to the script to bind the Object Request Broker (ORB) to a specific host and port, specify the domain, or enable security.

Procedure

- **Start a single catalog server process.**
  To start a single catalog server, type the following commands from the command line:

  1. Navigate to the bin directory.

     ```
     cd objectgridRoot/bin
     ```

  2. Run the **startOgServer** command.

     ```
     startOgServer.bat|sh catalogServer
     ```

  For a list of all of the available command-line parameters, see startOgServer script. Do not use a single Java™ virtual machine (JVM) to run the catalog service in a production environment. If the catalog service fails, no new clients are able to route to the deployed eXtreme Scale, and no new ObjectGrid instances can be added to the domain. For these reasons, you should start a set of Java virtual machines to run a catalog service domain.
- **Start a catalog service domain that consists of multiple endpoints.**
  To start a set of servers to run a catalog service, you must use the -catalogServiceEndPoints option on the startOgServer script. This argument accepts a list of catalog service endpoints in the format of *serverName*:*hostName*:*clientPort*:*peerPort*.

  The following example shows how to start the first of three Java virtual machines to host a catalog service:
  1. Navigate to the bin directory.

     ```
     cd wxs_install_root/bin
     ```

  2. Run the **startOgServer** command.

     ```
     startOgServer.bat|sh cs1 -catalogServiceEndPoints
     cs1:MyServer1.company.com:6601:6602,cs2:MyServer2.company.com:6601:6602,cs3:MyServer3.company.com:6601:6602
     ```

     In this example, the `cs1` server on the `MyServer1.company.com` host is started. This server name is the first argument that is passed to the script. During initialization of the `cs1` server, the -catalogServiceEndpoints parameters are examined to determine which ports are allocated for this process. The list is also used to allow the `cs1` server to accept connections from other servers: `cs2` and `cs3`.
  3. To start the remaining catalog servers in the list, pass the following arguments to the **startOgServer** script. Starting the `cs2` server on the `MyServer2.company.com` host.

     ```
     startOgServer.bat|sh cs2 -catalogServiceEndPoints
     cs1:MyServer1.company.com:6601:6602,cs2:MyServer2.company.com:6601:6602,cs3:MyServer3.company.com:6601:6602
     ```

     Starting `cs3` on `MyServer3.company.com`:

     ```
     startOgServer.bat|sh cs3 -catalogServiceEndPoints
     cs3:MyServer3.company.com:6601:6602,cs1:MyServer1.company.com:6601:6602,cs2:MyServer2.company.com:6601:6602
     ```

     The order of the list for the -catalogServiceEndpoints parameter can be different for the various catalog servers, but the servers contained in the list must be the same. Do not put any spaces in the list.
     Important: **Start at least two catalog servers in parallel.**
     You must start catalog servers that are in a data grid in parallel, because each server pauses to wait for the other catalog servers to join the core group. A catalog server that is configured for a data grid does not start until it identifies other members in the group. The catalog server eventually times out if no other servers become available.

- **Bind the ORB to a specific host and port.**
  Aside from ports defined in the catalogServiceEndpoints argument, each catalog service also uses an Object Request Broker (ORB) to accept connections from clients and containers. By default, the ORB listens on port `2809` of the localhost. If you want to bind the ORB to a specific host and port on a catalog service JVM, use the -listenerHost and -listenerPort arguments. The following example shows how to start a single JVM catalog server with its ORB bound to port `7000` on `MyServer1.company.com`:

  ```
  startOgServer.sh catalogServer -listenerHost MyServer1.company.com
  -listenerPort 7000
  ```

  Each eXtreme Scale container and client must be provided with catalog service ORB endpoint data. Clients only need a subset of this data, but you should use at least two endpoints for high availability.
- Optional: **Name the catalog service domain**

A catalog service domain name is not required when starting a catalog service. However, if you are using multi-master replication or are using multiple catalog service domains within the same set of processes, then you need to define a unique catalog service domain name. The default domain name is `DefaultDomain`. To give your domain a name, use the -domain option. The following example demonstrates how to start a single catalog service JVM with the domain name `myDomain`.

```
startOgServer.sh catalogServer -domain myDomain
```

For more information about configuring multi-master replication, see Configuring multiple data center topologies.

- **Start a secure catalog service.** For more information, see Starting secure servers in a stand-alone environment.
- **Start the catalog service programmatically.**
  Any JVM setting that is flagged by the CatalogServerProperties.setCatalogServer method can host the catalog service for eXtreme Scale. This method indicates to the eXtreme Scale server run time to instantiate the catalog service when the server is started. The following code shows how to instantiate the eXtreme Scale catalog server:

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

//The getInstance() method will start the catalog service.
Server server = ServerFactory.getInstance();
```

For more information about starting servers programmatically, see Using the embedded server API to start and stop servers.

Related concepts:
Example: Configuring catalog service domains
Administering
Related tasks:
Starting container servers
Related reference:
startOgServer script

Starting container servers

You can start container servers from the command line using a deployment topology or using a server.properties file.

About this task

To start a container process, you need an ObjectGrid XML file. The ObjectGrid XML file specifies which eXtreme Scale servers the container hosts. Ensure that your container is equipped to host each ObjectGrid in the XML that you pass to it. All of the classes that these ObjectGrids require must be in the classpath for the container. For more information about the ObjectGrid XML file, see objectGrid.xsd file.

Procedure

- **Start the container server from the command line.**
  1. From the command line, navigate to the bin directory:

     ```
     cd wxs_install_root/bin
     ```

  2. Run the following command:

     ```
     startOgServer.sh c0 -objectGridFile ../xml/companyGrid.xml
     -catalogServiceEndPoints MyServer1.company.com:2809
     ```

  Important: On the container server, the -catalogServiceEndPoints option is used to reference the Object Request Broker (ORB) host and port on the catalog service. The catalog service uses the -listenerHost and -listenerPort options to specify the host and port for ORB binding or accepts the default binding. When you are starting a container, use the -catalogServiceEndPoints option to reference the values that are passed to the -listenerHost and -listenerPort options on the catalog service. If -listenerHost and -listenerPort options are not used when the catalog service is started, the ORB binds to port 2809 on the localhost for the catalog service. Do not use the -catalogServiceEndPoints option to reference the hosts and ports that were passed to the -catalogServiceEndPoints option on the catalog service. On the catalog service, the -catalogServiceEndPoints option is used to specify ports necessary for static server configuration.
  This process is identified by c0, the first argument passed to the script. Use the companyGrid.xml to start the container. If your catalog server ORB is running on a different host than your container or it is using a non-default port, you must use the -catalogServiceEndPoints argument to connect to the ORB. For this example, assume that a single catalog service is running on port `2809` on `MyServer1.company.com`
- **Start the container using a deployment policy.**
  Although not required, a deployment policy is recommended during container start up. The deployment policy is used to set up partitioning and replication for eXtreme Scale. The deployment policy can also be used to influence placement behavior. Because the previous example did not provide a deployment policy file, the example receives all default values with regard to replication, partitioning, and placement. So, the maps in the CompanyGrid are in one mapSet. The mapSet is not partitioned or replicated. For more information about deployment policy files, see Deployment policy descriptor XML file. The following example uses the companyGridDpReplication.xml file to start a container JVM, the c0 JVM:

  1. From the command line, navigate to the bin directory:

     ```
     cd wxs_install_root/bin
     ```

  2. Run the following command:

```
startOgServer.sh c0 -objectGridFile ../xml/companyGrid.xml
-deploymentPolicyFile ../xml/companyGridDpReplication.xml
-catalogServiceEndPoints MyServer1.company.com:2809
```

Note: If you have Java™ classes stored in a specific directory, or you are using a loader or agent, instead of altering the StartOgServer script, you can launch the server with arguments as follows: `-jvmArgs -cp C:\ . . . \DirectoryPOJOs\POJOs.jar` . In the companyGridDpReplication.xml file, a single map set contains all of the maps. This mapSet is divided into 10 partitions. Each partition has one synchronous replica and no asynchronous replicas. Any container that uses the companyGridDpReplication.xml deployment policy paired with the companyGrid.xml ObjectGrid XML file is also able to host CompanyGrid shards. Start another container JVM, the c1 JVM:

1. From the command line, navigate to the bin directory:

   ```
   cd wxs_install_root/bin
   ```

2. Run the following command:

   ```
   startOgServer.sh c1 -objectGridFile ../xml/companyGrid.xml
   -deploymentPolicyFile ../xml/companyGridDpReplication.xml
   -catalogServiceEndPoints MyServer1.company.com:2809
   ```

Each deployment policy contains one or more objectgridDeployment elements. When a container is started, it publishes its deployment policy to the catalog service. The catalog service examines each objectgridDeployment element. If the objectgridName attribute matches the objectgridName attribute of a previously received objectgridDeployment element, the latest objectgridDeployment element is ignored. The first objectgridDeployment element received for a specific objectgridName attribute is used as the master. For example, assume that the c2 JVM uses a deployment policy that divides the mapSet into a different number of partitions:

companyGridDpReplicationModified.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy
                ../deploymentPolicy.xsd"
    xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">

    <objectgridDeployment objectgridName="CompanyGrid">
        <mapSet name="mapSet1" numberOfPartitions="5"
            minSyncReplicas="1" maxSyncReplicas="1"
            maxAsyncReplicas="0">
            <map ref="Customer" />
            <map ref="Item" />
            <map ref="OrderLine" />
            <map ref="Order" />
        </mapSet>
    </objectgridDeployment>

</deploymentPolicy>
```

Now, you can start a third JVM, the c2 JVM:

1. From the command line, navigate to the bin directory:

   ```
   cd wxs_install_root/bin
   ```

2. Run the following command:

   ```
   startOgServer.sh c2 -objectGridFile ../xml/companyGrid.xml
   -deploymentPolicyFile ../xml/companyGridDpReplicationModified.xml
   -catalogServiceEndPoints MyServer1.company.com:2809
   ```

The container on the c2 JVM is started with a deployment policy that specifies 5 partitions for mapSet1. However, the catalog service already holds the master copy of the objectgridDeployment for the CompanyGrid. When the c0 JVM was started it specified that 10 partitions exist for this mapSet. Because it was the first container to start and publish its deployment policy, its deployment policy became the master. Therefore, any objectgridDeployment attribute value that is equal to `CompanyGrid` in a subsequent deployment policy is ignored.

- **Start a container using a server properties file.**
  You can use a server properties file to set up trace and configure security on a container. Run the following commands to start container c3 with a server properties file:

  1. From the command line, navigate to the bin directory:

     ```
     cd wxs_install_root/bin
     ```

  2. Run the following command:

     ```
     startOgServer.sh c3 -objectGridFile ../xml/companyGrid.xml
     -deploymentPolicyFile ../xml/companyGridDpReplicationModified.xml
     -catalogServiceEndPoints MyServer1.company.com:2809
     -serverProps ../serverProps/server.properties
     ```

  An example server.properties file follows:

  ```
  server.properties
  workingDirectory=
  traceSpec=*=all=disabled
  systemStreamToFileEnabled=true
  enableMBeans=true
  memoryThresholdPercentage=50
  ```

  This is a basic server properties file that does not have security enabled. For more information about the server.properties file, see Server properties file.

- **Start a container server programmatically.**
  For more information about starting container servers programmatically, see Using the embedded server API to start and stop servers.

Related tasks:
Starting a stand-alone catalog service
Related reference:
startOgServer script
Deployment policy descriptor XML file
Configuring distributed deployments

startOgServer script

The **startOgServer** script starts container and catalog servers . You can use a variety of parameters when you start your servers to enable trace, specify port numbers, and so on.

Purpose

You can use the **startOgServer** script to start servers.

Location

The **startOgServer** script is in the bin directory of the root directory, for example:

```
cd wxs_install_root/bin
```

Note: If you have Java™ classes stored in a specific directory, or you are using a loader or agent, instead of altering the **startOgServer** script, you can launch the server with arguments as follows: `-jvmArgs -cp C:\ . . . \DirectoryPOJOs\POJOs.jar`
.

Usage for catalog servers

To start a catalog server:

Windows

```
startOgServer.bat <server> [options]
```

UNIX

```
startOgServer.sh <server>[options]
```

To start a default configured catalog server, use the following commands:
Windows

```
startOgServer.bat catalogServer
```

UNIX

```
startOgServer.sh catalogServer
```

Options for starting catalog servers

The following parameters are all optional.

Parameters for starting a catalog server:

-catalogServiceEndPoints <serverName:hostName:clientPort:peerPort>
Specifies a list of catalog servers to link together into a catalog service domain. Each attribute is defined as follows:

serverName
Specifies the name of the catalog server.
hostName
Specifies the host name for the computer where the server is launched.
clientPort
Specifies the port that is used for peer catalog service communication.
peerPort
This value is the same as the haManagerPort. Specifies the port that is used for peer catalog service communication.

The following example starts the cs1 catalog server, which is in the same catalog service domain as the cs2 and cs3 servers:

```
startOgServer.bat|sh cs1 -catalogServiceEndPoints
cs1:MyServer1.company.com:6601:6602,cs2:MyServer2.company.com:6601:6602,cs3:MyServer3.company.com:6601:6602
```

If you start additional catalog servers, they must include the same servers in the -catalogServiceEndPoints argument. The order of the list can be different, but the servers contained in the list must be the same for each catalog server. Do not put any spaces in the list.
-clusterSecurityFile <cluster security xml file>

Specifies the objectGridSecurity.xml file on the hard disk, which describes the security properties that are common to all servers (including catalog servers and container servers). One of the property example is the authenticator configuration which represents the user registry and authentication mechanism.
Example: `/opt/xs/ogsecurity.xml`

Windows Important: If you are using Windows, the directory path does not support backslashes. If you have used backslashes, you must escape any backslash (\ ) characters in the path. For example, if you want to use the path C:\opt\ibm, enter C:\\opt\\ibm in the properties file. Windows directories with spaces are not supported.

-clusterSecurityUrl <cluster security xml URL>
Specifies the objectGridSecurity.xml file as a URL to the file on the hard disk or on the network, which describes the security properties that are common to all servers (including catalog servers and container servers). One of the property example is the authenticator configuration which represents the user registry and authentication mechanism.
Example: `file:///opt/xs/ogsecurity.xml`

-domain <domain name>
Specifies the name of the catalog service domain for this catalog server. The catalog service domain creates a group of highly available catalog servers. Each catalog server for a single domain should specify the same value for the -domain parameter.

-haManagerPort <port>
Specifies the port that is used by the high availability (HA) manager for heartbeat communication between peer container servers. The haManagerPort port is only used for peer-to-peer communication between container servers that are in same domain. If the haManagerPort property is not defined, then an ephemeral port is used. In WebSphere® Application Server, this setting is inherited by the high availability manager port configuration.

-heartbeat 0|1|-1
Specifies how often a server failover is detected. An aggressive heartbeat interval can be useful when the processes and network are stable. If the network or processes are not optimally configured, heartbeats might be missed, which can result in a false failure detection. The heartbeat frequency level is a trade-off between use of resources and failure discovery time. The more frequent a heartbeat occurs, then more resources are used. However, failures are discovered more quickly. This property applies to the catalog service only.

Table 1. Valid heartbeat values

Value	Action	Description
-1	Aggressive	Specifies an aggressive heartbeat level. With this value, failures are detected more quickly, but more processor and network resources are used. This level is more sensitive to missing heartbeats when the server is busy. Failovers are typically detected within 5 seconds.
0	Typical (default)	Specifies a heartbeat level at a typical rate. With this value, failover detection occurs at a reasonable rate without overusing resources. Failovers are typically detected within 30 seconds.
1	Relaxed	Specifies a relaxed heartbeat level. With this value, a decreased heartbeat frequency increases the time to detect failures, but also decreases processor and network use. Failovers are typically detected within 180 seconds.

-JMXConnectorPort <port>
Defines the Secure Sockets Layer (SSL) port to which the Java Management Extensions (JMX) service binds. Only required if an SSL transport protocol is needed for JMX data.

-JMXServicePort <port>
Required only for WebSphere eXtreme Scale in a stand-alone environment. Specifies the port number on which the MBean server listens for communication with Java Management Extensions (JMX).
Default: `1099` for catalog servers

-jvmArgs <JVM arguments>
Specifies a set of JVM arguments. Every option after the -jvmArgs option is used to start the server Java virtual machine (JVM). When the -jvmArgs parameter is used, ensure that it is the last optional script argument specified.
Example: `-jvmArgs -Xms256M -Xmx1G`

-listenerHost <host name>
Specifies the host name to which the Object Request Broker (ORB) transport protocol binds for communication. The value must be a fully qualified domain name or IP address. If your configuration involves multiple network cards, set the listener host and port to the IP address for which to bind. By setting the listener and host port, it allows the transport mechanism in the JVM know which IP address to use. If you do not specify which IP address to use, symptoms such as connection timeouts, unusual API failures, and clients that seem to hang can occur.

Default: `localhost`

-listenerPort <port>
Specifies the port number to which the ORB transport protocol binds for communication. **Default:** `2809`

-quorum true|false
Enables quorum for the catalog service. Quorum is used to ensure that most of the catalog service domain is available before partitions are moved to the available container servers. To enable quorum, set the value to `true` or `enabled`. The default value is `disabled`. This property applies to the catalog service only. For more information, see Catalog server quorums.

-script <script file>
Specifies the location of a custom script for commands you specify to start catalog servers or containers and then parameterize or edit as you require.

-serverProps <server properties file>
Specifies the server property file that contains the server-specific security properties. The file name specified for this property is just in plain file path format, such as c:/tmp/og/catalogserver.props.

-traceSpec <trace specification>
Enables trace and the trace specification string for the container server. Trace is disabled by default. This property applies to both the container server and the catalog service. Examples:

- `ObjectGrid=all=enabled`
- `ObjectGrid*=all=enabled`

-traceFile <trace file>
Specifies a file name to write trace information. This property applies to both the container server and the catalog service.

-timeout <seconds>
Specifies a number of seconds before the server start times out.

Usage for container servers

Windows

```
startOgServer.bat <server> -objectgridFile <xml file>
-deploymentPolicyFile <xml file> [options]
```

Windows

```
startOgServer.bat <server> -objectgridUrl <xml URL>
-deploymentPolicyUrl <xml URL> [options]
```

UNIX

```
startOgServer.sh <server> -objectgridFile <xml file>
-deploymentPolicyFile <xml file> [options]
```

UNIX

```
 startOgServer.sh <server> -objectgridUrl <xml URL>
-deploymentPolicyUrl <xml URL> [options]
```

Options for container servers

-catalogServiceEndPoints <hostName:port,hostName:port>
Specifies the Object Request Broker (ORB) host and port on the catalog service.
Default: localhost:2809

-deploymentPolicyFile <deployment policy xml file>
Specifies the path to the deployment policy file on the hard disk. The deployment policy is used to set up partitioning and replication. The deployment policy can also be used to influence placement behavior.
Example: ../xml/SimpleDP.xml

-deploymentPolicyUrl <deployment policy url>
Specifies the URL for the deployment policy file on the hard disk or on the network. The deployment policy is used to set up partitioning and replication. The deployment policy can also be used to influence placement behavior.
Example: file://xml/SimpleDP.xml

-JMXConnectorPort <port>
Defines the Secure Sockets Layer (SSL) port to which the Java Management Extensions (JMX) service binds. Only required if an SSL transport protocol is needed for JMX data.
-JMXServicePort <port>
Required only for WebSphere eXtreme Scale in a stand-alone environment. Specifies the port number on which the MBean server listens for communication with Java Management Extensions (JMX).

Default: `1099`
-jvmArgs <JVM arguments>
Specifies a set of JVM arguments. Every option after the -jvmArgs option is used to start the server Java virtual machine (JVM). When the -jvmArgs parameter is used, ensure that it is the last optional script argument specified.
Example:`-jvmArgs -Xms256M -Xmx1G`

-listenerHost <host name>
Specifies the host name to which the Object Request Broker (ORB) transport protocol binds for communication. The value must be a fully qualified domain name or IP address. If your configuration involves multiple network cards, set the listener host and port to the IP address for which to bind. By setting the listener and host port, it allows the transport mechanism in the JVM know which IP address to use. If you do not specify which IP address to use, symptoms such as connection timeouts, unusual API failures, and clients that seem to hang can occur.

Default: `localhost`
-listenerPort <port>
Specifies the port number to which the ORB transport protocol binds for communication. **Default:** `2809`
-objectgridFile <ObjectGrid descriptor xml file>
Specifies the path to the ObjectGrid descriptor file. The ObjectGrid XML file specifies which eXtreme Scale servers the container hosts.
-objectgridUrl <ObjectGrid descriptor url>
Specifies a URL for the ObjectGrid descriptor file. The ObjectGrid XML file specifies which eXtreme Scale servers the container hosts.
-script <script file>
Specifies the location of a custom script for commands you specify to start catalog servers or containers and then parameterize or edit as you require.
-serverProps <server properties file>
Specifies the path to the server property file.
Example:../security/server.props

-timeout <seconds>
Specifies a number of seconds before the server start times out.
-traceFile <trace file>
Specifies a file name to write trace information. This property applies to both the container server and the catalog service.

-traceSpec <trace specification>
Enables trace and the trace specification string for the container server. Trace is disabled by default. This property applies to both the container server and the catalog service. Examples:

- `ObjectGrid=all=enabled`
- `ObjectGrid*=all=enabled`

-zone <zone name>
Specifies the zone to use for all of the containers within the server. See Zone-preferred routing for more information about configuring zones.

Related concepts:
Statistics modules
Tuning Java virtual machines
Java SE considerations
Java EE considerations
Tuning garbage collection with WebSphere Real Time
Related tasks:
Starting a stand-alone catalog service
Starting container servers
Starting and stopping stand-alone servers
Monitoring with CSV files
Enabling statistics
Monitoring with the statistics API
Administering with the xscmd utility
Related information:
Getting started tutorial lesson 3.1: Starting catalog and container servers
StatsSpec class
Tuning the IBM virtual machine for Java

Stopping stand-alone servers

You can use the stopOgServer script to stop eXtreme Scale server processes.

About this task

Run the **stopOgServer** script by navigating to the bin directory:

```
cd wxs_install_root/bin
```

Procedure

- **Stop a single container server.**
  Run the **stopOgServer** script to stop the container server. Use this command only when you are stopping a single container server. If you run the single catalog server stop command on several container servers in succession, you might see performance and churn issues for shard placement.

  ```
  stopOgServer containerServer -catalogServiceEndPoints MyServer1.company.com:2809
  ```

  Attention: The -catalogServiceEndPoints option should match the value of the -catalogServiceEndPoints option that was used to start the container. If a -catalogServiceEndPoints was not used to start the container, the default values are likely localhost or the hostname and 2809 for the ORB port to connect to the catalog service. Otherwise, use the values that are passed to -listenerHost and -listenerPort on the catalog service. If the -listenerHost and -listenerPort options are not used when starting the catalog service, the ORB binds to port 2809 on the localhost for the catalog service.
- **Stop multiple container servers.**
  To prevent churn and performance issues for shard placement when you want to stop multiple container servers at the same time, use the following command format. Separate a list of container servers with commas:

  ```
  stopOgServer containerServer0,containerServer1,containerServer2
  -catalogServiceEndPoints MyServer1.company.com:2809
  ```

  If you want to stop all of the containers on a specific zone or host, you can use the -teardown parameter. See Stopping servers gracefully with the xscmd utility for more information.
- **Stop catalog servers.**
  Run the **stopOgServer** script to stop the catalog server.

  ```
  stopOgServer.sh catalogServer -catalogServiceEndPoints MyServer1.company.com:2809
  ```

  Attention: When you are stopping a catalog service, use the -catalogServiceEndPoints option to reference the Object Request Broker (ORB) host and port on the catalog service. The catalog service uses -listenerHost and -listenerPort options to specify the host and port for ORB binding or accepts the default binding. If the -listenerHost and -listenerPort options are not used when starting the catalog service, the ORB binds to port 2809 on the localhost for the catalog service. The -catalogServiceEndPoints option is different when stopping a catalog service than when you started the catalog service.
  Starting a catalog service requires peer access ports and client access ports, if the default ports were not used. Stopping a catalog service requires only the ORB port.

- **Stop the web console server.** To stop the web console server, run the **stopConsoleServer.bat|sh** script. This script is in the *wxs_install_root*/ObjectGrid/bin directory of your installation. For more information, see Starting and logging on to the web console.
- **Enable trace for the server stop process.**

If a container fails to stop, you can enable trace to help with debugging the problem. To enable trace during the stop of a server, add the -traceSpec and -traceFile parameters to the stop commands. The -traceSpec parameter specifies the type of trace to enable and the -traceFile parameter specifies path and name of the file to create and use for the trace data.

1. From the command line, navigate to the bin directory.

   ```
   cd wxs_install_root/bin
   ```

2. Run the **stopOgServer** script with trace enabled.

   ```
   stopOgServer.sh c4 -catalogServiceEndPoints MyServer1.company.com:2809
    -traceFile ../logs/c4Trace.log -traceSpec ObjectGrid=all=enabled
   ```

   After the trace is obtained, look for errors related to port conflicts, missing classes, missing or incorrect XML files or any stack traces. Suggested startup trace specifications are:
   - `ObjectGrid=all=enabled`
   - `ObjectGrid*=all=enabled`

   For all of the trace specification options, see Server trace options.

- **Stop embedded servers programmatically.**
  For more information about stopping embedded servers programmatically, see Using the embedded server API to start and stop servers.

Related tasks:
Starting stand-alone servers

stopOgServer script

The **stopOgServer** script stops catalog and container servers.

Purpose

Use the **stopOgServer** script to stop a server. You must provide the name of the server and its catalog service endpoints.
Deprecated: The **startOgServer** and **stopOgServer** commands start servers that use the Object Request Broker (ORB) transport mechanism. The ORB is deprecated, but you can continue using these scripts if you were using the ORB in a previous release. The IBM eXtremeIO (XIO) transport mechanism replaces the ORB. Use the **startXsServer** and **stopXsServer** scripts to start and stop servers that use the XIO transport.

Location

The **stopOgServer** script is in the bin directory of the root directory, for example:

```
cd wxs_install_root/bin
```

Usage

To stop a catalog or container server: Windows

```
stopOgServer.bat <server_name> -catalogServiceEndPoints
<csHost:csListenerPort,csHost:csListenerPort> [options]
```

UNIX

```
stopOgServer.sh <server_name> -catalogServiceEndPoints
<csHost:csListenerPort,csHost:csListenerPort> [options]
```

Options

-catalogServiceEndPoints <csHost:csListenerPort, csHost:csListenerPort...>
Specifies the Object Request Broker (ORB) host and port number.
For container severs: The list of catalog service endpoints should be the same as the list that was used to start the container server. If you did not specify this option when you started the container server, use the default value of `localhost:2809`.

For catalog servers: If you are stopping the catalog service, use the values that you indicated for the -listenerHost and -listenerPort options when you started the catalog service. If you did not specify these options when you started the catalog server, use the default value of `localhost:2809`. The -catalogServiceEndPoints value you use when you stop the catalog service is different from when you start the catalog service.

-clientSecurityFile <security properties file>
Specifies the path to the client properties file that defines security properties for the client. See Client properties file for more information about the security settings in this file.

-traceSpec <trace specification>
Enables trace and the trace specification string for the container server. Trace is disabled by default. This property applies to both the container server and the catalog service. Examples:

- `ObjectGrid=all=enabled`
- `ObjectGrid*=all=enabled`

-traceFile <trace file>

Specifies a file name to write trace information. This property applies to both the container server and the catalog service.

-jvmArgs <JVM arguments>
Specifies a set of JVM arguments. Every option after the -jvmArgs option is used to start the server Java™ virtual machine (JVM). When the -jvmArgs parameter is used, ensure that it is the last optional script argument specified.
Example:`-jvmArgs -Xms256M -Xmx1G`

Related tasks:
Viewing statistics with the web console
Monitoring with the web console
Starting and logging on to the web console
Connecting the web console to catalog servers
Monitoring with the xscmd utility
Administering with the xscmd utility
Related information:
Getting started tutorial lesson 4: Monitor your environment

Stopping servers gracefully with the xscmd utility

You can use the **xscmd** utility with the -c teardown command to stop a list or group of catalog and container servers. This command simplifies shutting down all or portions of a data grid. It also prevents unnecessary placement and recovery catalog service actions that normally occur when processes are stopped ungracefully.

Procedure

- Stop a specific list of servers.
  Provide a list of servers after the -teardown parameter:

  ```
  xscmd -c teardown -sl catalogServer1,catalogServer2,containerServer1
  ```

- Stop all the servers in a specific zone.
  Use the -z parameter and provide the name of the zone. The catalog server determines the servers that are running in the zone. The **xscmd** utility also prompts you with a list of the servers in the selected zone before the servers are shut down.

  ```
  xscmd -c teardown -z zone_name
  ```

- Stop all the servers on a specific host. For example, to shut down all the servers on `myhost.mycompany.com`, enter `-hf myhost.mycompany.com`.
  Use the -hf parameter and provide the name of the host. The catalog server determines the servers that are running on the host. The **xscmd** utility prompts you with a list of the servers in the selected host before the servers are shut down.

  ```
  xscmd -teardown -hf <host_name>
  ```

  Attention: By default, the JVM continues to run when each eXtreme Scale server in an OSGi framework is stopped in the **xscmd** utility with the **-c teardown** command. If you want eXtreme Scale to exit the JVM, then this type of implementation must be planned for. You must set the server property exitJVMOnTeardown to `true` before the server is started. For more information, see Server properties file.

Related tasks:
Administering with the xscmd utility
Related reference:
xsadmin utility reference
xscmd utility reference

Starting and stopping servers in a WebSphere Application Server environment

Catalog and container servers can automatically start in a WebSphere® Application Server or WebSphere Application Server Network Deployment environment.

Before you begin

Configure catalog servers and container servers to run on WebSphere Application Server:

- Configuring the catalog service in WebSphere Application Server
- Configuring container servers in WebSphere Application Server

About this task

The life cycle of catalog and container servers in WebSphere Application Server is linked to the processes on which these servers run.

Procedure

- **Start catalog services in WebSphere Application Server**:

The life cycle a catalog server is tied to the WebSphere Application Server process. After you configure the catalog service domain in WebSphere Application Server, restart each server that you defined as a part of the catalog service domain. The catalog service starts automatically on the servers that you associated with the catalog service domain. The catalog service can also start automatically in the following scenarios, depending on the edition of WebSphere Application Server:

  - **Base WebSphere Application Server**: You can configure your application to automatically start a container server and catalog service. This feature simplifies unit testing in development environments such as Rational® Application Developer because you do not need to explicitly start a catalog service. See Configuring WebSphere Application Server applications to automatically start container servers for more information.
  - **WebSphere Application Server Network Deployment**: The catalog service automatically starts in the deployment manager process if the deployment manager node has WebSphere eXtreme Scale installed and the deployment manager profile is augmented. See Configuring the catalog service in WebSphere Application Server for more information.

- **Start container servers in WebSphere Application Server**:
  The life cycle of a container server is tied to the WebSphere Application Server application. When you start the configured application, the container servers also start.

- **Stop an entire data grid of servers**:
  You can stop catalog and container servers by stopping the applications and associated application servers. However, you can also stop an entire data grid with the **xscmd** utility or MBeans:
  - **In the xscmd utility:**
    See Stopping servers gracefully with the xscmd utility for more information about stopping an entire data grid.

  - **With Mbeans:**
    Use the tearDownServers operation on the PlacementServiceMBean Mbean.

- **Stop Liberty profile processes**:
  You can stop Liberty servers using the Liberty profile server command.

Related concepts:
Configuration considerations for multi-master topologies
8.5+ Installing fix packs using IBM Installation Manager
Example: Configuring catalog service domains
Related tasks:
Retrieving eXtreme Scale environment information with the xscmd utility
Updating eXtreme Scale servers
Configuring multiple data center topologies
Migrating to WebSphere eXtreme ScaleVersion 8.5
Starting and stopping stand-alone servers
Related reference:
Catalog service domain administrative tasks

Starting and stopping servers in the Liberty profile

8.5+ Use the Liberty profile **server** command to start and stop WebSphere® eXtreme Scale servers in the Liberty profile.

Before you begin

The Liberty profile is the Java virtual machine (JVM) process that starts in OSGi and provides a flexible framework for serving applications. The eXtreme Scale server is the mechanism that provides grid services. The main two services are the catalog service and the container service. A Liberty profile is started from the command line with the `server start server_name` or `server run server_name` commands. When the Liberty profile starts, it reads the `server.xml` file to determine what Liberty profile features to start. WebSphere eXtreme Scale has three features: server, client, and web. If you add the server feature to the feature manager, then the Liberty profile starts the eXtreme Scale server bundles. However, adding the server feature does not necessarily start any eXtreme Scale servers.

The eXtreme Scale servers start only when a valid service is configured. For example, if you want to configure a catalog service, then the `isCatalog="true"` attribute must be set on the `xsServer ...` element in the server.xml file.

The following options are available to configure a container service:

- Copy a valid objectgrid.xml file (with or without an accompanying objectGridDeployment.xml file) into the wlp_home/usr/servers/*server_name*/grids directory. This grids directory is a monitored directory, and changes to files in this directory initiate events in the Liberty profile runtime environment. For example, when new objectgrid.xml, objectGridDeployment.xml, or both files are found, eXtreme Scale creates and starts a new container. When one of these files are deleted, eXtreme Scale stops that container. When files are modified, eXtreme Scale stops and restarts the container. Multiple containers can exist in the same eXtreme Scale server, which requires that subdirectories exist inside the grids directory.
- Install an OSGi bundle. This bundle must reference a blueprint.xml file that contains server metadata. This method of starting a server is similar to how you can start servers OSGi environments without the Liberty profile in WebSphere eXtreme Scale Version 7.1.1. In Version 8.5, the server element is no longer required in the blueprint.xml file. Therefore, you must define the server metadata in the server.xml file. Additionally, bundles are simple to install and start by dropping them into the grids directory in a similar way that you drop XML files in the grids directory.
- Use the embedded server API. This option is similar to the process for starting the server in a stand-alone environment. With the Liberty profile, however, you must ensure that you run your code to start the eXtreme Scale server.

About this task

Use this task to start eXtreme Scale servers with the Liberty profile **server** command. The wlp/bin directory contains a script called **server** to help control the server process. The following syntax for this command is supported:

```
server <task> [server] [options]
```

Procedure

- Start eXtreme Scale servers. When you run the **start** command, the server is launched as a background process. Use the following example to start the server:

  ```
  bin/server start server_name
  bin/server.bat start server_name
  ```

- Stop eXtreme Scale servers; for example: When you run the **stop** command, the running server is stopped. Use the following example to stop the server:

  ```
  bin/server stop server_name
  bin/server.bat stop server_name
  ```

Using the embedded server API to start and stop servers

With WebSphere® eXtreme Scale, you can use a programmatic API for managing the life cycle of embedded servers and containers. You can programmatically configure the server with any of the options that you can also configure with the command line options or file-based server properties. You can configure the embedded server to be a container server, a catalog service, or both.

Before you begin

- You must have a method for running code from within an already existing Java™ virtual machine. The eXtreme Scale classes must be available through the class loader tree.
- 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

You can run many administration tasks with the Administration API. One common use of the API is as an internal server for storing Web application state. The Web server can start an embedded WebSphere eXtreme Scale server, report the container server to the catalog service, and the server is then added as a member of a larger distributed grid. This usage can provide scalability and high availability to an otherwise volatile data store.

You can programmatically control the complete life cycle of an embedded eXtreme Scale server. The examples are as generic as possible and only show direct code examples for the outlined steps.

Procedure

1. Obtain the ServerProperties object from the ServerFactory class and configure any necessary options. For more information about the ServerProperties interface, see ServerProperties interface.
   Every eXtreme Scale server has a set of configurable properties. When a server starts from the command line, those properties are set to defaults, but you can override several properties by providing an external source or file. In the embedded scope, you can directly set the properties with a ServerProperties object. You must set these properties before you obtain a server instance from the ServerFactory class. The following example snippet obtains a ServerProperties object, sets the CatalogServiceBootStrap field, and initializes several optional server settings. See the API documentation for a list of the configurable settings.

   ```
   ServerProperties props = ServerFactory.getServerProperties();
   props.setCatalogServiceBootstrap("host:port"); // required to connect to specific catalog service
   props.setServerName("ServerOne"); // name server
   props.setTraceSpecification("com.ibm.ws.objectgrid=all=enabled"); // Sets trace spec
   ```

2. If you want the server to be a catalog service, obtain the CatalogServerProperties object.
   For more information about the CatalogServerProperties interface, see CatalogServerProperties interface.

   Every embedded server can be a catalog service, a container server, or both a container server and a catalog service. The following example obtains the CatalogServerProperties object, enables the catalog service option, and configures various catalog service settings.

   ```
   CatalogServerProperties catalogProps = ServerFactory.getCatalogProperties();
   catalogProps.setCatalogServer(true); // false by default, it is required to set as a catalog service
   catalogProps.setQuorum(true); // enables / disables quorum
   ```

3. Obtain a Server instance from the ServerFactory class. The Server instance is a process-scoped singleton that is responsible for managing the membership in the grid. After this instance has been instantiated, this process is connected and is highly available with the other servers in the grid.
   For more information about the Server interface, see Server interface. For more information abut the ServerFactory class, see ServerFactory class.

   The following example shows how to create the Server instance:

   ```
   Server server = ServerFactory.getInstance();
   ```

   Reviewing the previous example, the ServerFactory class provides a static method that returns a Server instance. The ServerFactory class is intended to be the only interface for obtaining a Server instance. Therefore, the class ensures that the instance is a singleton, or one instance for each JVM or isolated classloader. The getInstance method initializes the Server instance. You must configure all the server properties before you initialize the instance. The Server class is responsible for creating new Container instances. You can use both the ServerFactory and Server classes for managing the life cycle of the embedded Server instance.

For more information about the Container interface, see Container interface.

4. Start a Container instance using the Server instance.
   Before shards can be placed on an embedded server, you must create a container on the server. The Server interface has a createContainer method that takes a DeploymentPolicy argument. The following example uses the server instance that you obtained to create a container using a created DeploymentPolicy file. Note that Containers require a classloader that has the application binaries available to it for serialization. You can make these binaries available by calling the createContainer method with the Thread context classloader set to the classloader that you want to use.

```
DeploymentPolicy policy = DeploymentPolicyFactory.createDeploymentPolicy(new
          URL("file://urltodeployment.xml"),
        new URL("file://urltoobjectgrid.xml"));
Container container = server.createContainer(policy);
```

5. Remove and clean up a container server.
   You can remove and clean up a container server by using the running the teardown method on the obtained Container instance. Running the teardown method on a container properly cleans up the container and removes the container from the embedded server.

   The process of cleaning up the container includes the movement and tearing down of all the shards that are placed within that container. Each server can contain many containers and shards. Cleaning up a container does not affect the life cycle of the parent Server instance. The following example demonstrates how to run the teardown method on a server. The teardown method is made available through the ContainerMBean interface. By using the ContainerMBean interface, if you no longer have programmatic access to this container, you can still remove and clean up the container with its MBean. A terminate method also exists on the Container interface, do not use this method unless it is absolutely needed. This method is more forceful and does not coordinate appropriate shard movement and clean up.

```
container.teardown();
```

6. Stop the embedded server.
   When you stop an embedded server, you also stop any containers and shards that are running on the server. When you stop an embedded server, you must clean up all open connections and move or tear down all the shards. The following example demonstrates how to stop a server and using the waitFor method on the Server interface to ensure that the Server instance shuts down completely. Similarly to the container example, the stopServer method is made available through the ServerMBean interface. With this interface, you can stop a server with the corresponding Managed Bean (MBean).

```
ServerFactory.stopServer();  // Uses the factory to kill the Server singleton
 // or
server.stopServer(); // Uses the Server instance directly
server.waitFor(); // Returns when the server has properly completed its shutdown procedures
```

   Full code example:

```
 import java.net.MalformedURLException;
import java.net.URL;

import com.ibm.websphere.objectgrid.ObjectGridException;
import com.ibm.websphere.objectgrid.deployment.DeploymentPolicy;
import com.ibm.websphere.objectgrid.deployment.DeploymentPolicyFactory;
import com.ibm.websphere.objectgrid.server.Container;
import com.ibm.websphere.objectgrid.server.Server;
import com.ibm.websphere.objectgrid.server.ServerFactory;
import com.ibm.websphere.objectgrid.server.ServerProperties;

public class ServerFactoryTest {

    public static void main(String[] args) {

        try {

            ServerProperties props = ServerFactory.getServerProperties();
            props.setCatalogServiceBootstrap("catalogservice-hostname:catalogservice-port");
            props.setServerName("ServerOne"); // name server
            props.setTraceSpecification("com.ibm.ws.objectgrid=all=enabled"); // TraceSpec

            /*
             * In most cases, the server will serve as a container server only
             * and will connect to an external catalog service. This is a more
             * highly available way of doing things. The commented code excerpt
             * below will enable this Server to be a catalog service.
             *
             *
             * CatalogServerProperties catalogProps =
             * ServerFactory.getCatalogProperties();
             * catalogProps.setCatalogServer(true); // enable catalog service
             * catalogProps.setQuorum(true); // enable quorum
             */

            Server server = ServerFactory.getInstance();

            DeploymentPolicy policy = DeploymentPolicyFactory.createDeploymentPolicy
                                      (new URL("url to deployment xml"),  new URL("url to objectgrid xml file"));
            Container container = server.createContainer(policy);

            /*
             * Shard will now be placed on this container if the deployment requirements are met.
             * This encompasses embedded server and container creation.
             *
             * The lines below will simply demonstrate calling the cleanup methods
             */
```

```
            container.teardown();
            server.stopServer();
            int success = server.waitFor();

        } catch (ObjectGridException e) {
            // Container failed to initialize
        } catch (MalformedURLException e2) {
            // invalid url to xml file(s)
        }

    }

}
```

- Embedded server API
  WebSphere eXtreme Scale includes application programming interfaces (APIs) and system programming interfaces for embedding eXtreme Scale servers and clients within your existing Java applications.

Related concepts:
Installation topologies
Catalog service
Container servers, partitions, and shards
Related tasks:
Configuring catalog and container servers
Starting and stopping stand-alone servers
Related reference:
Server properties file
ObjectGrid descriptor XML file

Embedded server API

WebSphere® eXtreme Scale includes application programming interfaces (APIs) and system programming interfaces for embedding eXtreme Scale servers and clients within your existing Java™ applications.

Instantiate the eXtreme Scale server

You can use several properties to configure the eXtreme Scale server instance, which you can retrieve from the ServerFactory.getServerProperties method. The ServerProperties object is a singleton, so each call to the getServerProperties method retrieves the same instance.

You can create a server with the following code.

```
Server server = ServerFactory.getInstance();
```

All properties set before the first invocation of the getInstance method are used to initialize the server.

Set server properties

You can set the server properties until the ServerFactory.getInstance method is called for the first time. The first call of the getInstance method instantiates the eXtreme Scale server, and reads all the configured properties. Setting the properties after creation has no effect. The following example shows how to set properties before instantiating a Server instance.

```
// Get the server properties associated with this process.
ServerProperties serverProperties = ServerFactory.getServerProperties();

// Set the server name for this process.
serverProperties.setServerName("EmbeddedServerA");

// Set the name of the zone this process is contained in.
serverProperties.setZoneName("EmbeddedZone1");

// Set the end point information required to bootstrap to the catalog service.
serverProperties.setCatalogServiceBootstrap("localhost:2809");

// Set the listener host name to use to bind to.
serverProperties.setListenerHost("host.local.domain");

// Set the listener port to use to bind to.
serverProperties.setListenerPort(9010);

// Turn off all MBeans for this process.
serverProperties.setMBeansEnabled(false);

Server server = ServerFactory.getInstance();
```

Embed the catalog service

Any JVM setting that is flagged by the CatalogServerProperties.setCatalogServer method can host the catalog service for eXtreme Scale. This method indicates to the eXtreme Scale server run time to instantiate the catalog service when the server is started. The following code shows how to instantiate the eXtreme Scale

catalog server:

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

Server server = ServerFactory.getInstance();
```

Embed a container server

Run the Server.createContainer method for any JVM to host multiple eXtreme Scale container servers. The following code shows how to instantiate a container server:

```
Server server = ServerFactory.getInstance();
DeploymentPolicy policy = DeploymentPolicyFactory.createDeploymentPolicy(
    new File("META-INF/embeddedDeploymentPolicy.xml").toURI().toURL(),
    new File("META-INF/embeddedObjectGrid.xml").toURI().toURL());
Container container = server.createContainer(policy);
```

Self-contained server process

You can start all the services together, which is useful for development and also practical in production. By starting the services together, a single process does all of the following actions: starts the catalog service, starts a set of containers, and runs the client connection logic. Starting the services in this way sorts out programming issues before deploying in a distributed environment. The following code shows how to instantiate a self-contained eXtreme Scale server:

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

Server server = ServerFactory.getInstance();
DeploymentPolicy policy = DeploymentPolicyFactory.createDeploymentPolicy(
    new File("META-INF/embeddedDeploymentPolicy.xml").toURI().toURL(),
    new File("META-INF/embeddedObjectGrid.xml").toURI().toURL());
Container container = server.createContainer(policy);
```

Embed eXtreme Scale in WebSphere Application Server

The configuration for eXtreme Scale is set up automatically when you install eXtreme Scale in a WebSphere Application Server environment. You are not required to set any properties before you access the server to create a container. The following code shows how to instantiate an eXtreme Scale server inWebSphere Application Server:

```
Server server = ServerFactory.getInstance();
DeploymentPolicy policy = DeploymentPolicyFactory.createDeploymentPolicy(
    new File("META-INF/embeddedDeploymentPolicy.xml").toURI().toURL(),
    new File("META-INF/embeddedObjectGrid.xml").toURI().toURL);
Container container = server.createContainer(policy);
```

For a step by step example on how to start an embedded catalog service and container programmatically, see Using the embedded server API to start and stop servers.

Related tasks:
Configuration methods
Configuring data grids
Connecting to distributed ObjectGrid instances programmatically
Configuring deployment policies
Installing eXtreme Scale bundles
Installing the Liberty profile developer tools for WebSphere eXtreme Scale

Related reference:
ObjectGrid descriptor XML file
Deployment policy descriptor XML file
Server properties file
Client properties file

Related information:
ObjectGridManager interface
ClientClusterContext interface
DeploymentPolicy interface

Administering with the xscmd utility

With the **xscmd** utility, you can complete administrative tasks in the environment such as: establishing multi-master replication links, overriding quorum, and stopping groups of servers with the teardown command.

Before you begin

- Your catalog servers and container servers must be started. If your catalog servers are in a catalog service domain, at least two catalog servers must be started.

- 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 **xscmd** utility replaces the **xsadmin** sample utility as a fully supported monitoring and administration tool. You could complete similar operations with the **xsadmin** tool, but this tool is not supported. The **xsadmin** sample provides a method for parsing and discovering current deployment data, and can be used as a foundation for writing custom utilities. If you were previously using the **xsadmin** tool for monitoring and administration, consider updating your scripts to use the **xscmd** utility. For information about mapping **xsadmin** commands to the new **xscmd** commands, see xsadmin tool to xscmd tool migration.

Procedure

1. Optional: If client authentication is enabled: Open a command-line window. On the command line, set appropriate environment variables.
2. Go to the *wxs_home*/bin directory.
   `cd wxs_home/bin`
3. Display help for the various **xscmd** options.
   - To display the general help, run the following command:
     - UNIX `./xscmd.sh -h`
     - Windows `xscmd.bat -h`
   - To display a list of all of the commands, run the following command:
     - UNIX `./xscmd.sh -lc`
     - Windows `xscmd.bat -lc`
   - To display the help for a specific command, run the following command:
     - UNIX `./xscmd.sh -h command_name`
     - Windows `xscmd.bat -h command_name`
   - To display a list of the command groups, run the following command:
     - UNIX `./xscmd.sh -lcg`
     - Windows `xscmd.bat -lcg`
   - To display a list of the commands within a command group, run the following command:
     - UNIX `./xscmd.sh -lc command_group_name`
     - Windows `xscmd.bat -lc command_group_name`
4. Run commands that connect to specific catalog servers. By default, **xscmd** connects to the catalog server on the local host, using the host name and port of `localhost:2809`. You can also provide a list of host names and ports to the command so that you can connect to catalog servers on other hosts. From the list, the **xscmd** utility connects to a random host. The list of hosts that you provide must be within the same catalog service domain.
   - Provide a list of stand-alone catalog servers to connect:
     - UNIX `./xscmd.sh -c <command_name> -cep hostname:port(,hostname:port)`
     - Windows `xscmd.bat -c <command_name> -cep hostname:port(,hostname:port)`

     In the previous commands, *command_name* is the name of the command that you are running. The *hostname:port* value is the catalog server host name and listener port. The listener port value on a stand-alone catalog server is specified when you run the **startOgServer** command.
   - Provide a list of WebSphere® Application Server catalog servers to connect. You cannot connect to catalog servers that are running on WebSphere Application Server with the default localhost value:
     - UNIX `./xscmd.sh -c <command_name> -cep was_hostname:port(,hostname:port)`
     - Windows `xscmd.bat -c <command_name> -cep was_hostname:port(,hostname:port)`

     In the previous commands, *command_name* is the name of the command that you are running. The *was_hostname* value is the host name of the catalog server in the WebSphere Application Server cell. The *port* value is the listener port. The listener port value in WebSphere Application Server is inherited by the BOOTSTRAP_ADDRESS port configuration. The default value is `9809` if the catalog server is running on the deployment manager. If you are running the catalog server on an application server, check the BOOTSTRAP_ADDRESS port configuration of the application server to determine the port number.
     Important: If your container servers are running in a secured WebSphere Application Server environment, run the **xscmd** utility from the WebSphere eXtreme Scale Client installation in the WebSphere Application Server environment. For example, from the /opt/IBM/WebSphere/AppServer/bin directory.

Related concepts:
IBM eXtremeMemory
Zones
Zone-preferred routing
Catalog server quorums
Catalog service
Planning for network ports
Statistics modules
Example: Configuring catalog service domains
Administering
Related tasks:
Planning to use IBM eXtremeMemory
Configuring IBM eXtremeMemory
Updating eXtreme Scale servers
Viewing zone information with the xscmd utility
Managing data center failures
Managing data center failures when quorum is enabled
Configuring the quorum mechanism
Controlling placement
Starting stand-alone servers
Administering OSGi-enabled services using the xscmd utility
Updating OSGi services for eXtreme Scale plug-ins with xscmd

Stopping servers gracefully with the xscmd utility
Configuring security profiles for the xscmd utility
Monitoring with the xscmd utility
Enabling logging
Collecting trace
Related reference:
Web console statistics
stopOgServer script
Example: Zone definitions in the deployment policy descriptor XML file
Deployment policy descriptor XML file
xscmd utility reference
Server properties file
startOgServer script
xsadmin tool to xscmd tool migration
Server trace options
Messages
Related information:
Module 5: Use the xscmd utility to monitor data grids and maps
Module 5: Use the xscmd tool to monitor data grids and maps
Getting started tutorial lesson 4: Monitor your environment
Interface PlacementServiceMBean
Eclipse runtime options
Port number settings in WebSphere Application Server versions
StatsSpec class

Controlling placement

You can use several different options to control when shards are placed on various container servers in the configuration. During startup, you might choose to delay the placement of shards. When you are running all of your container servers, you might need to suspend, resume, or change placement while you maintain servers.

Procedure

Controlling placement during startup

You can control when shards begin to be placed while your environment is starting. Some control is in place by default. If you do not take any actions to control placement, shards begin to be placed immediately. When shards are placed immediately, the shards might not be placed evenly as subsequent container servers start, and further placement operations run to balance the distribution.

- Temporarily suspend the balancing of shards to prevent immediate shard placement when your container servers are starting.
  Suspending the balancing of shards prevents uneven shard placement. Before you start your container servers, use the **xscmd -c suspendBalancing** command to stop the balancing of shards for a specific data grid and map set. After the container servers are started, you can use the **xscmd -c resumeBalancing** command to begin the placement of shards on the container servers.

- Suspend or resume placement and heartbeating. A placement bottleneck occurs when the catalog service runs through the balance algorithm on each container start. In this case, you can suspend placement, start all the containers, and then resume placement so that the balance algorithm is only run once after all the containers are started. You can also suspend heartbeating to prevent consistency issues when the high availability manager core group is formed. When many container servers are started at the same time, the catalog server sends out a new defined list of container servers to a high availability core group. At the same time, the container servers are also trying to create the current view of the high availability core group. Since there is a chance for inconsistencies in the catalog server's list of core group members versus the container's list of core group members, it is optimal to have the catalog process reports from the containers on core group health only after all the containers are started. The result is that there is a final defined set for each corresponding core group, and the containers can form a high availability manager view around those core groups. As such, there is benefit to ignoring high availability manager core group events when servers are started concurrently. WebSphere® eXtreme Scale provides an administrative option that is called suspend heartbeating, which ignores high availability manager core group events. When you suspend heartbeating, the catalog server ignores any failure detection reports from the high availability catalog service. You can also resume heartbeating, in which the default behavior returns and container failures are detected and reported by the high availability catalog service. When heartbeat is resumed, it also includes any containers that failed to start while heartbeating was suspended. Those failures are still detected and acted upon after heartbeating is resumed. For more information, see Tuning the heartbeat interval setting for failover detection.
  1. Use the **xscmd -c suspend** command to stop heartbeating and placement for a shard container or a domain.
     The following example illustrates how heartbeating and placement for a domain named `E13896FA-6141-4435-E000-000C2962AA71` is suspended using this command. The domain contains three data grids **myOG3**, **myOG2**, and **myOG1** and a mapset named **myMapSet**.

```
Console> xscmd -c suspend

[Tue Oct 22 2013 17:38:36] xscmd starting...
Starting at: 2013-10-22 17:38:38.012

CWXSI0068I: Executing command: suspend

Type      Suspendable Object Name Status Details
--------- ------------------ ---- ------ -------
heartbeat Domain name E13896FA-6141-4435-E000-000C2962AA71 Heartbeating was suspended.
placement Grid Name/Map Set Name myOG3/myMapSet Balancing was suspended.
placement Grid Name/Map Set Name myOG1/myMapSet Balancing was suspended.
placement Grid Name/Map Set Name myOG2/myMapSet Balancing was suspended.
```

```
CWXSI0040I: The suspend command completed successfully.
```

When heartbeating and placement is suspended, the following placement actions can still run:
- Shard promotion can occur when container servers fail.
- Shard role swapping with the **xscmd -c swapShardWithPrimary** command.
- Shard placement triggered balancing with the **xscmd -c triggerPlacement -g myOG -ms myMapSet** command, where *myOG* and *myMapSet* are set to values for your data grid and map set.

2. Start the container servers.
3. Use the **xscmd -c resume -t placement** command to resume placement. The following example illustrates how placement is resumed for the data grids **myOG3**, **myOG1**, and **myOG2** and the map set **myMapSet**.

```
Console> xscmd -c resume -t placement

[Tue Oct 22 2013 17:40:28] xscmd starting...
Starting at: 2013-10-22 17:40:29.509

CWXSI0068I: Executing command: resume

Type Suspendable Object Name Status Details
---- ------------------ ---- ------ -------
placement Grid Name/Map Set Name myOG3/myMapSet Balancing has resumed.
placement Grid Name/Map Set Name myOG1/myMapSet Balancing has resumed.
placement Grid Name/Map Set Name myOG2/myMapSet Balancing has resumed.

CWXSI0040I: The resume command completed successfully.
Ending at: 2013-10-22 17:40:34.371
```

Important: After placement is resumed, use the **xscmd -c showPlacement** command until all partitions show up as placed.

4. Optional: Use the **xscmd -c balanceShardTypes** command to adjust the ratio of primary and replica shards to be equitable among the running container servers in the configuration. The ratio is consistent within one shard on each container server.
5. Use the **xscmd -c resume -t heartbeat** command to resume heartbeating after placement has finished. The following example illustrates how heartbeating is resumed for the domain `E13896FA-6141-4435-E000-000C2962AA71`.

```
Console> xscmd -c resume -t heartbeat

[Tue Oct 22 2013 17:42:59] xscmd starting...
Starting at: 2013-10-22 17:42:59.872

CWXSI0068I: Executing command: resume

Type Suspendable Object Name Status Details
---- ------------------ ---- ------ -------
heartbeat Domain name E13896FA-6141-4435-E000-000C2962AA71 Balancing has resumed.

CWXSI0040I: The resume command completed successfully.
Ending at: 2013-10-22 17:43:04.549
```

- Configure the placementDeferralInterval property to minimize the number of shard placement cycles on the container servers. Shard placement is triggered at the defined time interval.

  placementDeferralInterval
  : Specifies the interval in milliseconds for deferring the balancing and placement of shards on the container servers. Placement does not start until after the time specified in the property has passed. Increasing the deferral interval lowers processor utilization, but the placement of work items is completed over time. A decrease in the deferral interval increases short-term processor usage, but the placement of work items is more immediate and expedited.
    If multiple container servers are starting in succession, the deferral interval timer is reset if a new container server starts within the given interval. For example, if a second container server starts 10 seconds after the first container server, placement does not start until 15 seconds after the second container server started. However, if a third container server starts 20 seconds after the second container server, placement has already begun on the first two container servers.

    When container servers become unavailable, placement is triggered as soon as the catalog server learns of the event so that recovery can occur as quickly as possible.

    Default: 15000 ms (15 seconds)

  You can use the following tips to help determine if your placement deferral value is set to the right amount of time:
  - As you concurrently start the container servers, look at the CWOBJ1001 messages in the SystemOut.log file for each container server. The timestamp of these messages in each container server log file indicates the actual container server start time. You might consider adjusting the placementDeferralInterval property to include more container server starts. For example, if the first container server starts 90 seconds before the last container server, you might set the property to 90 seconds.
  - Note how long the CWOBJ1511 messages occur after the CWOBJ1001 messages. This amount of time can indicate if the deferral has occurred successfully.
  - If you are using a development environment, consider the length of the interval when you are testing your application.
- Configure the numInitialContainers attribute.
  If you previously used the numInitialContainers attribute, you can continue using the attribute. However, the use of the **xscmd -c suspendBalancing** and **xscmd -c resumeBalancing** commands followed by the placementDeferralInterval are suggested over the numInitialContainers attribute to control placement. The numInitialContainers attribute specifies the number of container servers that are required before initial placement occurs for the shards in this mapSet element. The numInitialContainers attribute is in the deployment policy descriptor XML file. If you have both numInitialContainers and placementDeferralInterval set, note that until the numInitialContainers value has been met, no placement occurs, regardless of the value of the placementDeferralInterval property.

Controlling placement after initial startup

- Force placement to occur.
  You can use the **xscmd -c triggerPlacement -g myOG -ms myMapSet** command, where *myOG* and *myMapSet* are set to values for your data grid and map set, to force placement to occur during a point in time at which placement might not occur otherwise. For example, you might run this command when the amount of time specified by the placementDeferralInterval property has not yet passed or when balancing is suspended.
- Reassign a primary shard.
  Use the **xscmd -c swapShardWithPrimary** command to assign a replica shard to be the new primary shard. The previous primary shard becomes a replica.
- Rebalance the primary and replica shards.
  Use the **xscmd -c balanceShardTypes** command to adjust the ratio of primary and replica shards to be equitable among the running container servers in the configuration. The ratio is consistent within one shard on each container server.
- Suspend or resume placement.
  Use the **xscmd -c suspendBalancing** command or the **xscmd -c resumeBalancing** command to stop and start the balancing of shards for a specific data grid and map set. When balancing has been suspended, the following placement actions can still run:
  - Shard promotion can occur when container servers fail.
  - Shard role swapping with the **xscmd -c swapShardWithPrimary** command.
  - Shard placement triggered balancing with the **xscmd -c triggerPlacement -g myOG -ms myMapSet** command, where *myOG* and *myMapSet* are set to values for your data grid and map set.

What to do next

You can monitor the placement in the environment with the **xscmd -c placementServiceStatus** command.
Related concepts:
Installation topologies
Catalog service
Container servers, partitions, and shards
Related tasks:
Configuring the catalog service in WebSphere Application Server
Configuring WebSphere Application Server applications to automatically start container servers
Configuring container servers in WebSphere Application Server
Administering with the xscmd utility
Starting stand-alone servers
Related reference:
Deployment policy descriptor XML file
Configuring distributed deployments
ObjectGrid descriptor XML file
Server properties file
Related information:
Interface PlacementServiceMBean

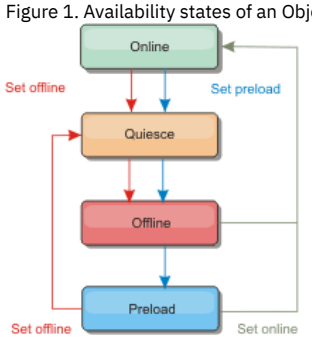
Managing ObjectGrid availability

The availability state of an ObjectGrid instance determines which requests can be processed at any particular time. You can use the StateManager interface to set and retrieve the state of an ObjectGrid instance.

About this task

Four availability states exist for a given ObjectGrid instance.

Figure 1. Availability states of an ObjectGrid instance



ONLINE
: The ONLINE state is the default availability state for an ObjectGrid. An ONLINE ObjectGrid is able to process any requests from a typical eXtreme Scale client. However, requests from a preload client are rejected while the ObjectGrid is ONLINE.

QUIESCE
: The QUIESCE state is transitional. An ObjectGrid that is in QUIESCE is soon moved to the OFFLINE state. While in the QUIESCE state, an ObjectGrid is allowed to process outstanding transactions. However, any new transactions are rejected. An ObjectGrid can remain in QUIESCE for up to 30 seconds. After this time, the availability state is moved to OFFLINE.

OFFLINE
: The OFFLINE state results in the rejection of all transactions that are sent to the ObjectGrid.

PRELOAD
: The PRELOAD state can be used to load data into an ObjectGrid from a preload client. While the ObjectGrid is in the PRELOAD state, only a preload client can commit transactions against the ObjectGrid. All other transactions are rejected.

A request is rejected if an ObjectGrid is not in the appropriate availability state to support that request. An AvailabilityException exception results whenever a request is rejected.

Procedure

1. Set the initial state of an ObjectGrid with the ObjectGrid configuration XML file.
   You can use the initialState attribute on an ObjectGrid to indicate its startup state. Normally, when an ObjectGrid completes initialization, it is available for routing. The state can later be changed to prevent traffic from routing to an ObjectGrid. If the ObjectGrid needs to be initialized, but not immediately available, you can use the initialState attribute.

   The initialState attribute is set on the ObjectGrid configuration XML file. The default state is ONLINE. Valid values include:
   - ONLINE (default)
   - PRELOAD
   - OFFLINE

   See ObjectGrid descriptor XML file for more information about the initialState attribute.
   If the initialState attribute is set on an ObjectGrid, the state must be explicitly set back to online or the ObjectGrid will remain unavailable. An AvailabilityException exception results if the ObjectGrid is not in the ONLINE state.

   See AvailabilityState API documentation for more information.

   Using the initialState attribute for preloading

   If the ObjectGrid is preloaded with data, there can be a period of time between when the ObjectGrid is available and switching to a preload state to block client traffic. To avoid this time period, the initial state on an ObjectGrid can be set to PRELOAD. The ObjectGrid still completes all the necessary initialization, but it blocks traffic until the state has changed and allows the preload to occur.

   The PRELOAD and OFFLINE states both block traffic, but you must use the PRELOAD state if you want to initiate a preload.

   Failover and balancing behavior

   If a replica data grid is promoted to be a primary data grid, the replica does not use the initialState setting. If the primary data grid is moved for a rebalance, the initialState setting is not used because the data is copied to the new primary location before the move is completed. If replication is not configured, then the primary goes into the initialState setting if failover occurs, and a new primary must be placed.

2. Change the availability state with the StateManager interface.
   Use the StateManager interface to set the availability state of an ObjectGrid. To set the availability state of an ObjectGrid running on the servers, pass a corresponding ObjectGrid client to the StateManager interface. The following code demonstrates how to change the availability state of an ObjectGrid.

   ```
   ClientClusterContext client = ogManager.connect("localhost:2809", null, null);
   ObjectGrid myObjectGrid = ogManager.getObjectGrid(client, "myObjectGrid");
   StateManager stateManager = StateManagerFactory.getStateManager();
   stateManager.setObjectGridState(AvailabilityState.OFFLINE, myObjectGrid);
   ```

   Each shard of the ObjectGrid transitions to the desired state when the setObjectGridState method is called on the StateManager interface. When the method returns, all shards within the ObjectGrid should be in the proper state.
   Use an ObjectGridEventListener plug-in to change the availability state of a server side ObjectGrid. Only change the availability state of a server-side ObjectGrid when the ObjectGrid has a single partition. If the ObjectGrid has multiple partitions, the shardActivated method is called on each primary, which results in superfluous calls to change the state of the ObjectGrid

   ```
   public class OGListener implements ObjectGridEventListener,
                           ObjectGridEventGroup.ShardEvents {
       public void shardActivated(ObjectGrid grid) {
               StateManager stateManager = StateManagerFactory.getStateManager();
               stateManager.setObjectGridState(AvailabilityState.PRELOAD, grid);
       }
   }
   ```

   Because QUIESCE is a transitional state, you cannot use the StateManager interface to put an ObjectGrid into the QUIESCE state. An ObjectGrid passes through this state on its way to the OFFLINE state.

3. Retrieve the availability state.
   Use the getObjectGridState method of the StateManager interface to retrieve the availability state of a particular ObjectGrid.

   ```
   StateManager stateManager = StateManagerFactory.getStateManager();
   AvailabilityState state = stateManager.getObjectGridState(inventoryGrid);
   ```

   The getObjectGridState method chooses a random primary within the ObjectGrid and returns its AvailabilityState. Because all shards of an ObjectGrid should be in the same availability state or transitioning to the same availability state, this method provides an acceptable result for the current availability state of the ObjectGrid.

Related tasks:
Administering OSGi-enabled services using the xscmd utility
Updating OSGi services for eXtreme Scale plug-ins with xscmd
Related reference:
Plug-ins for providing event listeners

Managing data center failures when quorum is enabled

When the data center enters a failure scenario, consider overriding quorum so that the container server life cycle events are not ignored based on your analysis of the failure scenario. You can use the **xscmd** utility to query about and run quorum tasks, such as the quorum status and overriding quorum.

Before you begin

- Configure the quorum mechanism to be the same setting in all of your catalog servers. See Configuring the quorum mechanism for more information.
- Quorum is the minimum number of catalog servers that are necessary to conduct placement operations for the data grid and is the full set of catalog servers, unless you configure a lower number. WebSphere® eXtreme Scale expects to lose quorum for the following reasons:
  - Catalog service JVM member fails
  - Network brown out
  - Data center loss
  - Garbage collection
  - Disk I/O
  - Severe swapping

  The following message indicates that quorum has been lost. Look for this message in your catalog service logs.

  ```
  CWOBJ1254W: The catalog service is waiting for quorum.
  ```

About this task

Override quorum in a data center failure scenario only. When you override quorum, any surviving catalog server instance can be used. All survivors are notified when one is told to override quorum.
To resolve these communication issues, you search the logs for your catalog servers for certain messages to determine and fix the problem. You also must look at your environment to determine if any existing issues need to be resolved. To find the message in the logs, you can either search the SystemOut*.log files for each catalog server, or you can use the message center to filter the logs for all the connected catalog servers.

Procedure

1. Query quorum status with the **xscmd** utility.

   ```
   xscmd -c showQuorumStatus -cep cathost:2809
   ```

   Use this option to display the quorum status of a catalog service instance. The command displays the current quorum status of each catalog server. The quorum column displays one of the following outcomes:
   - `TRUE`: The server has quorum enabled and the system is working normally. Quorum is met.
   - `FALSE`: The server has quorum enabled, but quorum is lost. The catalog servers do not allow changes to the catalog service domain.
   - `UNAVAILABLE`: The server cannot be contacted. It is either not running or there is a network problem and the server cannot be reached.
   - `DISABLED`: The server does not have quorum enabled.
2. Remove catalog servers that are having heartbeating failures.
   a. Examine the log files for each catalog server. If you see the following message in multiple catalog server logs, multiple catalog servers have declared themselves as the primary catalog server.

      ```
      CWOBJ8106I: The master catalog service cluster activated with cluster {0}
      ```

   b. Declare one primary catalog server by manually ending the processes for the other catalog servers.
      Manually stop the processes that are associated with the primary catalog servers that you have chosen not to use. End the processes with the command that is appropriate for your operating system, with a command such as the `kill` command.

   c. On the servers where you stopped catalog server processes, resolve any garbage collection, operating system, hardware, or networking issues. Garbage collection information is in the file where the JVM stores the garbage collection information.
3. Override quorum with the **xscmd** utility.

   ```
   xscmd -c overrideQuorum -cep hostname:port
   ```

   Running this command forces the surviving catalog servers to re-establish a quorum.
4. Run the **xscmd -c triggerPlacement** command. Running this command initiates failure recovery so that the remaining system can service requests.
5. Validate that recovery was successful.
   a. Run the **xscmd -c showPlacement** command every 15 seconds for a minute. Confirm placement is stable and that no changes are occurring.
   b. Run the **xscmd -c routetable** command. This command displays the current route table by simulating a new client connection to the data grid. It also validates the route table by confirming that all container servers are recognizing their role in the route table, such as which type of shard for which partition.

      ```
      xscmd -c routetable -cep hostname:port -g myGrid
      ```

   c. Run the **xscmd -c showMapSizes** command to track that data is flowing to the data grid as expected. Verify that key distribution is uniform over the shards in the key. If some container servers have more keys than others, then it is likely the hash function on the key objects has a poor distribution.

      ```
      xscmd -c showMapSizes -cep hostname:port -g myGrid -ms myMapSet
      ```

   d. Run the **xscmd -c revisions** command. If any revisions come back in the list, the primary and replica pairs are not completely replicated. Depending on your load, incomplete replication is okay. However, if you see a difference between the revision numbers of a primary and replica growing over time, then replication might not be working or is struggling to keep up with your load. Run this command multiple times to watch for trends.
   e. Run the **xscmd -c listCoreGroups** command to display a list of all the core groups for the catalog server.

```
xscmd -c listCoreGroups -cep hostname:port
```

Related concepts:
Catalog server quorums
Catalog service
Related tasks:
Managing data center failures
Configuring the quorum mechanism
Administering with the xscmd utility
8.5+

Querying and invalidating data

8.5+ You can use the query interfaces in the monitoring console and in the **xscmd** utility to retrieve small sets of keys from a map and invalidate sets of data.

Before you begin

- If you are using the web console to query and invalidate data, configure the monitoring console first. For more information, see Monitoring with the web console.
- If you are using **xscmd** to query and invalidate data, set up the **xscmd** utility. For more information, see Administering with the xscmd utility.

About this task

You can use the console or the **xscmd** utility to query data grid contents. You can query the data by running a regular expression on the data key. You can then use the same query to invalidate data. For examples of regular expressions, see Regular expression syntax.

Procedure

- Query or invalidate data with the console.
  1. Go to the query page in the console. In the web console, click Management > Query Data Grid Contents. Choose the map on which you want to filter.
  2. Search or filter the data in the map. You can use one of the following options to search or filter the data:
     - Type a regular expression in the field and click the Search button (). A list of keys that match the regular expression displays. The list of data could be a subset of all of the matching data.
     - To filter the results on a set of partitions, click the Filter button (). You can then type a regular expression and choose a range of partitions on which you want to filter the results.
  3. Invalidate data. When you invalidate the data, the data is permanently removed from the data grid.

     Selected keys
     : You can select keys from the table to invalidate. You can either click entries individually or click the select all check box, which selects a maximum of 500 entries that are in the table. When you have the entries selected that you want to remove, click Invalidate > Selected keys.

     All keys matching query
     : You can also invalidate all the data that matches your regular expression. Using this option deletes all data in the data grid that matches the regular expression, not just the maximum of 500 entries that is displayed in the console. To invalidate entries with the selected regular expression, click Invalidate > All keys matching query.

- Query or invalidate data with the **xscmd** utility.

  Query data:

  ```
  xscmd.sh -c findbykey -g <data_grid> -m <map>
  -fs <find_string> [-fp <partitionid>]
  ```

  You must include the data grid, map, and regular expression for the find string value. You can also filter on the partition ID. The result returns a subset of the entire query.

  Invalidate data:
  : Include the -inv argument in the command to invalidate the data that is selected by the query.

  ```
  xscmd -c findbykey -g <data_grid> -m <map>
  -fs <find_string> [-fp <partitionid>] -inv
  ```

  You must include the data grid, map, and regular expression for the find string value. You can also filter on the partition ID. When you run the invalidation, all matching values are invalidated, not just the small set that is returned by the query.

UNIX Linux Important: If your regular expression starts with the characters `.*`, the characters might not process correctly when you run the command. To resolve this issue, format your regular expression in one of the following ways:
- Enclose your regular expression with apostrophe characters: `-fs '.*'`
- Use a backslash to escape the asterisk character: `-fs .\*`

Example:
The following example looks for all entries in the `Grid` data grid and `Map1` map.

```
xscmd -c findbykey -g Grid -m Map1 -fs ".*"
```

The command returns the following results:

```
3 matching keys were found.

Partition Key
--------- ---
2         keyghi
4         keydef
6         keyabc
```

Related concepts:
Data invalidation
JMS event listener
Related reference:
ObjectGridEventListener plug-in
Introduction to ObjectMap
8.5+ xscmd utility reference
8.5+ Regular expression syntax
java.utils.regex.Pattern API documentation
Related information:
ObjectMap.invalidate method
EntityManager.invalidate method
ObjectGridEventListener interface
8.5+ http://docs.oracle.com/javase/1.4.2/docs/

Retrieving eXtreme Scale environment information with the xscmd utility

8.5+ You can use the **xscmd** utility with the -c showinfo command to view important details about the servers running in your WebSphere® eXtreme Scale environment, including: WebSphere eXtreme Scale servers, Java™ virtual machines, and (if applicable) servers running with WebSphere Application Server. Issue this command to retrieve name and version information, hostname and IP address, and the installation directories of these servers. Using -c showinfo command lets you retrieve these details without having to check log files, directories, or use third party applications.

Procedure

- Ensure that at least one of your catalog servers is running. If you want to retrieve environment details for your entire eXtreme Scale domain, then ensure that all of your servers are running.
  To retrieve environment information for your entire eXtreme Scale domain, issue: Windows

  ```
  xscmd.bat -c showinfo
  ```

  UNIX

  ```
  ./xscmd.sh -c showinfo
  ```

  The command returns all information about the servers running in your environment.
- To retrieve information about a specific server, use the -s parameter and specify the name of the server.
  Windows

  ```
  xscmd.bat -c showinfo -s <server_name>
  ```

  UNIX

  ```
  ./xscmd.sh -c showinfo -s <server_name>
  ```

- To view a list of servers, use the -sl parameter.
  Windows

  ```
  xscmd.bat -c showinfo -sl  <server_name>[,<server_name>]
  ```

  UNIX

  ```
  ./xscmd.sh -c showinfo -sl  <server_name>[,<server_name>]
  ```

- To retrieve environment information for a specific set of servers running on a particular host, use the -hf parameter and provide the name of the host.
  Windows

  ```
  xscmd.bat -c showinfo -hf <host_name>
  ```

  UNIX

  ```
  ./xscmd.sh -c showinfo -hf <host_name>
  ```

Related concepts:
Configuration considerations for multi-master topologies
8.5+ Installing fix packs using IBM Installation Manager
Related tasks:
Updating eXtreme Scale servers
Configuring multiple data center topologies
Migrating to WebSphere eXtreme ScaleVersion 8.5
Starting and stopping stand-alone servers
Starting and stopping servers in a WebSphere Application Server environment
Related reference:
8.5+ xscmd utility reference

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:

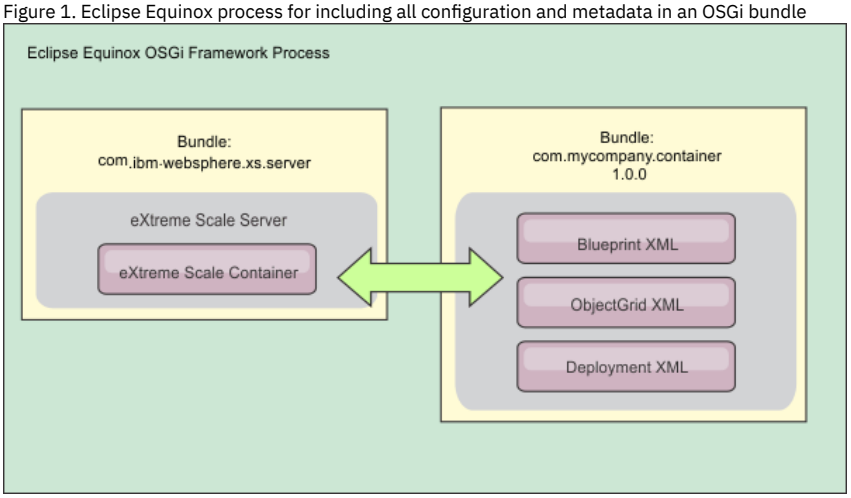
1. The WebSphere eXtreme Scale server bundle must be installed into Eclipse Equinox.
2. Your application must be packaged as an OSGi bundle.
3. Your WebSphere eXtreme Scale plug-ins (if any) must be packaged as an OSGi bundle. They can be bundled in the same bundle as your application or as separate 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:

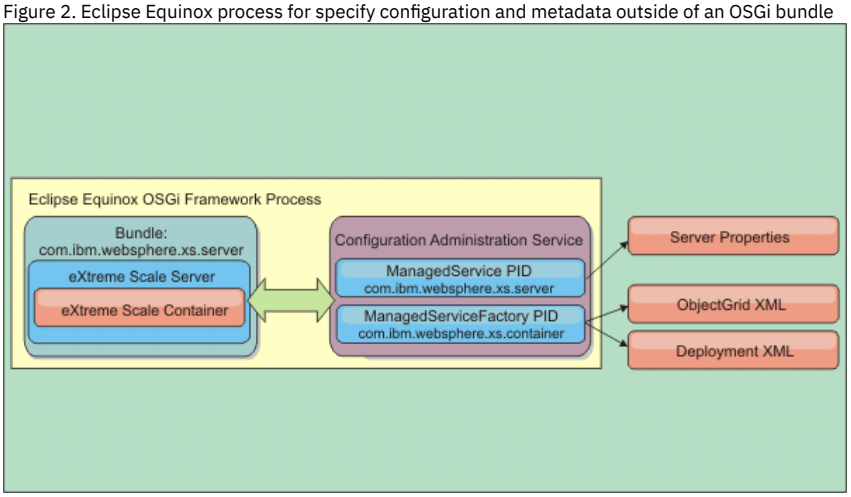
- OSGi Blueprint service
  You can include all configuration and metadata in an OSGi bundle. See the following image to understand the Eclipse Equinox process for this method:

  Figure 1. 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 2. 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.
  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.

Related tasks:
Running eXtreme Scale containers with non-dynamic plug-ins in an OSGi environment

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

Complete the following tasks before the OSGi-enabled plug-ins are installed and started.

- The eXtreme Scale server or client bundle is installed into the Eclipse Equinox OSGi framework. See Installing eXtreme Scale bundles.
- One or more dynamic BackingMap or ObjectGrid plug-ins are implemented. See Building eXtreme Scale dynamic plug-ins.
- The dynamic plug-ins are packaged as OSGi services in OSGi bundles.
- By default, the JVM continues to run when each eXtreme Scale server in an OSGi framework is stopped in the **xscmd** utility with the **-c teardown** command. If you want eXtreme Scale to exit the JVM after each server is stopped, then set the server property exitJVMOnTeardown to `true`. For more information, see Server properties file.

About this task

Install the bundle with the Eclipse Equinox console. There are several different methods to install the bundle, including a modification of the config.ini configuration file. Products that embed Eclipse Equinox include alternative methods for adding bundles in the config.ini file. For more information, seeEclipse runtime options.

OSGi allows bundles to be started that have duplicate services. WebSphere eXtreme Scale uses the latest service ranking. When multiple OSGi frameworks are started 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, see Updating OSGi services for eXtreme Scale plug-ins with xscmd.

Procedure

Install the plug-in bundle into the Eclipse Equinox OSGi framework with the OSGi console.

1. Start the Eclipse Equinox framework with the console enabled.

   ```
   <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 lists 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 started:

   ```
   osgi> ss
   ```

When the bundle starts, the bundle lists the ACTIVE state, for example:

```
17      ACTIVE      com.mycompany.plugin.bundle_VRM
```

Install the plug-in bundle into the Eclipse Equinox OSGi framework with 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 that 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 started. For example:

   ```
   osgi> ss
   ```

   When the bundle starts, the bundle lists 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.

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.

- Updating OSGi services for eXtreme Scale plug-ins with xscmd
  WebSphere eXtreme Scale supports upgrading container server plug-in bundles while the grid is active. This support allows administrators to complete application updates and additions without needing to restart grid processes.

Related tasks:
Updating OSGi services for eXtreme Scale plug-ins with xscmd
Administering with the xscmd utility
Managing ObjectGrid availability
Related reference:
Plug-ins for providing event listeners
Related information:
Eclipse runtime options

Updating OSGi services for eXtreme Scale plug-ins with xscmd

WebSphere® eXtreme Scale supports upgrading container server plug-in bundles while the grid is active. This support allows administrators to complete application updates and additions without needing to restart grid processes.

Before you begin

Complete the following steps before you update eXtreme Scale OSGi bundles to a new version:

1. Start eXtreme Scale servers in a supported OSGi framework.
2. Separate all eXtreme Scale plug-ins into bundles, and they must use service rankings to identify each version of the plug-ins.
3. Specify cache objects as either Java™ primitive types such as `byte[]`, Integer or String, or they must be stored using a MapSerializerPlugin plug-in The data objects are stored in the eXtreme Scale bundle and are not upgraded. Only the plug-ins that interact with the data are updated.
4. Design cache object data to be version compatible. New plug-ins must be able to interact with data created by older plug-ins.
5. Design plug-ins to listen for ObjectGridLifecycle and BackingMapLifecycle events to refresh any references to other plug-ins or the metadata that the plug-ins might have so that they can be refreshed when it is updated.
6. The eXtreme Scale OSGi update process only affects servers. You must independently update any clients that are using plug-ins.

About this task

Without OSGi enablement, if an administrator needs to update the application plug-ins or cache objects, each grid node must be upgraded one-by-one, causing stress on the network, memory and cpu utilization. This is required since plug-ins and cache Java objects are directly stored in the grid. When classes are updated without restarting the processes, the grid plug-ins have conflicts because each class has a different ClassLoader.

The eXtreme Scale product includes the xscmd utility and MBeans that allows administrators to view all the plug-in bundles installed in each grid container's hosting OSGi framework and choose which revision to use. When the xscmd is used to update the plug-ins to a new ranking, the grid is quiesced and all transactions are drained, the plug-ins are updated, and the grid is activated again. If an error occurs during the update process, the process is rolled-back and the old ranking is restored.

Procedure

1. Create a version of the bundle, increasing the version number in the bundle manifest, and increasing the ranking for each eXtreme Scale plug-in service. If the original bundle version is `Bundle-Version: 1.0.0`, then the next version can be defined as `Bundle-Version: 1.1.0`.
   If the original service ranking is `ranking="1"`, then the next ranking can be defined as `ranking="2"`.

   Important: OSGi service rankings must be integers.
2. Copy the new bundle to each OSGi framework node that is hosting an eXtreme Scale container server.
3. Install the new bundle into the OSGi framework. The bundle is assigned a bundle identifier; for example:

   ```
   osgi> install <URL to bundle>
   ```

4. Start the new bundle using the assigned bundle identifier; for example:

   ```
   osgi> start <id>
   ```

   After the new bundle is started, the eXtreme Scale OSGi service tracker detects the bundle and makes it available for updating.

5. Use the **xscmd -c osgiAll** command to verify that each container server sees the new bundle. The **osgiAll** command queries all containers in the grid for all services that are referenced in the ObjectGrid descriptor XML file and displays all rankings that are available; for example:

```
xscmd -c osgiAll

Server: server1
   OSGi Service Name          Available Rankings
   -----------------          ------------------
   myLoaderServiceFactory     1, 2
   mySerializerServiceFactory 1, 2

Server: server2
   OSGi Service Name          Available Rankings
   -----------------          ------------------
   myLoaderServiceFactory     1, 2
   mySerializerServiceFactory 1, 2

Summary - All servers have the same service rankings.
```

6. Use the **xscmd -c osgiCheck** command to verify that one or more service rankings are valid update targets; for example:

```
xscmd -c osgiCheck -sr
mySerializerServiceFactory;2,myLoaderServiceFactory;2

CWXSI0040I: The command osgiCheck has completed successfully.
```

7. If the **osgiCheck** command did not find any resulting errors, suspend the balancer of the placement service to avoid shard movements, in case of a failure during the update process. To suspend placement, use the **xscmd -c suspendBalancing** command for each object grid and map set that are affected by the update; for example:

```
xscmd -c suspendBalancing -g MyGrid -ms MyMapSet
```

8. After balancing has been suspended for each object grid and map set, use the **xscmd -c osgiCheck** command again to verify that one or more service rankings are valid update targets; for example:

```
xscmd -c osgiCheck -sr
mySerializerServiceFactory;2,myLoaderServiceFactory;2

CWXSI0040I: The command osgiCheck has completed successfully.
```

9. After balancing has been suspended for the object grid and map set, use the **osgiUpdate** command to update the service on all of the servers for an object grid and map set; for example:

```
xscmd -c osgiUpdate  -sr
mySerializerServiceFactory;2,myLoaderServiceFactory;2 -g MyGrid -ms MyMapSet
```

10. Verify that the upgrade succeeded; for example:

```
Update succeeded for the following service rankings:
Service                    Ranking
-------                    -------
mySerializerServiceFactory  2
myLoaderServiceFactory      2
```

11. After you verify that the ranking has been updated successfully, enable balancing again, using the **xscmd -c resumeBalancing** command; for example:

```
xscmd -c resumeBalancing -g MyGrid -ms MyMapSet
```

12. Stop and uninstall the old bundle in each OSGi framework that is hosting the eXtreme Scale container. For example, enter the following code in the Eclipse Equinox console:

```
osgi> stop <id>
osgi> uninstall <id>
```

Results

The eXtreme Scale bundle has been updated to a new version.
Related concepts:
OSGi framework overview
Related tasks:
Programming to use the OSGi framework
Installing the Eclipse Equinox OSGi framework with Eclipse Gemini for clients and servers
Administering OSGi-enabled services using the xscmd utility
Administering with the xscmd utility
Managing ObjectGrid availability
Related reference:
Plug-ins for providing event listeners
Related information:
Eclipse runtime options

Administering with Managed Beans (MBeans)

You can use several different types of Java™ Management Extensions (JMX) MBeans to administer and monitor deployments. Each MBean refers to a specific entity, such as a map, data grid, server, or service.

JMX MBean interfaces and WebSphere eXtreme Scale

Each MBean has get methods that represent attribute values. These get methods cannot be called directly from your program. The JMX specification treats attributes differently from operations. You can view attributes with a vendor JMX console, and you can perform operations in your program or with a vendor JMX console.

Package com.ibm.websphere.objectgrid.management

See the API documentation for an overview and detailed programming specifications for all of the available MBeans:Package com.ibm.websphere.objectgrid.management .

- Accessing Managed Beans (MBeans) using the wsadmin tool
  You can use the wsadmin utility provided in WebSphere® Application Server to access managed bean (MBean) information.
- Accessing Managed Beans (MBeans) programmatically
  You can connect to MBeans with Java applications. These applications use the interfaces in the com.ibm.websphere.objectgrid.management package.

Related concepts:
Statistics overview
Related tasks:
Accessing Managed Beans (MBeans) using the wsadmin tool
Accessing Managed Beans (MBeans) programmatically
Monitoring server statistics with managed beans (MBeans)
Monitoring with the xscmd utility
Related information:
API documentation: Package com.ibm.websphere.objectgrid.management
Interface PlacementServiceMBean

Accessing Managed Beans (MBeans) using the wsadmin tool

You can use the wsadmin utility provided in WebSphere® Application Server to access managed bean (MBean) information.

Procedure

Run the wsadmin tool from the bin directory in your WebSphere Application Server installation. The following example retrieves a view of the current shard placement in a dynamic eXtreme Scale. You can run the wsadmin tool from any installation where eXtreme Scale is running. You do not have to run the wsadmin tool on the catalog service.

```
$ wsadmin.sh -lang jython
wsadmin>placementService = AdminControl.queryNames
        ("com.ibm.websphere.objectgrid:*,type=PlacementService")
wsadmin>print AdminControl.invoke(placementService,
        "listObjectGridPlacement","library ms1")

<objectGrid name="library" mapSetName="ms1">
  <container name="container-0" zoneName="DefaultDomain"
               hostName="host1.company.org" serverName="server1">
     <shard type="Primary" partitionName="0"/>
     <shard type="SynchronousReplica" partitionName="1"/>
  </container>
  <container name="container-1" zoneName="DefaultDomain"
               hostName="host2.company.org" serverName="server2">
     <shard type="SynchronousReplica" partitionName="0"/>
     <shard type="Primary" partitionName="1"/>
  </container>
  <container name="UNASSIGNED" zoneName="_ibm_SYSTEM"
               hostName="UNASSIGNED" serverName="UNNAMED">
    <shard type="SynchronousReplica" partitionName="0"/>
    <shard type="AsynchronousReplica" partitionName="0"/>
  </container>
</objectGrid>
```

Related concepts:
Statistics overview
Related tasks:
Accessing Managed Beans (MBeans) programmatically
Accessing Managed Beans (MBeans) programmatically
Monitoring server statistics with managed beans (MBeans)
Monitoring with the xscmd utility
Related reference:
Administering with Managed Beans (MBeans)
Related information:
API documentation: Package com.ibm.websphere.objectgrid.management
Interface PlacementServiceMBean

Accessing Managed Beans (MBeans) programmatically

You can connect to MBeans with Java applications. These applications use the interfaces in the com.ibm.websphere.objectgrid.management package.

About this task

Programmatic methods for accessing MBeans vary depending on the type of server to which you are connecting.

- Connect to a stand-alone catalog service MBean server
- Connect to a container MBean server
- Connect to a catalog service MBean server that is hosted in WebSphere® Application Server
- Connect to a catalog service MBean server with security enabled

Procedure

- **Connect to a stand-alone catalog service MBean server:**
  The following example program connects to a stand-alone catalog service MBean server and returns an XML formatted string that lists each container server along with its allocated shards for a given ObjectGrid and MapSet.

  Figure 1. CollectPlacementPlan.java

```
package com.ibm.websphere.sample.xs.admin;

import java.util.Set;

import javax.management.MBeanServerConnection;
import javax.management.ObjectName;
import javax.management.remote.JMXConnector;
import javax.management.remote.JMXConnectorFactory;
import javax.management.remote.JMXServiceURL;

/**
 * Collects the placement information from the Catalog Server for a given ObjectGrid.
 */
public final class CollectPlacementPlan {
    private static String hostName = "localhost";

    private static int port = 1099;

    private static String objectGridName = "library";

    private static String mapSetName = "ms1";

    /**
     * Connects to the ObjectGrid Catalog Service to retrieve placement information and
           * prints it out.
     *
     * @param args
     * @throws Exception
     *             If there is a problem connecting to the catalog service MBean server.
     */
    public static void main(String[] args) throws Exception {
        String serviceURL = "service:jmx:rmi:///jndi/rmi://" + hostName + ":" + port +
                    "/objectgrid/MBeanServer";
        JMXServiceURL jmxUrl = new JMXServiceURL(serviceURL);
        JMXConnector jmxCon = JMXConnectorFactory.connect(jmxUrl);

        try {
            MBeanServerConnection catalogServerConnection = jmxCon.getMBeanServerConnection();

            Set placementSet = catalogServerConnection.queryNames(new
                                                ObjectName("com.ibm.websphere.objectgrid"
             + ":*,type=PlacementService"), null);
            ObjectName placementService = (ObjectName) placementSet.iterator().next();
            Object placementXML = catalogServerConnection.invoke(placementService,
                "listObjectGridPlacement", new Object[] {
                objectGridName, mapSetName }, new String[] { String.class.getName(),
                                                        String.class.getName() });
            System.out.println(placementXML);
        } catch (Exception e) {
            if(jmxCon != null) {
                jmxCon.close();
            }
        }
    }

}
```

  A few notes regarding the sample program:
  - The JMXServiceURL value for the catalog service is always of the following form: `service:jmx:rmi:///jndi/rmi://<host>:<port>/objectgrid/MBeanServer`, where `<host>` is the host on which the catalog service is running and `<port>` is the JMX service port that is provided with the -JMXServicePort option when starting the catalog service. If no port is specified, the default is `1099`.

- For the ObjectGrid or map statistics to be enabled, you must specify the following property in the server properties file when you are starting an ObjectGrid container: `statsSpec=all=enabled`
- To disable the MBeans that are running in the container servers, specify the following property in the server properties file: `enableMBeans=false`.

An example of the output follows. This output indicates that two container servers are active. The `Container-0` container server hosts four primary shards. The `Container-1` container server hosts a synchronous replica for each of the primary shards on the `Container-0` container server. In this configuration, two synchronous replicas and one asynchronous replica are configured. As a result, the `Unassigned` container server is left with the remaining shards. If two more container servers are started, the `Unassigned` container server is not displayed.

```
<objectGrid name="library" mapSetName="ms1">
  <container name="Container-1" zoneName="DefaultZone"
    hostName="myhost.mycompany.com" serverName="ogserver">
    <shard type="SynchronousReplica" partitionName="0"/>
    <shard type="SynchronousReplica" partitionName="1"/>
    <shard type="SynchronousReplica" partitionName="2"/>
    <shard type="SynchronousReplica" partitionName="3"/>
  </container>
  <container name="Container-0" zoneName="DefaultZone"
    hostName="myhost.mycompany.com" serverName="ogserver">
    <shard type="Primary" partitionName="0"/>
    <shard type="Primary" partitionName="1"/>
    <shard type="Primary" partitionName="2"/>
    <shard type="Primary" partitionName="3"/>
  </container>
  <container name="library:ms1:_UnassignedContainer_" zoneName="_ibm_SYSTEM"
    hostName="UNASSIGNED" serverName="UNNAMED">
    <shard type="SynchronousReplica" partitionName="0"/>
    <shard type="SynchronousReplica" partitionName="1"/>
    <shard type="SynchronousReplica" partitionName="2"/>
    <shard type="SynchronousReplica" partitionName="3"/>
    <shard type="AsynchronousReplica" partitionName="0"/>
    <shard type="AsynchronousReplica" partitionName="1"/>
    <shard type="AsynchronousReplica" partitionName="2"/>
    <shard type="AsynchronousReplica" partitionName="3"/>
  </container>
</objectGrid>
```

- **Connect to a container MBean server:**
  Container servers host MBeans to query information about the individual maps and ObjectGrid instances that are running within the container server. The following example program prints the status of each container server that is hosted by the catalog server with the JMX address of `localhost:1099`:

  Figure 2. CollectContainerStatus.java

```
package com.ibm.websphere.sample.xs.admin;

import java.util.List;
import java.util.Set;

import javax.management.MBeanServerConnection;
import javax.management.ObjectInstance;
import javax.management.ObjectName;
import javax.management.remote.JMXConnector;
import javax.management.remote.JMXConnectorFactory;
import javax.management.remote.JMXServiceURL;

/**
 * Collects placement status from each of the available containers directly.
 */
public final class CollectContainerStatus {
    private static String hostName = "localhost";

    private static int port = 1099;

    /**
     * @param args
     */
    public static void main(String[] args) throws Exception {
        String serviceURL = "service:jmx:rmi:///jndi/rmi://" + hostName + ":" + port + "/objectgrid/MBeanServer";
        JMXServiceURL jmxUrl = new JMXServiceURL(serviceURL);
        JMXConnector jmxCon = JMXConnectorFactory.connect(jmxUrl);

        try {
            MBeanServerConnection catalogServerConnection = jmxCon.getMBeanServerConnection();

            Set placementSet = catalogServerConnection.queryNames(new ObjectName("com.ibm.websphere.objectgrid"
                    + ":*,type=PlacementService"), null);

            ObjectName placementService = (ObjectName) placementSet.iterator().next();
            List<String> containerJMXAddresses = (List<String>) catalogServerConnection.invoke(placementService,
                    "retrieveAllServersJMXAddresses", new Object[0], new String[0]);
            for (String address : containerJMXAddresses) {
                JMXServiceURL containerJMXURL = new JMXServiceURL(address);
                JMXConnector containerConnector = JMXConnectorFactory.connect(containerJMXURL);
                MBeanServerConnection containerConnection = containerConnector.getMBeanServerConnection();
                Set<ObjectInstance> containers = containerConnection.queryMBeans(
                        new ObjectName("*:*,type=ObjectGridContainer"), null);
                for (ObjectInstance container : containers) {
                    System.out.println(containerConnection.getAttribute(container.getObjectName(), "Status"));
                }
            }
```

```
        } finally {
            if(jmxCon != null) {
                jmxCon.close();
            }
        }
    }
}
```

The example program prints out the container server status for each container. An example of the output follows:

```
<container name="Container-0" zoneName="DefaultZone" hostName="descartes.rchland.ibm.com"
               serverName="ogserver">
  <shard type="Primary" partitionName="1"/>
  <shard type="Primary" partitionName="0"/>
  <shard type="Primary" partitionName="3"/>
  <shard type="Primary" partitionName="2"/>
</container>
```

- **Connect to a catalog service MBean server that is hosted in WebSphere Application Server:**
  The method for programmatically accessing MBeans in WebSphere Application Server is slightly different from accessing MBeans in a stand-alone configuration.

  1. Create and compile a Java program to connect to the MBean server. An example program follows:
     Figure 3. CollectPlacementPlan.java

     ```
     package com.ibm.websphere.sample.xs.admin;

     import java.util.Set;

     import javax.management.MBeanServerConnection;
     import javax.management.ObjectName;
     import javax.management.remote.JMXConnector;
     import javax.management.remote.JMXConnectorFactory;
     import javax.management.remote.JMXServiceURL;

     /**
      * Collects the placement information from the catalog server running in a deployment manager for a given
     ObjectGrid.
      */
     public final class CollectPlacementPlanWAS {
         private static String hostName = "localhost";

         private static int port = 9809;

         private static String objectGridName = "library";

         private static String mapSetName = "ms1";

         /**
          * Connects to the catalog service to retrieve placement information and prints it out.
          *
          * @param args
          * @throws Exception
          *             If there is a problem connecting to the catalog service MBean server.
          */
         public static void main(String[] args) throws Exception {

             // connect to bootstrap port of the deployment manager
             String serviceURL = "service:jmx:iiop://" + hostName + ":" + port + "/jndi/JMXConnector";
             JMXServiceURL jmxUrl = new JMXServiceURL(serviceURL);
             JMXConnector jmxCon = JMXConnectorFactory.connect(jmxUrl);

             try {
                 MBeanServerConnection catalogServerConnection = jmxCon.getMBeanServerConnection();

                 Set placementSet =
                                                     catalogServerConnection.queryNames(new
     ObjectName("com.ibm.websphere.objectgrid"
                      + ":*,type=PlacementService"), null);

                 ObjectName placementService = (ObjectName) placementSet.iterator().next();
                 Object placementXML = catalogServerConnection.invoke(placementService,
                             "listObjectGridPlacement", new Object[] {
                         objectGridName, mapSetName }, new String[] { String.class.getName(),
                                                                      String.class.getName() });
                 System.out.println(placementXML);
             } finally {
                 if(jmxCon != null) {
                     jmxCon.close();
                 }
             }
         }

     }
     ```

  2. Run the following command.

     ```
     "$JAVA_HOME/bin/java" "$WAS_LOGGING" -
     Djava.security.auth.login.config="$app_server_root/properties/wsjaas_client.conf" \
      -Djava.ext.dirs="$JAVA_HOME/jre/lib/ext:$WAS_EXT_DIRS:$WAS_HOME/plugins:$WAS_HOME/lib/WMQ/java/lib" \
     ```

```
-Djava.naming.provider.url=<an_IIOP_URL_or_a_corbaloc_URL_to_your_application_server_machine_name> \
-Djava.naming.factory.initial=com.ibm.websphere.naming.WsnInitialContextFactory \
-Dserver.root="$WAS_HOME" "$CLIENTSAS" "$CLIENTSSL" $USER_INSTALL_PROP \
-classpath "$WAS_CLASSPATH":<list_of_your_application_jars_and_classes> \
<fully_qualified_class_name_to_run> <your_application_parameters>
```

This command assumes that the *was_root*/bin/setupCmdLine.sh script has been run to set the variables properly. An example of the format of the java.naming.provider.url property value is `corbaloc:iiop:1.0@<host>:<port>/NameService`.

- **Connect to a catalog service MBean server with security enabled:**
  For more information about connecting to the catalog service MBean with security enabled, see Java Management Extensions (JMX) security.

What to do next

For more examples on how to display statistics and perform administrative operations with MBeans, see the **xsadmin** sample application. 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. See Sample: xsadmin utility for more information about the operations you can complete with the **xsAdmin** sample application.
You can also find more information about MBeans in the com.ibm.websphere.objectgrid.management package.

Related concepts:
Statistics overview
Related tasks:
Accessing Managed Beans (MBeans) using the wsadmin tool
Accessing Managed Beans (MBeans) using the wsadmin tool
Monitoring server statistics with managed beans (MBeans)
Monitoring with the xscmd utility
Related reference:
Administering with Managed Beans (MBeans)
Related information:
API documentation: Package com.ibm.websphere.objectgrid.management
Interface PlacementServiceMBean

Administering J2C client connections

8.5+ 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 XSConnectionFactory 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.

Previous topic: 8.5+ Developing eXtreme Scale client components to use transactions
Related information:
JCA lifecycle management
Object grid J2C connection MBean API documentation

Developing applications

	8.5 Develop applications that use the data grid. The tasks for developing applications include: • Accessing data • System APIs and plug-ins • JPA integration • Spring integration

- Setting up the development environment
  Before you begin developing applications, you must set up your development environment.
- Accessing data with client applications
  After you configure your development environment, you can begin to develop applications that create, access, and manage the data in your data grid.
- System APIs and plug-ins
  A plug-in is a component that provides a function to the pluggable components, which include ObjectGrid and BackingMap. To most effectively use eXtreme Scale as an in-memory data grid or database processing space, you should carefully determine how best you can maximize performance with available plug-ins.
- Programming to use the OSGi framework
  You can start eXtreme Scale servers and clients in an OSGi container, which allows you to dynamically add and update eXtreme Scale plug-ins to the runtime environment.
- Programming for JPA integration
  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.
- Developing applications with the Spring framework
  Learn how to integrate your eXtreme Scale applications with the popular Spring framework.

Setting up the development environment

Before you begin developing applications, you must set up your development environment.

Before you begin

See Planning to develop WebSphere eXtreme Scale applications for more information about the available programming interfaces and considerations.

- Accessing API documentation
  You can access the API documentation for WebSphere® eXtreme Scale by downloading a zip file archive, incorporating the API documentation into your development environment, or viewing the API documentation in the information center.
- Setting up a stand-alone development environment in Eclipse
  Use Eclipse-based integrated development environment to build and run a Java™ SE application with the stand-alone version of WebSphere eXtreme Scale.
- Running a WebSphere eXtreme Scale application that uses an application server other than WebSphere Application Server in Eclipse
  You can configure a Java EE application that uses WebSphere eXtreme Scale to run in an application server other than WebSphere Application Server in Eclipse.
- Running an integrated client or server application with WebSphere Application Server in Rational Application Developer
  Configure and run a Java EE application with a WebSphere eXtreme Scale client or server with the WebSphere Application Server runtime embedded in Rational® Application Developer.

Related concepts:
Java API overview
Java API overview
Related information:
API documentation
API documentation

Accessing API documentation

You can access the API documentation for WebSphere® eXtreme Scale by downloading a zip file archive, incorporating the API documentation into your development environment, or viewing the API documentation in the information center.

About this task

You can access API documentation in one of the following locations:

Information center
: Using the information center API documentation is useful for searching along with the rest of the WebSphere eXtreme Scale product information.

Zip file archive
: You can download this file for each release. You can then use compare tools to see what APIs changed from release to release. You can also directly link the compressed file in your Eclipse projects when you are compiling against the objectgrid.jar file. Using this linking integrates the API documentation in the IDE.

Procedure

- View API documentation in the information center. For more information, see API documentation.
- Download a zip archive of the API documentation.
  If you want to download the API documentation to browse offline, you can download a zip file for the appropriate release from the following page: IBM Elastic Caching Community wiki: API documentation downloads.

What to do next

For more information about accessing the API documentation within the development environment, see Setting up a stand-alone development environment in Eclipse.

Related concepts:
Java API overview
Java API overview
Related information:
API documentation
API documentation

Setting up a stand-alone development environment in Eclipse

Use Eclipse-based integrated development environment to build and run a Java™ SE application with the stand-alone version of WebSphere® eXtreme Scale.

Before you begin

- Install the WebSphere eXtreme Scale product into a new or empty directory and apply the latest WebSphere eXtreme Scale fix pack. For more information, see Installing.
- Download the API documentation. For more information, see IBM Elastic Caching Community wiki: API documentation downloads.

Procedure

- Configure Eclipse to build and run a Java SE application with WebSphere eXtreme Scale.
  1. Define a user library to allow your application to reference WebSphere eXtreme Scale application programming interfaces.
     a. In your Eclipse or IBM® Rational® Application Developer environment, click Window > Preferences.
     b. Expand the Java > Build Path branch and select User Libraries. Click New.
     c. Select the eXtreme Scale user library. Click Add JARs.
        i. Browse and select the objectgrid.jar or ogclient.jar files from the *wxs_root*/lib directory. Click OK. Select the ogclient.jar file if you are developing client applications or local, in-memory caches. If you are developing and testing eXtreme Scale servers, use the objectgrid.jar file.
        ii. To include Javadoc for the ObjectGrid APIs, select the Javadoc location for the objectgrid.jar or ogclient.jar file that you added in the previous step. Click Edit.
     d. Click OK to apply the settings and close the Preferences window.

     The eXtreme Scale libraries are now in the build path for the project.
  2. Add the user library to your Java project.
     a. From the package explorer, right-click the project and select Properties.
     b. Select the Libraries tab.
     c. Click Add Library.
     d. Select User Library. Click Next.
     e. Select the eXtreme Scale user library that you configured earlier.
     f. Click OK to apply the changes and close the Properties window.
- Run a Java SE application in Eclipse. Create a run configuration to run your application.
  1. Configure Eclipse to build and run a Java SE application with WebSphere eXtreme Scale. From the Run menu select Run Configurations.
  2. Right-click the Java Application category and select New.
  3. Select the new run configuration, named *New_Configuration*.
  4. Configure the profile.
     - Project (on main tabbed page): *your_project_name*
     - Main Class (on main tabbed page): *your_main_class*
     - VM arguments (on arguments tabbed page): `-Djava.endorsed.dirs=wxs_root/lib/endorsed`

     Problems with the VM Arguments often occur because the path to java.endorsed.dirs must be an absolute path with no variables or shortcuts.

     Other common setup problems involve the Object Request Broker (ORB). You might see the following error. Refer to Configuring a custom Object Request Broker for more information:

     ```
     Caused by: java.lang.RuntimeException: The ORB that comes
     with the Sun Java implementation does not work with
     ObjectGrid at this time.
     ```

     If you do not have the objectGrid.xml or deployment.xml accessible to the application, you might see the following error:

     ```
     Exception in thread "P=211046:O=0:CT" com.ibm.websphere.objectgrid.
                                    ObjectGridRuntimeException: Cannot start OG container at
                                    Client.startTestServer(Client.java:161) at Client.
                                    main(Client.java:82) Caused by: java.lang.IllegalArgumentException:
                                    The objectGridXML must not be null      at com.ibm.websphere.objectgrid.
                                    deployment.DeploymentPolicyFactory.createDeploymentPolicy
                               (DeploymentPolicyFactory.java:55) at Client.startTestServer(Client.
                                    java:154) .. 1 more
     ```

  5. Click Apply and close the window, or click Run.

Related concepts:
Java API overview

Java API overview
Related information:
API documentation
API documentation

Running a WebSphere eXtreme Scale application that uses an application server other than WebSphere Application Server in Eclipse

You can configure a Java™ EE application that uses WebSphere® eXtreme Scale to run in an application server other than WebSphere Application Server in Eclipse.

Before you begin

- Install the stand-alone version of the WebSphere eXtreme Scale product, or download and extract the WebSphere eXtreme Scale trial version. For more information, see Installing.
- Install an application server, such as Apache Tomcat Version 6.0 or later.
- Install Eclipse and create a Java EE web application. The Java EE perspective is required and must be installed in your Eclipse environment.
- Download the API documentation. For more information, see IBM Elastic Caching Community wiki: API documentation downloads.

About this task

The following procedure was tested with Apache Tomcat and JBoss Application Server. The instructions also apply to other application servers.

Procedure

1. Add WebSphere eXtreme Scale runtime library to your Java EE build path. The steps vary slightly if you are using a full installation of WebSphere eXtreme Scale or an installation of .
   WebSphere eXtreme Scale Client

   Use the following steps if you have only WebSphere eXtreme Scale Client installed:

   a. Window > Preferences > Java > Build Path > User Libraries. Click New.
   b. Enter a User library name of `eXtremeScaleClient`, and click OK.
   c. Click Add Jars..., and select the *wxs_home*/lib/ogclient.jar file. Click Open.
   d. Optional: To add Javadoc, select Javadoc location and click Edit.... Enter your local download location.
   e. Click OK.
   f. Click OK to close out the User Libraries dialog.
   g. Click Project > Properties.
   h. Click Java Build Path.
   i. Click Add Library.
   j. Select User Library. Click Next.
   k. Check the eXtremeScaleClient library and click Finish.
   l. Click OK to close the Project Properties dialog.

   Full WebSphere eXtreme Scale installation

   Use the following steps if you used the full installation to install a client and server:

   a. Click Window > Preferences > Java > Build Path > User Libraries. Click New.
   b. Enter a User library name of `eXtremeScale`, and click OK.
   c. Click Add Jars..., and select *wxs_home*/lib/objectgrid.jar. Click Open.
   d. (Optional) To add Javadoc, select Javadoc location and click Edit.... Enter your local download location.
   e. Click OK.
   f. Click OK to close out the User Libraries dialog.
   g. Click Project > Properties.
   h. Click Java Build Path.
   i. Click Add Library.
   j. Select User Library. Click Next.
   k. Check the eXtremeScaleClient library and click Finish.
   l. Click OK to close the Project Properties dialog.
2. Add Java EE application projects to the server.
   a. Ensure that you are in the Java EE perspective and click the Servers tab in the bottom pane. You can also click Window > Show View > Servers.
   b. Right-click in the Servers pane, and choose New > Server.
   c. Choose your application server. Click Next.
   d. Click Browse... Select the root directory of your application server. Click OK.
   e. Click Next.
   f. Select your Java EE application project in the left Available pane and click Add > to move it to the right Configured pane on the server, and click Finish.
3. Resolve any remaining errors for the Project. Use the following steps to eliminate errors in the Problems pane:
   a. Click Project > Clean > *project_name*. Click OK. Build the project.
   b. Right-click on the Java EE project, and choose Build Path > Configure Build Path.
   c. Click the Libraries tab. Ensure that Apache Tomcat or your other application server, eXtremeScaleClient, and JRE are on the path.

4. Create a run configuration to run your application.
    a. From the Run menu, select Run Configurations.
    b. Right-click the Java Application category and select New.
    c. Select the new run configuration, named *New_Configuration*.
    d. Configure the profile.
        - Project (on main tabbed page): *your_project_name*
        - Main Class (on main tabbed page): *your_main_class*
        - VM arguments (on arguments tabbed page): `-Djava.endorsed.dirs=wxs_home/lib/endorsed`

    Problems with the VM arguments often occur because the path to the java.endorsed.dirs directory must be an absolute path with no variables or shortcuts.

    Other common setup problems involve the Object Request Broker (ORB). You might see the following error:

    ```
    Caused by: java.lang.RuntimeException: The ORB that comes with the
    Java implementation does not work with ObjectGrid at this time.
    ```

    For more information, see Configuring a custom Object Request Broker.
    If you do not have the objectGrid.xml or deployment.xml files accessible to the application, you might see the following error:

    ```
    Exception in thread "P=211046:O=0:CT" com.ibm.websphere.objectgrid.ObjectGridRuntimeException:
    Cannot start OG container
            at Client.startTestServer(Client.java:161)
            at Client.main(Client.java:82)
    Caused by: java.lang.IllegalArgumentException: The objectGridXML must not be null
            at com.ibm.websphere.objectgrid.deployment.DeploymentPolicyFactory.createDeploymentPolicy
      (DeploymentPolicyFactory.java:55)
            at Client.startTestServer(Client.java:154)
            ... 1 more
    ```

5. Click Apply and close the window, or click Run.

Results

You can now run your Java EE application that uses WebSphere eXtreme Scale in Eclipse.
Related concepts:
Java API overview
Java API overview
Related information:
API documentation
API documentation

Running an integrated client or server application with WebSphere Application Server in Rational Application Developer

Configure and run a Java™ EE application with a WebSphere® eXtreme Scale client or server with the WebSphere Application Server runtime embedded in Rational® Application Developer.

Before you begin

The following steps are for WebSphere Application Server Version 7.0 with Rational Application Developer Version 7.5. The following steps might vary if you are using different versions of these products.

- Install Rational Application Developer with WebSphere Application Server Test Environment extensions.
- Install WebSphere eXtreme Scale into the WebSphere Application Server, Version 7.0 Test Environment in the *rad_home*\runtimes\base_v7 directory. For more information, see Installing WebSphere eXtreme Scale or WebSphere eXtreme Scale Client with WebSphere Application Server.

Procedure

1. Define a eXtreme Scale server that is integrated with WebSphere Application Server for your project.
    a. In the Java EE perspective, click Window > Show View > Servers.
    b. Right-click in the Servers pane. Choose New > Server.
    c. Choose **IBM® WebSphere Application Server v7.0**. Click Next.
    d. Select a profile to use. The default is `was70profile1`.
    e. Enter the server name. The default is `server1`.
    f. Click Next.
    g. Select your Java EE application in the Available pane. Click Add > to move it to the Configured pane on the server. Click Finish.
2. To run the Java EE application, start the application server. Right-click **WebSphere Application Server v7.0** and select Start.

Related concepts:
Java API overview
Java API overview
Related information:
API documentation
API documentation

Accessing data with client applications

After you configure your development environment, you can begin to develop applications that create, access, and manage the data in your data grid.

About this task

From the perspective of a client application, using WebSphere® eXtreme Scale involves the following main steps:

- Connecting to the catalog service by obtaining a ClientClusterContext instance.
- Obtaining a client ObjectGrid instance.
- Getting a Session instance.
- Getting an ObjectMap instance.
- Using the ObjectMap methods.

- Connecting to distributed ObjectGrid instances programmatically
  You can connect to a distributed ObjectGrid with the connection end points for the catalog service domain. You must have the host name and listener port of each catalog server in the catalog service domain to which you want to connect.
- Tracking map updates by an application
  When an application is making changes to a Map during a transaction, a LogSequence object tracks those changes. If the application changes an entry in the map, a corresponding LogElement object provides the details of the change.
- Interacting with an ObjectGrid using the ObjectGridManager interface
  The ObjectGridManagerFactory class and the ObjectGridManager interface provide a mechanism to create, access, and add data to ObjectGrid instances. The ObjectGridManagerFactory class is a static helper class to access the ObjectGridManager interface, a singleton. The ObjectGridManager interface includes several convenience methods to create instances of an ObjectGrid object. The ObjectGridManager interface also facilitates creation and caching of ObjectGrid instances that can be accessed by several users.
- Accessing data with indexes (Index API)
  Use indexing for more efficient data access.
- Using Sessions to access data in the grid
  Your applications can begin and end transactions through the Session interface. The Session interface also provides access to the application-based ObjectMap and JavaMap interfaces.
- Caching objects with no relationships involved (ObjectMap API)
  ObjectMaps are like Java™ Maps that allow data to be stored as key-value pairs. ObjectMaps provide a simple and intuitive approach for the application to store data. An ObjectMap is ideal for caching objects that have no relationships involved. If object relationships are involved, then you should use the EntityManager API.
- Caching objects and their relationships (EntityManager API)
  Most cache products use map-based APIs to store data as key-value pairs. The ObjectMap API and the dynamic cache in WebSphere Application Server, among others, use this approach. However, map-based APIs have limitations. The EntityManager API simplifies the interaction with the data grid by providing an easy way to declare and interact with a complex graph of related objects.
- Retrieving entities and objects (Query API)
  WebSphere eXtreme Scale provides a flexible query engine for retrieving entities using the EntityManager API and Java objects using the ObjectQuery API.
- Programming for transactions in Java applications
  When you write a Java application that requires transactions, you must consider issues such as lock handling, collision handling, and transaction isolation.
- Configuring Java clients programmatically
  You can override client-side settings programmatically. Create an ObjectGridConfiguration object that is similar in structure to the server-side ObjectGrid instance.

Connecting to distributed ObjectGrid instances programmatically

You can connect to a distributed ObjectGrid with the connection end points for the catalog service domain. You must have the host name and listener port of each catalog server in the catalog service domain to which you want to connect.

Before you begin

- To connect to a distributed data grid, you must configure your server-side environment with a catalog service and container servers.
- You must have the listener port for each catalog service. For more information, see Planning for network ports.
- If the client application is running in WebSphere Application Server augmented with eXtreme Scale, configure the catalog service domain using the WebSphere Application Server administrative console or wsadmin.

About this task

8.5+ When running in a Java™ EE application, consider using the eXtreme Scale resource adapter. The resource adapter allows the application to look up a ObjectGrid connection in Java Naming Directory Interface (JNDI) using a Java Connector Architecture (JCA) connection factory, which significantly simplifies access to the data grid and allows integration with Java Transaction API (JTA) transactions. For more information, see Scenario: Using JCA to connect transactional applications to eXtreme Scale clients.

The ObjectGridManager.connect() methods connect to a catalog service domain using the supplied connection end points and returns a ClientClusterContext object that is used to retrieve ObjectGrid instances for the domain. The connection end points are a comma-delimited list of host and port combinations for each catalog server in the catalog service domain. See the following format of the catalog service endpoints:

```
catalogServiceEndpoints ::= <catalogServiceEndpoint> [,<catalogServiceEndpoint>]
catalogServiceEndpoint  ::= <hostName> : <listenerPort>
hostName                ::= The IP address or host name of a catalog service.
listenerPort            ::= The listener port that the catalog service is configured to use.
```

After you connect to the catalog service domain, use the ObjectGridManagerFactory.getObjectGrid(ClientClusterContext ccc, String objectGridName) method to retrieve a named ObjectGrid client instance. This ObjectGrid instance is a proxy for the named data grid and is cached in the client application. The ObjectGrid instance represents a logical connection to the remote data grid and is thread safe. All underlying physical connections to the data grid are managed automatically and can tolerate failure events.

The connection steps vary depending on whether you are using a stand-alone configuration or WebSphere Application Server.

Procedure

- Connect to a stand-alone distributed data grid using explicit catalog service end points.

```
// Retrieve an ObjectGridManager instance.
ObjectGridManager ogm = ObjectGridManagerFactory.getObjectGridManager();

// Obtain a ClientClusterContext by connecting to a catalog
// service domain, manually suppling the catalog service endpoints,
// and optionally specifying the ClientSecurityConfiguration and
// client ObjectGrid override XML file URL.
String catalogServiceEndpoints = "host1:2809,host2:2809";
ClientClusterContext ccc = ogm.connect(catalogServiceEndpoints,
                (ClientSecurityConfiguration) null, (URL) null);

// Obtain a distributed ObjectGrid using ObjectGridManager and providing
// the ClientClusterContext.
ObjectGrid og = ogm.getObjectGrid(ccc, "Mygrid");
```

- Connect to a catalog service domain from a client application that is hosted in WebSphere Application Server, where the catalog service domain was configured using the administrative console or admin task. The catalog service endpoints can be retrieved from a named domain identifier or for the default domain using the ObjectGridManager.

```
// Retrieve an ObjectGridManager instance.
ObjectGridManager ogm = ObjectGridManagerFactory.getObjectGridManager();

// Retrieve the domain by its ID (the name given to it in the admin console or wsadmin)
// The CatalogDomainManager also includes methods to retrieve all domains and the default domain.
CatalogDomainInfo di = ogm.getCatalogDomainManager().getDomainInfo("ProductionDomain");
if(di == null) throw new IllegalStateException("Domain not configured");

// Connect to the domain using the catalog service endpoints and the security configuration
// in the CatalogDomainInfo object.  The client override ObjectGrid XML is optional
// and is manually supplied.
ClientClusterContext ccc = ogm.connect(di.getClientCatalogServiceEndpoints(),
                di.getClientSecurityConfiguration(), (URL) null);

// Obtain a distributed ObjectGrid using ObjectGridManager and by providing
// the ClientClusterContext.
ObjectGrid og = ogm.getObjectGrid(ccc, "MyGrid");
```

What to do next

If the catalog service domain is hosted in a WebSphere Application Server deployment manager, clients outside of the cell, including Java Platform, Enterprise Edition clients, must connect to the catalog service using the deployment manager host name and the IIOP bootstrap port. When the catalog service runs in WebSphere Application Server cells, and the clients run outside of the cells, look to the eXtreme Scale domain configuration pages in the WebSphere Application Server administrative console for the information that you need to point a client to the catalog service.

Related concepts:
Embedded server API
Interacting with an ObjectGrid using the ObjectGridManager interface
Related tasks:
Configuration methods
Configuring data grids
Configuring deployment policies
Related reference:
ObjectGrid descriptor XML file
Deployment policy descriptor XML file
Server properties file
Client properties file
Related information:
ObjectGridManager interface
ClientClusterContext interface
DeploymentPolicy interface

Tracking map updates by an application

When an application is making changes to a Map during a transaction, a LogSequence object tracks those changes. If the application changes an entry in the map, a corresponding LogElement object provides the details of the change.

Loaders are given a LogSequence object for a particular map whenever an application calls for a flush or commit to the transaction. The Loader iterates over the LogElement objects within the LogSequence object and applies each LogElement object to the backend.

ObjectGridEventListener listeners that are registered with an ObjectGrid also use LogSequence objects. These listeners are given a LogSequence object for each map in a committed transaction. Applications can use these listeners to wait for certain entries to change, like a trigger in a conventional database.

The following log-related interfaces or classes are provided by the eXtreme Scale framework:

- com.ibm.websphere.objectgrid.plugins.LogElement
- com.ibm.websphere.objectgrid.plugins.LogSequence
- com.ibm.websphere.objectgrid.plugins.LogSequenceFilter
- com.ibm.websphere.objectgrid.plugins.LogSequenceTransformer

LogElement interface

A LogElement represents an operation on an entry during a transaction. A LogElement object has several methods to get its various attributes. The most commonly used attributes are the type and the current value attributes fetched by getType() and getCurrentValue().

8.5 The type is represented by one of the constants defined in the LogElement interface: INSERT, UPDATE, DELETE, EVICT, FETCH, or TOUCH.

The current value represents the new value for the operation if it is INSERT, UPDATE, or FETCH. If the operation is TOUCH, DELETE, or EVICT, then the current value is null. This value can be cast to ValueProxyInfo when a ValueInterface is in use.

See the API documentation for more details on the LogElement interface.

LogSequence interface

In most transactions, operations to more than one entry in a map occur, so multiple LogElement objects are created. You should create an object that behaves as a composite of multiple LogElement objects. The LogSequence interface serves this purpose by containing a list of LogElement objects.

See the API documentation for more details on the LogSequence interface.

Using LogElement and LogSequence

LogElement and LogSequence are widely used in eXtreme Scale and by ObjectGrid plug-ins that are written by users when operations are propagated from one component or server to another component or server. For example, a LogSequence object can be used by the distributed ObjectGrid transaction propagation function to propagate the changes to other servers, or it can be applied to the persistence store by the loader. LogSequence is mainly used by the following interfaces.

- com.ibm.websphere.objectgrid.plugins.ObjectGridEventListener
- com.ibm.websphere.objectgrid.plugins.Loader
- com.ibm.websphere.objectgrid.plugins.Evictor
- com.ibm.websphere.objectgrid.Session

Loader example

This section demonstrates how the LogSequence and LogElement objects are used in a Loader. A Loader is used to load data from and persist data into a persistent store. The batchUpdate method of the Loader interface uses LogSequence object:

```
void batchUpdate(TxID txid, LogSequence sequence) throws
                        LoaderException, OptimisticCollisionException;
```

The batchUpdate method is called when an ObjectGrid needs to apply all current changes to the Loader. The Loader is given a list of LogElement objects for the map, encapsulated in a LogSequence object. The implementation of the batchUpdate method must iterate over the changes and apply them to the backend. The following code snippet demonstrates how a Loader uses a LogSequence object. The snippet iterates over the set of changes and builds up three batch Java™ database connectivity (JDBC) statements: inserts, updates, and deletes:

```
public void batchUpdate(TxID tx, LogSequence sequence) throws LoaderException
{
    // Get a SQL connection to use.
    Connection conn = getConnection(tx);
    try
    {
    // Process the list of changes and build a set of prepared
    // statements for executing a batch update, insert, or delete
    // SQL operations. The statements are cached in stmtCache.
    Iterator iter = sequence.getPendingChanges();
    while ( iter.hasNext() )
    {
        LogElement logElement = (LogElement)iter.next();
        Object key = logElement.getCacheEntry().getKey();
        Object value = logElement.getCurrentValue();
        switch ( logElement.getType().getCode() )
        {
            case LogElement.CODE_INSERT:
                buildBatchSQLInsert( key, value, conn );
                break;
```

```
            case LogElement.CODE_UPDATE:
                buildBatchSQLUpdate( key, value, conn );
                break;
            case LogElement.CODE_DELETE:
                buildBatchSQLDelete( key, conn );
                break;
        }
    }
    // Run the batch statements that were built by above loop.
    Collection statements = getPreparedStatementCollection( tx, conn );
    iter = statements.iterator();
    while ( iter.hasNext() )
    {
        PreparedStatement pstmt = (PreparedStatement) iter.next();
        pstmt.executeBatch();
    }
} catch (SQLException e)
{
    LoaderException ex = new LoaderException(e);
    throw ex;
}
}
```

The previous sample illustrates the high-level logic of processing the LogSequence argument. However, the sample does not illustrate the details of how an SQL insert, update, or delete statement is built. The getPendingChanges method is called on the LogSequence argument to obtain an iterator of LogElement objects that a Loader needs to process, and the LogElement.getType().getCode() method is used to determine whether a LogElement is for an SQL insert, update, or delete operation.

Evictor sample

You can also use LogSequence and LogElement objects with an Evictor. An Evictor is used to evict the map entries from the backing map based on certain criteria. The apply method of the Evictor interface uses LogSequence.

```
/**
* This is called during cache commit to allow the evictor to track object usage
* in a backing map. This will also report any entries that have been successfully
* evicted.
*
* @param sequence LogSequence of changes to the map
*/
void apply(LogSequence sequence);
```

For information on how the apply method uses LogSequence, refer to the code sample in the Custom evictors topic.

LogSequenceFilter and LogSequenceTransformer interfaces

Sometimes, it is necessary to filter the LogElement objects so that only LogElement objects with certain criteria are accepted, and reject other objects. For example, you might want to serialize a certain LogElement based on some criterion.

LogSequenceFilter solves this problem with the following method.

```
public boolean accept (LogElement logElement);
```

This method returns true if the given LogElement should be used in the operation, and returns false if the given LogElement should not be used.

LogSequenceTransformer is a class that uses the LogSequenceFilter function. It uses the LogSequenceFilter to filter out some LogElement objects and then serialize the accepted LogElement objects. This class has two methods. The first method follows.

```
public static void serialize(Collection logSequences, ObjectOutputStream stream,
                        LogSequenceFilter filter, DistributionMode mode) throws IOException
```

This method allows the caller to provide a filter for determining which LogElements to include in the serialization process. The DistributionMode parameter allows the caller to control the serialization process. For example, if the distribution mode is invalidation only, then there is no need to serialize the value. The second method of this class is the inflate method, as follows.

```
public static Collection inflate(ObjectInputStream stream, ObjectGrid
                        objectGrid) throws IOException, ClassNotFoundException
```

The inflate method reads the log sequence serialized form, which was created by the serialize method, from the provided object input stream.

Interacting with an ObjectGrid using the ObjectGridManager interface

The ObjectGridManagerFactory class and the ObjectGridManager interface provide a mechanism to create, access, and add data to ObjectGrid instances. The ObjectGridManagerFactory class is a static helper class to access the ObjectGridManager interface, a singleton. The ObjectGridManager interface includes several convenience methods to create instances of an ObjectGrid object. The ObjectGridManager interface also facilitates creation and caching of ObjectGrid instances that can be accessed by several users.

- Creating ObjectGrid instances with the ObjectGridManager interface
  Each of these methods creates a local instance of an ObjectGrid.

- Retrieving a ObjectGrid instance with the ObjectGridManager interface
  Use the ObjectGridManager.getObjectGrid methods to retrieve cached instances.
- Removing ObjectGrid instances with the ObjectGridManager interface
  You can use two different removeObjectGrid methods to remove ObjectGrid instances from the cache.
- Controlling the lifecycle of an ObjectGrid with the ObjectGridManager interface
  You can use the ObjectGridManager interface to control the lifecycle of an ObjectGrid instance using either a startup bean or a servlet.
- Accessing the ObjectGrid shard
  WebSphere® eXtreme Scale achieves high processing rates by moving the logic to where the data is and returning only results back to the client.

Related tasks:
Configuration methods
Configuring data grids
Connecting to distributed ObjectGrid instances programmatically
Configuring deployment policies
Related reference:
ObjectGrid descriptor XML file
Deployment policy descriptor XML file
Server properties file
Client properties file
Related information:
ObjectGridManager interface
ClientClusterContext interface
DeploymentPolicy interface

Creating ObjectGrid instances with the ObjectGridManager interface

Each of these methods creates a local instance of an ObjectGrid.

Local in-memory instance

The following code snippet illustrates how to obtain and configure a local ObjectGrid instance with eXtreme Scale.

```
// Obtain a local ObjectGrid reference
        // you can create a new ObjectGrid, or get configured ObjectGrid
        // defined in ObjectGrid xml file
        ObjectGridManager objectGridManager =
                                        ObjectGridManagerFactory.getObjectGridManager();
        ObjectGrid ivObjectGrid =
                                        objectGridManager.createObjectGrid("objectgridName");

        // Add a TransactionCallback into ObjectGrid
        HeapTransactionCallback tcb = new HeapTransactionCallback();
        ivObjectGrid.setTransactionCallback(tcb);

        // Define a BackingMap
        // if the BackingMap is configured in ObjectGrid xml
        // file, you can just get it.
        BackingMap ivBackingMap = ivObjectGrid.defineMap("myMap");

        // Add a Loader into BackingMap
        Loader ivLoader = new HeapCacheLoader();
        ivBackingMap.setLoader(ivLoader);

        // initialize ObjectGrid
        ivObjectGrid.initialize();

        // Obtain a session to be used by the current thread.
        // Session can not be shared by multiple threads
        Session ivSession = ivObjectGrid.getSession();

        // Obtaining ObjectMap from ObjectGrid Session
        ObjectMap objectMap = ivSession.getMap("myMap");
```

Default shared configuration

The following code is a simple case of creating an ObjectGrid to share among many users.

```
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridException;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
final ObjectGridManager oGridManager=
                ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid employees =
        oGridManager.createObjectGrid("Employees",true);
employees.initialize();
employees.
/*sample continues..*/
```

The preceding Java™ code snippet creates and caches the Employees ObjectGrid. The Employees ObjectGrid is initialized with the default configuration and is ready for use. The second parameter in the createObjectGrid method is set to true, which instructs the ObjectGridManager to cache the ObjectGrid instance it creates. If this parameter is set to false, the instance is not cached. Every ObjectGrid instance has a name, and the instance can be shared among many clients or users based on that name.

If the objectGrid instance is used in peer-to-peer sharing, the caching must be set to true. For more information on peer-to-peer sharing, see Distributing changes between peer Java Virtual Machines.

XML configuration

WebSphere® eXtreme Scale is highly configurable. The previous example demonstrates how to create a simple ObjectGrid without any configuration. This example shows you how to create a pre-configured ObjectdGrid instance that is based on an XML configuration file. You can configure an ObjectGrid instance programmatically or using an XML-based configuration file. You can also configure ObjectGrid using a combination of both approaches. The ObjectGridManager interface allows creation of an ObjectGrid instance based on the XML configuration. The ObjectGridManager interface has several methods that take a URL as an argument. Every XML file that is passed into the ObjectGridManager must be validated against the schema. XML validation can be disabled only when the file is previously validated and no changes have been made to the file since its last validation. Disabling validation saves a small amount of overhead but introduces the possibility of using an invalid XML file. The IBM® Java Developer Kit (JDK)Version 5 or later has support for XML validation. When using a JDK that does not have this support, Apache Xerces might be required to validate the XML.

The following Java code snippet demonstrates how to pass in an XML configuration file to create an ObjectGrid.

```
import java.net.MalformedURLException;
import java.net.URL;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridException;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
boolean validateXML = true; // turn XML validation on
boolean cacheInstance = true; // Cache the instance
String objectGridName="Employees"; // Name of Object Grid URL
allObjectGrids = new URL("file:test/myObjectGrid.xml");
final ObjectGridManager oGridManager=
        ObjectGridManagerFactory.getObjectGridManager();
 ObjectGrid employees =
        oGridManager.createObjectGrid(objectGridName, allObjectGrids,
                bvalidateXML, cacheInstance);
```

The XML file can contain configuration information for several ObjectGrids. The previous code snippet specifically returns ObjectGrid Employees, assuming that the Employees configuration is defined in the file.

createObjectGrid methods

.

```
/**
 * A simple factory method to return an instance of an
 * Object Grid. A unique name is assigned.
 * The instance of ObjectGrid is not cached.
 * Users can then use {@link ObjectGrid#setName(String)} to change the
 * ObjectGrid name.
 *
 * @return ObjectGrid an instance of ObjectGrid with a unique name assigned
 * @throws ObjectGridException any error encountered during the
 * ObjectGrid creation
 */
public ObjectGrid createObjectGrid() throws ObjectGridException;

/**
 * A simple factory method to return an instance of an ObjectGrid with the
 * specified name. The instances of ObjectGrid can be cached. If an ObjectGrid
 * with the this name has already been cached, an ObjectGridException
 * will be thrown.
 *
 * @param objectGridName the name of the ObjectGrid to be created.
 * @param cacheInstance true, if the ObjectGrid instance should be cached
 * @return an ObjectGrid instance
 * @this name has already been cached or
 * any error during the ObjectGrid creation.
 */
public ObjectGrid createObjectGrid(String objectGridName, boolean cacheInstance)
                throws ObjectGridException;

/**
 * Create an ObjectGrid instance with the specified ObjectGrid name. The
 * ObjectGrid instance created will be cached.
 * @param objectGridName the Name of the ObjectGrid instance to be created.
 * @return an ObjectGrid instance
 * @throws ObjectGridException if an ObjectGrid with this name has already
 * been cached, or any error encountered during the ObjectGrid creation
 */
public ObjectGrid createObjectGrid(String objectGridName)
                throws ObjectGridException;

/**
 * Create an ObjectGrid instance based on the specified ObjectGrid name and the
```

```
 * XML file. The ObjectGrid instance defined in the XML file with the specified
 * ObjectGrid name will be created and returned. If such an ObjectGrid
 * cannot be found in the xml file, an exception will be thrown.
 *
 * This ObjecGrid instance can be cached.
 *
 * If the URL is null, it will be simply ignored. In this case, this method behaves
 * the same as {@link #createObjectGrid(String, boolean)}.
 *
 * @param objectGridName the Name of the ObjectGrid instance to be returned. It
 * must not be null.
 * @param xmlFile a URL to a wellformed xml file based on the ObjectGrid schema.
 * @param enableXmlValidation if true the XML is validated
 * @param cacheInstance a boolean value indicating whether the ObjectGrid
 * instance(s)
 * defined in the XML will be cached or not. If true, the instance(s) will
 * be cached.
 *
 * @throws ObjectGridException if an ObjectGrid with the same name
 * has been previously cached, no ObjectGrid name can be found in the xml file,
 * or any other error during the ObjectGrid creation.
 * @return an ObjectGrid instance
 * @see ObjectGrid
 */
public ObjectGrid createObjectGrid(String objectGridName, final URL xmlFile,
final boolean enableXmlValidation, boolean cacheInstance)
        throws ObjectGridException;

/**
 * Process an XML file and create a List of ObjectGrid objects based
 * upon the file.
 * These ObjecGrid instances can be cached.
 * An ObjectGridException will be thrown when attempting to cache a
 * newly created ObjectGrid
 * that has the same name as an ObjectGrid that has already been cached.
 *
 * @param xmlFile the file that defines an ObjectGrid or multiple
 * ObjectGrids
 * @param enableXmlValidation setting to true will validate the XML
 * file against the schema
 * @param cacheInstances set to true to cache all ObjectGrid instances
 * created based on the file
 * @return an ObjectGrid instance
 * @throws ObjectGridException if attempting to create and cache an
 * ObjectGrid with the same name as
 * an ObjectGrid that has already been cached, or any other error
 * occurred during the
 * ObjectGrid creation
 */
public List createObjectGrids(final URL xmlFile, final boolean enableXmlValidation,
boolean cacheInstances) throws ObjectGridException;

/** Create all ObjectGrids that are found in the XML file. The XML file will be
 * validated against the schema. Each ObjectGrid instance that is created will
 * be cached. An ObjectGridException will be thrown when attempting to cache a
 * newly created ObjectGrid that has the same name as an ObjectGrid that has
 * already been cached.
 * @param xmlFile The XML file to process. ObjectGrids will be created based
 * on what is in the file.
 * @return A List of ObjectGrid instances that have been created.
 * @throws ObjectGridException if an ObjectGrid with the same name as any of
 * those found in the XML has already been cached, or
 * any other error encounterred during ObjectGrid creation.
 */
public List createObjectGrids(final URL xmlFile) throws ObjectGridException;

/**
 * Process the XML file and create a single ObjectGrid instance with the
 * objectGridName specified only if an ObjectGrid with that name is found in
 * the file. If there is no ObjectGrid with this name defined in the XML file,
 * an ObjectGridException
 * will be thrown. The ObjectGrid instance created will be cached.
 * @param objectGridName name of the ObjectGrid to create. This ObjectGrid
 * should be defined in the XML file.
 * @param xmlFile the XML file to process
 * @return A newly created ObjectGrid
 * @throws ObjectGridException if an ObjectGrid with the same name has been
 * previously cached, no ObjectGrid name can be found in the xml file,
 * or any other error during the ObjectGrid creation.
 */
public ObjectGrid createObjectGrid(String objectGridName, URL xmlFile)
                throws ObjectGridException;
```

Related tasks:
Troubleshooting client connectivity

Retrieving a ObjectGrid instance with the ObjectGridManager interface

Use the ObjectGridManager.getObjectGrid methods to retrieve cached instances.

Retrieving a cached instance

Since the Employees ObjectGrid instance was cached by the ObjectGridManager interface, another user can access it with the following code snippet:

```
ObjectGrid myEmployees = oGridManager.getObjectGrid("Employees");
```

The following two getObjectGrid methods return cached ObjectGrid instances:

- **Retrieving all cached instances**
  To obtain all of the ObjectGrid instances that have been previously cached, use the getObjectGrids method, which returns a list of each instance. If no cached instances exist, the method will return null.
- **Retrieving a cached instance by name**
  To obtain a single cached instance of an ObjectGrid, use getObjectGrid(String objectGridName), passing the name of the cached instance into the method. The method either returns the ObjectGrid instance with the specified name or returns null if there is no ObjectGrid instance with that name.

Note: You can also use the getObjectGrid method to connect to a distributed grid. See Connecting to distributed ObjectGrid instances programmatically for more information.

Removing ObjectGrid instances with the ObjectGridManager interface

You can use two different removeObjectGrid methods to remove ObjectGrid instances from the cache.

Remove an ObjectGrid instance

To remove ObjectGrid instances from the cache, use one of the removeObjectGrid methods. The ObjectGridManager interface does not keep a reference of the instances that are removed. Two remove methods exist. One method takes a boolean parameter. If the boolean parameter is set to `true`, the destroy method is called on the ObjectGrid. The call to the destroy method on the ObjectGrid shuts down the ObjectGrid and frees up any resources the ObjectGrid is using. A description of how to use the two removeObjectGrid methods follows:

```
/**
 * Remove an ObjectGrid from the cache of ObjectGrid instances
 *
 * @param objectGridName the name of the ObjectGrid instance to remove
 * from the cache
 *
 * @throws ObjectGridException if an ObjectGrid with the objectGridName
 * was not found in the cache
 */
public void removeObjectGrid(String objectGridName) throws ObjectGridException;

/**
 * Remove an ObjectGrid from the cache of ObjectGrid instances and
 * destroy its associated resources
 *
 * @param objectGridName the name of the ObjectGrid instance to remove
 * from the cache
 *
 * @param destroy destroy the objectgrid instance and its associated
 * resources
 *
 * @throws ObjectGridException if an ObjectGrid with the objectGridName
 * was not found in the cache
 */
public void removeObjectGrid(String objectGridName, boolean destroy)
        throws ObjectGridException;
```

Controlling the lifecycle of an ObjectGrid with the ObjectGridManager interface

You can use the ObjectGridManager interface to control the lifecycle of an ObjectGrid instance using either a startup bean or a servlet.

Managing lifecycle with a startup bean

A startup bean is used to control the lifecycle of an ObjectGrid instance. A startup bean loads when an application starts. With a startup bean, code can run whenever an application starts or stops as expected. To create a startup bean, use the home com.ibm.websphere.startupservice.AppStartUpHome interface and use the remote com.ibm.websphere.startupservice.AppStartUp interface. Implement the start and stop methods on the bean. The start method is invoked whenever the application starts up. The stop method is invoked when the application shuts down. The start method is used to create ObjectGrid instances. The stop method is used to remove ObjectGrid instances. A code snippet that demonstrates this ObjectGrid lifecycle management in a startup bean follows:

```
public class MyStartupBean implements javax.ejb.SessionBean {
    private ObjectGridManager objectGridManager;

    /* The methods on the SessionBean interface have been
```

```
     * left out of this example for the sake of brevity */

    public boolean start(){
        // Starting the startup bean
        // This method is called when the application starts
        objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
        try {
            // create 2 ObjectGrids and cache these instances
            ObjectGrid bookstoreGrid = objectGridManager.createObjectGrid("bookstore", true);
            bookstoreGrid.defineMap("book");
            ObjectGrid videostoreGrid = objectGridManager.createObjectGrid("videostore", true);
            // within the JVM,
            // these ObjectGrids can now be retrieved from the
            //ObjectGridManager using the getObjectGrid(String) method
        } catch (ObjectGridException e) {
            e.printStackTrace();
            return false;
        }

        return true;
    }

    public void stop(){
        // Stopping the startup bean
        // This method is called when the application is stopped
        try {
            // remove the cached ObjectGrids and destroy them
            objectGridManager.removeObjectGrid("bookstore", true);
            objectGridManager.removeObjectGrid("videostore", true);
        } catch (ObjectGridException e) {
            e.printStackTrace();
        }
    }
}
```

After the start method is called, the newly created ObjectGrid instances are retrieved from the ObjectGridManager interface. For example, if a servlet is included in the application, the servlet accesses the eXtreme Scale using the following code snippet:

```
ObjectGridManager objectGridManager =
        ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid bookstoreGrid = objectGridManager.getObjectGrid("bookstore");
ObjectGrid videostoreGrid = objectGridManager.getObjectGrid("videostore");
```

Managing lifecycle with a servlet

To manage the lifecycle of an ObjectGrid in a servlet, you can use the init method to create an ObjectGrid instance and the destroy method to remove the ObjectGrid instance. If the ObjectGrid instance is cached, it is retrieved and manipulated in the servlet code. Sample code that demonstrates ObjectGrid creation, manipulation, and destruction within a servlet follows:

```
public class MyObjectGridServlet extends HttpServlet implements Servlet {
    private ObjectGridManager objectGridManager;

    public MyObjectGridServlet() {
        super();
    }

    public void init(ServletConfig arg0) throws ServletException {
        super.init();
        objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
        try {
            // create and cache an ObjectGrid named bookstore
            ObjectGrid bookstoreGrid =
                                        objectGridManager.createObjectGrid("bookstore", true);
            bookstoreGrid.defineMap("book");
        } catch (ObjectGridException e) {
            e.printStackTrace();
        }
    }

    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        ObjectGrid bookstoreGrid = objectGridManager.getObjectGrid("bookstore");
        Session session = bookstoreGrid.getSession();
        ObjectMap bookMap = session.getMap("book");
        // perform operations on the cached ObjectGrid
        // ...
        // Close the session (optional in Version 7.1.1 and later) for improved performance
        session.close();
         }
    public void destroy() {
        super.destroy();
        try {
            // remove and destroy the cached bookstore ObjectGrid
            objectGridManager.removeObjectGrid("bookstore", true);
        } catch (ObjectGridException e) {
            e.printStackTrace();
        }
    }
}
```

Accessing the ObjectGrid shard

WebSphere® eXtreme Scale achieves high processing rates by moving the logic to where the data is and returning only results back to the client.

Application logic in a client Java™ virtual machine (JVM) needs to pull data from the server JVM that is holding the data and push it back when the transaction commits. This process slows down the rate the data can be processed. If the application logic was on the same JVM as the shard that is holding the data, then the network latency and marshalling cost is eliminated and can provide a significant performance boost.

Local reference to shard data

The ObjectGrid APIs provide a Session to the server-side method. This session is a direct reference to the data for that shard. No routing logic is on that path. The application logic can work with the data for that shard directly. The session cannot be used to access data in another partition because no routing logic exists.

A Loader plug-in also provides a way to receive an event when a shard becomes a primary partition. An application can implement a Loader and implement the ReplicaPreloadController interface. The check preload status method is only called when a shard becomes a primary. The session provided to that method is a local reference to the shards data. This approach is typically used if a partition primary needs to start some threads or subscribe to a message fabric for partition-related traffic. It might start a thread to listen for messages in a local Map using the getNextKey API.

Collocated client-server optimization

If an application uses the client APIs to access a partition that happens to be collocated with the JVM that contains the client, then the network is avoided but some marshalling still occurs because of current implementation issues. If a partitioned grid is used, then no impact on the performance of the application is made because (N-1)/N number of calls route to a different JVM. If you need local access always with a shard, then use the Loader or ObjectGrid APIs to invoke that logic.

Accessing data with indexes (Index API)

Use indexing for more efficient data access.

About this task

The HashIndex class is the built-in index plug-in implementation that can support both of the built-in application index interfaces: MapIndex and MapRangeIndex. You also can create your own indexes. You can add HashIndex as either a static or dynamic index into the backing map, obtain either MapIndex or MapRangeIndex index proxy object, and use the index proxy object to find cached objects.

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.

Note: In a distributed environment, if the index object is obtained from a client ObjectGrid, the index has a type client index object and all index operations run in a remote server ObjectGrid. If the map is partitioned, the index operations run on each partition remotely. The results from each partition are merged before returning the results to the application. The performance is determined by the number of partitions and the size of the result returned by each partition. Poor performance might occur if both factors are high.

Procedure

1. If you want to use indexes other than the default local index, add index plug-ins to the backing map.
   - **XML configuration:**

```
<backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsidxnonkey_person">
     <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsidxnonkey_MapIndexplugin
"
                        className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
        <property name="Name" type="java.lang.String" value="CODE"
                                              description="index name" />
        <property name="RangeIndex" type="boolean" value="true"
                                              description="true for MapRangeIndex" />
        <property name="AttributeName" type="java.lang.String" value="employeeCode"
                                              description="attribute name" />
     </bean>
</backingMapPluginCollection>
```

     In this XML configuration example, the built-in HashIndex class is used as the index plug-in. The HashIndex class supports properties that users can configure, such as Name, RangeIndex, and AttributeName in the previous example.

     - The Name property is configured as `CODE`, a string identifying this index plug-in. The Name property value must be unique within the scope of the BackingMap, and can be used to retrieve the index object by name from the ObjectMap instance for the BackingMap.
     - The RangeIndex property is configured as `true`, which means the application can cast the retrieved index object to the MapRangeIndex interface. If the RangeIndex property is configured as `false`, the application can only cast the retrieved index object to the MapIndex interface. A MapRangeIndex supports functions to find data using range functions such as greater than, less than, or both, while a MapIndex

only supports equals functions. If the index is used by query, the RangeIndex property must be configured to `true` on single-attribute indexes. For a relationship index and composite index, the RangeIndex property must be configured to `false`.
    - The AttributeName property is configured as `employeeCode`, which means the employeeCode attribute of the cached object is used to build a single-attribute index. If an application needs to search for cached objects with multiple attributes, the AttributeName property can be set to a comma-delimited list of attributes, yielding a composite index.
- **Programmatic configuration:**
  The BackingMap interface has two methods that you can use to add static index plug-ins: addMapIndexplugin and setMapIndexplugins. For more information, see BackingMap API. The following example creates the same configuration as the XML configuration example:

```
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;

    ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
    ObjectGrid ivObjectGrid = ogManager.createObjectGrid( "grid" );
    BackingMap personBackingMap = ivObjectGrid.getMap("person");

    // use the builtin HashIndex class as the index plugin class.
    HashIndex mapIndexplugin = new HashIndex();
    mapIndexplugin.setName("CODE");
    mapIndexplugin.setAttributeName("EmployeeCode");
    mapIndexplugin.setRangeIndex(true);
    personBackingMap.addMapIndexplugin(mapIndexplugin);
```

2. Access map keys and values with indexes.
    - **Local index:**
      To iterate through the keys and values in a local map, you can use the default index. The default index only works against the shard, using an agent or using the ObjectGrid instance retrieved from the ShardEvents.shardActivated(ObjectGrid shard) method. See the following example:

```
MapIndex keyIndex = (MapIndex)
objMap.getIndex(MapIndexPlugin.SYSTEM_KEY_INDEX_NAME);
Iterator keyIterator = keyIndex.findAll();
```

    - **Static indexes:**
      After a static index plug-in is added to a BackingMap configuration and the containing ObjectGrid instance is initialized, applications can retrieve the index object by name from the ObjectMap instance for the BackingMap. Cast the index object to the application index interface. Operations that the application index interface supports can now run.

```
Session session = ivObjectGrid.getSession();
ObjectMap map = session.getMap("person ");
MapRangeIndex codeIndex = (MapRangeIndex) m.getIndex("CODE");
Iterator iter = codeIndex.findLessEqual(new Integer(15));
while (iter.hasNext()) {
                Object key = iter.next();
                Object value = map.get(key);
}
// Close the session (optional in Version 7.1.1 and later) for improved performance
session.close();
```

    - **Dynamic indexes:**
      You can create and remove dynamic indexes from a BackingMap instance programmatically at any time. A dynamic index differs from a static index in that the dynamic index can be created even after the containing ObjectGrid instance is initialized. Unlike static indexing, the dynamic indexing is an asynchronous process, which requires the dynamic index to be in ready state before you use it. This method uses the same approach for retrieving and using the dynamic indexes as static indexes. You can remove a dynamic index if it is no longer needed. The BackingMap interface has methods to create and remove dynamic indexes.

      See the BackingMap API for more information about the createDynamicIndex and removeDynamicIndex methods.

```
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;

        ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
        ObjectGrid og = ogManager.createObjectGrid("grid");        BackingMap bm = og.getMap("person");
        og.initialize();
        // create index after ObjectGrid initialization without DynamicIndexCallback.
        bm.createDynamicIndex("CODE", true, "employeeCode", null);

        try {
            // If not using DynamicIndexCallback, need to wait for the Index to be ready.
            // The waiting time depends on the current size of the map
            Thread.sleep(3000);
        } catch (Throwable t) {
            // ...
        }

        // When the index is ready, applications can try to get application index
        // interface instance.
        // Applications have to find a way to ensure that the index is ready to use,
        // if not using DynamicIndexCallback interface.
        // The following example demonstrates the way to wait for the index to be ready
        // Consider the size of the map in the total waiting time.

        Session session = og.getSession();
        ObjectMap m = session.getMap("person");
```

```
        MapRangeIndex codeIndex = null;

        int counter = 0;
        int maxCounter = 10;
        boolean ready = false;
        while (!ready && counter < maxCounter) {
            try {
                counter++;
                codeIndex = (MapRangeIndex) m.getIndex("CODE");
                ready = true;
            } catch (IndexNotReadyException e) {
                // implies index is not ready, ...
                System.out.println("Index is not ready. continue to wait.");
                try {
                    Thread.sleep(3000);
                } catch (Throwable tt) {
                    // ...
                }
            } catch (Throwable t) {
                // unexpected exception
                t.printStackTrace();
            }
        }

        if (!ready) {
            System.out.println("Index is not ready.  Need to handle this situation.");
        }

        // Use the index to peform queries
        // Refer to the MapIndex or MapRangeIndex interface for supported operations.
        // The object attribute on which the index is created is the EmployeeCode.
        // Assume that the EmployeeCode attribute is Integer type: the
        // parameter that is passed into index operations has this data type.

        Iterator iter = codeIndex.findLessEqual(new Integer(15));

        // remove the dynamic index when no longer needed

        bm.removeDynamicIndex("CODE");
        // Close the session (optional in Version 7.1.1 and later) for improved performance
        session.close();
```

What to do next

You can use the DynamicIndexCallback interface to get notifications at the indexing events. See DynamicIndexCallback interface for more information.

- DynamicIndexCallback interface
  The DynamicIndexCallback interface is designed for applications that want to get notifications at the indexing events of ready, error, or destroy. The DynamicIndexCallback is an optional parameter for the createDynamicIndex method of the BackingMap. With a registered DynamicIndexCallback instance, applications can run business logic upon receiving notification of an indexing event.

Related concepts:
Plug-ins for indexing data
Plug-ins for custom indexing of cache objects
Using a composite index
Indexing
Tuning query performance
Related tasks:
Configuring the HashIndex plug-in
Related reference:
DynamicIndexCallback interface
HashIndex plug-in attributes
Related information:
DynamicIndexCallback API

DynamicIndexCallback interface

The DynamicIndexCallback interface is designed for applications that want to get notifications at the indexing events of ready, error, or destroy. The DynamicIndexCallback is an optional parameter for the createDynamicIndex method of the BackingMap. With a registered DynamicIndexCallback instance, applications can run business logic upon receiving notification of an indexing event.

Indexing events

For example, the ready event means that the index is ready for use. When a notification for this event is received, an application can try to retrieve and use the application index interface instance.

Example: Using the DynamicIndexCallback interface

```
BackingMap personBackingMap = ivObjectGrid.getMap("person");
    DynamicIndexCallback callback = new DynamicIndexCallbackImpl();
    personBackingMap.createDynamicIndex("CODE", true, "employeeCode", callback);


    class DynamicIndexCallbackImpl implements DynamicIndexCallback {
        public DynamicIndexCallbackImpl() {
        }

        public void ready(String indexName) {
            System.out.println("DynamicIndexCallbackImpl.ready() -> indexName = " + indexName);

            // Simulate what an application would do when notified that the index is ready.
            // Normally, the application would wait until the ready state is reached and then proceed
            // with any index usage logic.
            if("CODE".equals(indexName)) {
                ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
                ObjectGrid og = ogManager.createObjectGrid( "grid" );
                Session session = og.getSession();
                ObjectMap map = session.getMap("person");
                MapIndex codeIndex = (MapIndex) map.getIndex("CODE");
                Iterator iter = codeIndex.findAll(codeValue);
                // Close the session (optional in Version 7.1.1 and later) for improved performance
                session.close();
                                        }
        }

        public void error(String indexName, Throwable t) {
            System.out.println("DynamicIndexCallbackImpl.error() -> indexName = " + indexName);
            t.printStackTrace();
        }

        public void destroy(String indexName) {
            System.out.println("DynamicIndexCallbackImpl.destroy() -> indexName = " + indexName);
        }
    }
```

Related tasks:
Accessing data with indexes (Index API)
Related information:
DynamicIndexCallback API

Using Sessions to access data in the grid

Your applications can begin and end transactions through the Session interface. The Session interface also provides access to the application-based ObjectMap and JavaMap interfaces.

Each ObjectMap or JavaMap instance is directly tied to a specific Session object. Each thread that wants access to an eXtreme Scale must first obtain a Session instance from the ObjectGrid object. A Session instance cannot be shared concurrently between threads. WebSphere® eXtreme Scale does not use any thread local storage, but platform restrictions might limit the opportunity to pass a Session instance from one thread to another.

Methods

Get method

An application obtains a Session instance from an ObjectGrid object using the ObjectGrid.getSession method. The following example demonstrates how to obtain a Session instance:

```
ObjectGrid objectGrid = ...;
Session sess = objectGrid.getSession();
```

After a Session instance is obtained, the thread keeps a reference to the session for its own use. Calling the getSession method multiple times returns a new Session object each time.

Transactions and Session methods

A Session can be used to begin, commit, or rollback transactions. Operations against BackingMaps using ObjectMaps and JavaMaps are most efficiently performed within a Session transaction. After a transaction has started, any changes to one or more BackingMaps in that transaction scope are stored in a special transaction cache until the transaction is committed. When a transaction is committed, the pending changes are applied to the BackingMaps and Loaders and become visible to any other clients of that ObjectGrid.

WebSphere eXtreme Scale also supports the ability to automatically commit transactions, also known as auto-commit. If any ObjectMap operations are performed outside of the context of an active transaction, an implicit transaction is started before the operation and the transaction is automatically committed before returning control to the application.

```
Session session = objectGrid.getSession();
ObjectMap objectMap = session.getMap("someMap");
session.begin();
objectMap.insert("key1", "value1");
objectMap.insert("key2", "value2");
session.commit();
objectMap.insert("key3", "value3"); // auto-commit
```

Session.flush method

The Session.flush method only makes sense when a Loader is associated with a BackingMap. The flush method invokes the Loader with the current set of changes in the transaction cache. The Loader applies the changes to the backend. These changes are not committed when the flush is invoked. If a Session transaction is committed after a flush invocation, only updates that happen after the flush invocation are applied to the Loader. If a Session transaction is rolled back after a flush invocation, the flushed changes are discarded with all other pending changes in the transaction. Use the Flush method sparingly because it limits the opportunity for batch operations against a Loader. Following is an example of the usage of the Session.flush method:

```
Session session = objectGrid.getSession();
session.begin();
// make some changes
...
session.flush(); // push these changes to the Loader, but don't commit yet
// make some more changes
...
session.commit();
```

NoWriteThrough method

Some maps are backed by a Loader, which provides persistent storage for the data in the map. Sometimes it is useful to commit data just to the backing map and not push data out to the Loader. The Session interface provides the beginNoWriteThough method for this purpose. The beginNoWriteThrough method starts a transaction like the begin method. With the beginNoWriteThrough method, when the transaction is committed, the data is only committed to the in-memory map and is not committed to the persistent storage that is provided by the Loader. This method is very useful when performing data preload on the map.

When using a distributed ObjectGrid instance, the beginNoWriteThrough method is useful for making changes to the near cache only, without modifying the far cache on the server. If the data is known to be stale in the near cache, using the beginNoWriteThrough method can allow entries to be invalidated on the near cache without invalidating them on the server as well.

The Session interface also provides the isWriteThroughEnabled method to determine what type of transaction is currently active.

```
Session session = objectGrid.getSession();
session.beginNoWriteThrough();
// make some changes ...
session.commit(); // these changes will not get pushed to the Loader
```

Obtain the TxID object method

The TxID object is an opaque object that identifies the active transaction. Use the TxID object for the following purposes:

- For comparison when you are looking for a particular transaction.
- To store shared data between the TransactionCallback and Loader objects.

For more information, see Introduction to plug-in slots.

Performance monitoring method

If you are using eXtreme Scale within WebSphere Application Server, it might be necessary to reset the transaction type for performance monitoring. You can set the transaction type with the setTransactionType method. See Monitoring ObjectGrid performance with WebSphere Application Server performance monitoring infrastructure (PMI) for more information about the setTransactionType method.

Process a complete LogSequence method

WebSphere eXtreme Scale can propagate sets of map changes to ObjectGrid listeners as a means of distributing maps from one Java™ virtual machine to another. To make it easier for the listener to process the received LogSequences, the Session interface provides the processLogSequence method. This method examines each LogElement within the LogSequence and performs the appropriate operation, for example, insert, update, invalidate, and so on, against the BackingMap that is identified by the LogSequence MapName. An ObjectGrid Session must be available before the processLogSequence method is invoked. The application is also responsible for issuing the appropriate commit or rollback calls to complete the Session. Autocommit processing is not available for this method invocation. Normal processing by the receiving ObjectGridEventListener at the remote JVM would be to start a Session using the beginNoWriteThrough method, which prevents endless propagation of changes, followed by a call to this processLogSequence method, and then committing or rolling back the transaction.

```
// Use the Session object that was passed in during
//ObjectGridEventListener.initialization...
session.beginNoWriteThrough();
// process the received LogSequence
try {
        session.processLogSequence(receivedLogSequence);
} catch (Exception e) {
        session.rollback(); throw e;
}
// commit the changes
session.commit();
```

markRollbackOnly method

This method is used to mark the current transaction as "rollback only". Marking a transaction "rollback only" ensures that even if the commit method is called by application, the transaction is rolled back. This method is typically used by ObjectGrid itself or by the application when it knows that data corruption could occur if the transaction was allowed to be committed. After this method is called, the Throwable object that is passed to this method is chained to the com.ibm.websphere.objectgrid.TransactionException exception that results by the commit method if it is called on a Session that was previously marked a "rollback only". Any subsequent calls to this method for a transaction that is already marked as "rollback only" is ignored. That is, only the first call that passes a non-null Throwable reference is used. Once the marked transaction is completed, the "rollback only" mark is removed so that the next transaction that is started by the Session can be committed.

isMarkedRollbackOnly method

Returns if Session is currently marked as "rollback only". Boolean true is returned by this method if and only if markRollbackOnly method was previously called on this Session and the transaction started by the Session is still active.

setTransactionTimeout method

Set transaction timeout for next transaction started by this Session to a specified number of seconds. This method does not affect the transaction timeout of any transactions previously started by this Session. It only affects transactions that are started after this method is called. If this method is never called, then the timeout value that was passed to the setTxTimeout method of the com.ibm.websphere.objectgrid.ObjectGrid method is used.

getTransactionTimeout method

This method returns the transaction timeout value in seconds. The last value that was passed as the timeout value to the setTransactionTimeout method is returned by this method. If the setTransactionTimeout method is never called, then the timeout value that was passed to the setTxTimeout method of the com.ibm.websphere.objectgrid.ObjectGrid method is used.

transactionTimedOut method

This method returns boolean true if the current transaction that was started by this Session has timed out.

isFlushing method

This method returns boolean true if and only if all transaction changes are being flushed out to the Loader plug-in as a result of the flush method of Session interface being invoked. A Loader plug-in may find this method useful when it needs to know why its batchUpdate method was invoked.

isCommitting method

This method returns boolean true if and only if all transaction changes are being committed as a result of the commit method of Session interface being invoked. A Loader plug-in might find this method useful when it needs to know why its batchUpdate method was invoked.

setRequestRetryTimeout method

This method sets the request retry timeout value for the Session in milliseconds. If the client set a request retry timeout, the Session setting overrides the client value.

getRequestRetryTimeout method

This method gets the current request retry timeout setting on the Session. A value of `-1` indicates that the timeout is not set. A value of `0` indicates it is in fail-fast mode. A value greater than `0` indicates the timeout setting in milliseconds.

- SessionHandle for routing
  When you are using a per-container partition placement policy, you can use a SessionHandle object. A SessionHandle object contains partition information for the current Session and can be reused for a new Session.
- SessionHandle integration
  A SessionHandle object includes the partition information for the Session to which it is bound and facilitates request routing. SessionHandle objects apply to the per-container partition placement scenario only.

Related tasks:
Configuring request and retry timeout values

SessionHandle for routing

When you are using a per-container partition placement policy, you can use a SessionHandle object. A SessionHandle object contains partition information for the current Session and can be reused for a new Session.

A SessionHandle object includes information for the partition to which the current Session is bound. SessionHandle is extremely useful for the per-container partition placement policy and can be serialized with standard Java™ serialization.

If you have a SessionHandle object, you can apply that handle to a Session with the setSessionHandle(SessionHandle target) method, passing the handle in as the target. You can retrieve the SessionHandle object with the Session.getSessionHandle method.

Because it is only applicable in a per-container placement scenario, getting the SessionHandle object throws an IllegalStateException if a given data grid has multiple per-container map sets or has no per-container map sets. If you do not invoke the setSessionHandle method before calling the getSessionHandle method, the appropriate SessionHandle object is selected based on your client properties configuration.

You can also use the SessionHandleTransformer helper class to convert the handle into different formats. The methods of this class can change a handle's representation from byte array to instance, string to instance, and vice versa for both cases, and can also write the handle's contents into the output stream.

For an example on how you can use a SessionHandle object, see Zone-preferred routing.

SessionHandle integration

A SessionHandle object includes the partition information for the Session to which it is bound and facilitates request routing. SessionHandle objects apply to the per-container partition placement scenario only.

SessionHandle object for request routing

You can bind a SessionHandle object to a Session in the following ways:

Tip: In each of the following method calls, after a SessionHandle object is bound to a Session, the SessionHandle object can be obtained from the Session.getSessionHandle method.

- **Call the Session.getSessionHandle method:** When this method is called, if no SessionHandle object is bound to the Session, a SessionHandle object is selected randomly and bound to the Session.
- **Call transactional create, read, update, delete operations:** When these methods are called or at commit time, if no SessionHandle object is bound to the Session, a SessionHandle object is selected randomly and bound to the Session.
- **Call ObjectMap.getNextKey method:** When this method is called, if no SessionHandle object is bound to the Session, the operation request is randomly routed to individual partitions until a key is obtained. If a key is returned from a partition, a SessionHandle object that corresponds to that partition is bound to the Session. If no key is found, no SessionHandle is bound to the Session.
- **Call the QueryQueue.getNextEntity or QueryQueue.getNextEntities methods:** At the time this method is called, if no SessionHandle object is bound to the Session, the operation request is randomly routed to individual partitions until an object is obtained. If an object is returned from a partition, a SessionHandle object that corresponds to that partition is bound to the Session. If no object is found, no SessionHandle is bound to the Session.
- **Set a SessionHandle with the Session.setSessionHandle(SessionHandle sh) method:** If a SessionHandle is obtained from the Session.getSessionHandle method, the SessionHandle can be bound to a Session. Setting a SessionHandle influences request routing within the scope of the Session to which it is bound.

The Session.getSessionHandle method always returns a SessionHandle object. The method also automatically binds a SessionHandle on the Session if no SessionHandle object is bound to the Session. If you want to verify whether a Session has a SessionHandle object only , call the Session.isSessionHandleSet method. If this method returns a value of `false`, no SessionHandle object is currently bound to the Session.

Major operation types in the per-container placement scenario

A summary of the routing behavior of major operation types in the per-container partition placement scenario with respect to SessionHandle objects follows.

- **Session object with bound SessionHandle object**
  - Index - MapIndex and MapRangeIndex API: SessionHandle
  - Query and ObjectQuery: SessionHandle
  - Agent - MapGridAgent and ReduceGridAgent API: SessionHandle
  - ObjectMap.Clear: SessionHandle
  - ObjectMap.getNextKey: SessionHandle
  - QueryQueue.getNextEntity, QueryQueue.getNextEntities: SessionHandle
  - Transactional create, retrieve, update, and delete operations (ObjectMap API and EntityManager API): SessionHandle
- **Session object without bound SessionHandle object**
  - Index - MapIndex and MapRangeIndex API: All current active partitions
  - Query and ObjectQuery: Specified partition with setPartition method of Query and ObjectQuery
  - Agent - MapGridAgent and ReduceGridAgent
    - Not supported: ReduceGridAgent.reduce(Session s, ObjectMap map, Collection keys) and MapGridAgent.process(Session s, ObjectMap map, Object key) method.
    - All current active partitions: ReduceGridAgent.reduce(Session s, ObjectMap map) and MapGridAgent.processAllEntries(Session s, ObjectMap map) method.
  - ObjectMap.clear: All current active partitions.
  - ObjectMap.getNextKey: Binds a SessionHandle to the Session if a key is returned from one of the randomly selected partitions. If no key is returned, the Session is not bound to a SessionHandle.
  - QueryQueue: Specifies a partition with the QueryQueue.setPartition method. If no partition is set, the method randomly selects a partition to return. If an object is returned, the current Session is bound with the SessionHandle that is bound to the partition that returns the object. If no object is returned, the Session is not bound to a SessionHandle.
  - Transactional create, retrieve, update, and delete operations (ObjectMap API and EntityManager API): Randomly select a partition.

  In most cases, use SessionHandle to control routing to a particular partition. You can retrieve and cache the SessionHandle from the Session that inserts data. After caching the SessionHandle, you can set it on another Session so that you can route requests to the partition specified by the cached SessionHandle. To perform operations such as ObjectMap.clear without SessionHandle, you can temporarily set the SessionHandle to null by calling Session.setSessionHandle(null). Without a specified SessionHandle, operations run on all current active partitions.

- **QueryQueue routing behavior**
  In the per-container partition placement scenario, you can use SessionHandle to control routing of getNextEntity and getNextEntities methods of the QueryQueue API. If the Session is bound to a SessionHandle, requests route to the partition to which the SessionHandle is bound. If the Session is not bound to a SessionHandle, requests are routed to the partition set with the QueryQueue.setPartition method if a partition has been set in this way. If the Session has no bound SessionHandle or partition, a randomly selected partition are returned. If no such partition is found, the process stops and no SessionHandle is bound to the current Session.

The following snippet of code shows how to use SessionHandle objects.

```
Session ogSession = objectGrid.getSession();

// binding the SessionHandle
SessionHandle sessionHandle = ogSession.getSessionHandle();

ogSession.begin();
ObjectMap map = ogSession.getMap("planet");
map.insert("planet1", "mercury");

// transaction is routed to partition specified by SessionHandle
ogSession.commit();

// cache the SessionHandle that inserts data
SessionHandle cachedSessionHandle = ogSession.getSessionHandle();
```

```
// verify if SessionHandle is set on the Session
boolean isSessionHandleSet = ogSession.isSessionHandleSet();

// temporarily unbind the SessionHandle from the Session
if(isSessionHandleSet) {
    ogSession.setSessionHandle(null);
}

// if the Session has no SessionHandle bound,
// the clear operation will run on all current active partitions
// and thus remove all data from the map in the entire grid
map.clear();

// after clear is done, reset the SessionHandle back,
// if the Session needs to use previous SessionHandle.
// Optionally, calling getSessionHandle can get a new SessionHandle
ogSession.setSessionHandle(cachedSessionHandle);
```

Application design considerations

In the per-container placement strategy scenario, use the SessionHandle object for most operations. The SessionHandle object controls routing to partitions. To retrieve data, the SessionHandle object that you bind to the Session must be same SessionHandle object from any insert data transaction.

When you want to perform an operation without a SessionHandle set on the Session, you can unbind a SessionHandle from a Session by making a Session.setSessionHandle(null) method call.

When a Session is bound to a SessionHandle, all operation requests route to the partition that is specified by the SessionHandle object. Without the SessionHandle object set, operations route to either all partitions or a randomly selected partition.

Caching objects with no relationships involved (ObjectMap API)

ObjectMaps are like Java™ Maps that allow data to be stored as key-value pairs. ObjectMaps provide a simple and intuitive approach for the application to store data. An ObjectMap is ideal for caching objects that have no relationships involved. If object relationships are involved, then you should use the EntityManager API.

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

Applications typically obtain a WebSphere eXtreme Scale reference and then obtain a Session object from the reference for each thread. Sessions cannot be shared between threads. The getMap method of Session returns a reference to an ObjectMap to use for this thread.

- Introduction to ObjectMap
  The ObjectMap interface is used for transactional interaction between applications and BackingMaps.
- Creating dynamic maps with Java APIs
  You can create dynamic maps with Java APIs after the data grid has been instantiated. You can dynamically instantiate maps that are based on a set of map templates. You can create your own map templates.
- ObjectMap and JavaMap
  A JavaMap instance is obtained from an ObjectMap object. The JavaMap interface has the same method signatures as ObjectMap, but with different exception handling. JavaMap extends the java.util.Map interface, so all exceptions are instances of the java.lang.RuntimeException class. Because JavaMap extends the java.util.Map interface, it is easy to quickly use WebSphere eXtreme Scale with an existing application that uses a java.util.Map interface for object caching.
- Maps as FIFO queues
  With WebSphere eXtreme Scale, you can provide a first-in first-out (FIFO) queue-like capability for all maps. WebSphere eXtreme Scale tracks the insertion order for all maps. A client can ask a map for the next unlocked entry in a map in the order of insertion and lock the entry. This process allows multiple clients to consume entries from the map efficiently.

Related tasks:
Getting started with developing applications
Tutorial: Storing order information in entities
Related reference:
Introduction to ObjectMap
ObjectMap and JavaMap
Maps as FIFO queues
Related information:
API documentation
Getting started tutorial lesson 2.1: Creating a client application
ObjectMap interface
BackingMap interface
JavaMap interface

Routing cache objects to the same partition

When an eXtreme Scale configuration uses the fixed partition placement strategy, it depends on hashing the key to a partition to insert, get, update, or remove the value. The hashCode method is called on the key and it must be well defined if a custom key is created. However, another option is to use the

PartitionableKey interface. With the PartitionableKey interface, you can use an object other than the key to hash to a partition.

You can use the PartitionableKey interface in situations where there are multiple maps and the data you commit is related and thus should be put on the same partition. WebSphere® eXtreme Scale does not support two-phase commit so multiple map transactions should not be committed if they span multiple partitions. If the PartitionableKey hashes to the same partition for keys in different maps in the same map set, they can be committed together.

You can also use the PartitionableKey interface when groups of keys should be put on the same partition, but not necessarily during a single transaction. If keys should be hashed based on location, department, domain type, or some other type of identifier, children keys can be given a parent PartitionableKey.

For example, employees should hash to the same partition as their department. Each employee key would have a PartitionableKey object that belongs to the department map. Then, both the employee and department would hash to the same partition.

The PartitionableKey interface supplies one method, called ibmGetPartition. The object returned from this method must implement the hashCode method. The result returned from using the alternate hashCode will be used to route the key to a partition.

Example

See the following example key that demonstrates how to use the PartitionableKey interface and the hashCode method to clone an existing key, and route the resulting keys to the same partition.

```
package com.ibm.websphere.cjtester;

import java.io.Serializable;

import com.ibm.websphere.objectgrid.plugins.PartitionableKey;

public class RoutableKey implements Serializable, Cloneable, PartitionableKey {
    private static final long serialVersionUID = 1L;

    // The data that makes up the actual data object key.
    public final String realKey;

    // The key of the data object you want to use for routing.
    // This is typically the key of a parent object.
    public final Object keyToRouteWith;

    public RoutableKey(String realKey, Object keyToRouteWith) {
        super();
        this.realKey = realKey;
        this.keyToRouteWith = keyToRouteWith;
    }

    /**
     * Return the hashcode of the key we are using for routing.
     * If not supplied, eXtreme Scale will use the hashCode of THIS key.
     */
    public Object ibmGetPartition() {
        return new Integer(keyToRouteWith.hashCode());
    }

    @Override
    public RoutableKey clone() throws CloneNotSupportedException {
        return (RoutableKey) super.clone();
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + ((keyToRouteWith == null) ? 0 : keyToRouteWith.hashCode());
        result = prime * result + ((realKey == null) ? 0 : realKey.hashCode());
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true;
        if (obj == null) return false;
        if (getClass() != obj.getClass()) return false;
        RoutableKey other = (RoutableKey) obj;
        if (keyToRouteWith == null) {
            if (other.keyToRouteWith != null) return false;
        } else if (!keyToRouteWith.equals(other.keyToRouteWith)) return false;
        if (realKey == null) {
            if (other.realKey != null) return false;
        } else if (!realKey.equals(other.realKey)) return false;
        return true;
    }
}
```

Related concepts:
Tuning EntityManager interface performance
Caching objects and their relationships (EntityManager API)
Entity manager in a distributed environment
Interacting with EntityManager
EntityManager fetch plan support

Entity query queues
Related tasks:
Collocating multiple cache objects in the same partition
Related reference:
Entity performance instrumentation agent
Defining an entity schema
Entity listeners and callback methods
Entity listener examples
EntityTransaction interface
Related information:
Sample: Running Queries in Parallel using a ReduceGridAgent

Introduction to ObjectMap

The ObjectMap interface is used for transactional interaction between applications and BackingMaps.

Purpose

An ObjectMap instance is obtained from a Session object that corresponds to the current thread. The ObjectMap interface is the main vehicle that applications use to make changes to entries in a BackingMap.

Obtain an ObjectMap instance

An application gets an ObjectMap instance from a Session object using the Session.getMap(String) method. The following code snippet demonstrates how to obtain an ObjectMap instance:

```
ObjectGrid objectGrid = ...;
BackingMap backingMap = objectGrid.defineMap("mapA");
Session sess = objectGrid.getSession();
ObjectMap objectMap = sess.getMap("mapA");
```

Each ObjectMap instance corresponds to a particular Session object. Calling the getMap method multiple times on a particular Session object with the same BackingMap name always returns the same ObjectMap instance.

Automatically commit transactions

Operations against BackingMaps that use ObjectMaps and JavaMaps are performed most efficiently within a Session transaction. WebSphere® eXtreme Scale provides autocommit support when methods on the ObjectMap and JavaMap interfaces are called outside of a Session transaction. The methods start an implicit transaction, perform the requested operation, and commit the implicit transaction.

Method semantics

An explanation of the semantics behind each method on the ObjectMap and JavaMap interfaces follows.

containsKey method
: The containsKey method determines if a key has a value in the BackingMap or Loader. If null values are supported by an application, this method can be used to determine if a null reference that is returned from a get operation refers to a null value or indicates that the BackingMap and Loader do not contain the key.

flush method
: The flush method semantics are similar to the flush method on the Session interface. The notable difference is that the Session flush applies the current pending changes for all of the maps that are modified in the current session. With this method, only the changes in this ObjectMap instance are flushed to the loader.

get method
: The get method fetches the entry from the BackingMap instance. If the entry is not found in the BackingMap instance but a Loader is associated with the BackingMap instance, the BackingMap instance attempts to fetch the entry from the Loader. The getAll method is provided to allow batch fetch processing.

getForUpdate method
: The getForUpdate method is the same as the get method, but using the getForUpdate method tells the BackingMap and Loader that the intention is to update the entry. A Loader can use this hint to issue a SELECT for UPDATE query to a database backend. If a pessimistic locking strategy is defined for the BackingMap, the lock manager locks the entry. The getAllForUpdate method is provided to allow batch fetch processing.

insert method
: The insert method inserts an entry into the BackingMap and the Loader. Using this method tells the BackingMap and Loader that you want to insert an entry that did not previously exist. When you invoke this method on an existing entry, an exception occurs when the method is invoked or when the current transaction is committed.

invalidate method
: The semantics of the invalidate method depend on the value of the isGlobal parameter that is passed to the method. The invalidateAll method is provided to allow batch invalidate processing.
  Local invalidation is specified when the value false is passed as the isGlobal parameter of the invalidate method. Local invalidation discards any changes to the entry in the transaction cache. If the application issues a get method, the entry is fetched from the last committed value in the BackingMap. If no entry is present in the BackingMap, the entry is fetched from the last flushed or committed value in the Loader. When a transaction is committed, any entries that are marked as locally invalidated have no impact on the BackingMap. Any changes that were flushed to the Loader are still committed even if the entry was invalidated.

Global invalidation is specified when true is passed as the isGlobal parameter of the invalidate method. Global invalidation discards any pending changes to the entry in the transaction cache and bypasses the BackingMap value on subsequent operations that are performed on the entry. When a transaction is committed, any entries that are marked as globally invalidated are evicted from the BackingMap.

Consider the following use case for invalidation as an example: The BackingMap is backed by a database table that has an auto increment column. Increment columns are useful for assigning unique numbers to records. The application inserts an entry. After the insert, the application needs to know the sequence number for the inserted row. It knows that its copy of the object is old, so it uses global invalidation to get the value from the Loader. The following code demonstrates this use case:

```
Session sess = objectGrid.getSession();
ObjectMap map = sess.getMap("mymap");
sess.begin();
map.insert("Billy", new Person("Joe", "Bloggs", "Manhattan"));
sess.flush();
map.invalidate("Billy", true);
Person p = map.get("Billy");
System.out.println("Version column is: " + p.getVersion());
map.commit();
// Close the session (optional in Version 7.1.1 and later) for improved performance
session.close();
```

This code sample adds an entry for `Billy`. The version attribute of Person is set using an auto-increment column in the database. The application first performs an insert command. It then issues a flush, which causes the insert to be sent to the Loader and database. The database sets the version column to the next number in the sequence, which makes the Person object in the transaction outdated. To update the object, the application is globally invalidated. The next get method that is issued gets the entry from the Loader, ignoring the transaction value. The entry is fetched from the database with the updated version value.

put method
: The semantics of the put method are dependent on whether a previous get method was invoked in the transaction for the key. If the application issues a get operation that returns an entry that exists in the BackingMap or loader, the put method invocation is interpreted as an update and returns the previous value in the transaction. If a put method invocation ran without a previous get method invocation, or a previous get method invocation did not find an entry, the operation is interpreted as an insert. The semantics of the insert and update methods apply when the put operation is committed. The putAll method is provided to enable batch insert and update processing.

remove method
: The remove method removes the entry from the BackingMap and the Loader, if a Loader is plugged in. The value of the object that was removed is returned by this method. If the object does not exist, this method returns a null value. The removeAll method is provided to enable batch deletion processing without the return values.

setCopyMode method
: The setCopyMode method specifies a CopyMode value for this ObjectMap. With this method, an application can override the CopyMode value that is specified on the BackingMap. The specified CopyMode value is in effect until clearCopyMode method is invoked. Both methods are invoked outside of transactional bounds. A CopyMode value cannot be changed in the middle of a transaction.

touch method
: The touch method updates the last access time for an entry. This method does not retrieve the value from the BackingMap. Use this method in its own transaction. If the provided key does not exist in the BackingMap because of invalidation or removal, an exception occurs during commit processing.

update method
: The update method explicitly updates an entry in the BackingMap and the Loader. Using this method indicates to the BackingMap and Loader that you want to update an existing entry. An exception occurs if you invoke this method on an entry that does not exist when the method is invoked or during commit processing.

getIndex method
: The getIndex method attempts to obtain a named index that is built on the BackingMap. The index cannot be shared between threads and works on the same rules as a Session. The returned index object should be cast to the right application index interface such as the MapIndex interface, the MapRangeIndex interface, or a custom index interface.

clear method
: The clear method removes all cache entries from a map from all partitions. This operation is an auto-commit function, so no active transaction should be present when calling clear.
Note: The clear method only clears out the map on which it is called, leaving any related entity maps unaffected. This method does not invoke the Loader plug-in.

Related concepts:
Data invalidation
JMS event listener
Caching objects with no relationships involved (ObjectMap API)
Related tasks:
8.5+ Querying and invalidating data
Related information:
ObjectMap.invalidate method
EntityManager.invalidate method
ObjectGridEventListener interface
ObjectMap interface
BackingMap interface
JavaMap interface

ObjectMap and JavaMap

A JavaMap instance is obtained from an ObjectMap object. The JavaMap interface has the same method signatures as ObjectMap, but with different exception handling. JavaMap extends the java.util.Map interface, so all exceptions are instances of the java.lang.RuntimeException class. Because JavaMap extends the java.util.Map interface, it is easy to quickly use WebSphere® eXtreme Scale with an existing application that uses a java.util.Map interface for object caching.

Obtain a JavaMap instance

An application gets a JavaMap instance from an ObjectMap object using the ObjectMap.getJavaMap method. The following code snippet demonstrates how to obtain a JavaMap instance.

```
ObjectGrid objectGrid = ...;
BackingMap backingMap = objectGrid.defineMap("mapA");
Session sess = objectGrid.getSession();
ObjectMap objectMap = sess.getMap("mapA");
java.util.Map map = objectMap.getJavaMap();
JavaMap javaMap = (JavaMap) javaMap;
```

A JavaMap is backed by the ObjectMap from which it was obtained. Calling the getJavaMap method multiple times using a particular ObjectMap always returns the same JavaMap instance.

Methods

The JavaMap interface only supports a subset of the methods on the java.util.Map interface. The java.util.Map interface supports the following methods:

containsKey(java.lang.Object) method
get(java.lang.Object) method
put(java.lang.Object, java.lang.Object) method
putAll(java.util.Map) method
remove(java.lang.Object) method
clear()

All other methods inherited from the java.util.Map interface result in a java.lang.UnsupportedOperationException exception.
Related concepts:
Caching objects with no relationships involved (ObjectMap API)
Related information:
ObjectMap interface
BackingMap interface
JavaMap interface

Maps as FIFO queues

With WebSphere® eXtreme Scale, you can provide a first-in first-out (FIFO) queue-like capability for all maps. WebSphere eXtreme Scale tracks the insertion order for all maps. A client can ask a map for the next unlocked entry in a map in the order of insertion and lock the entry. This process allows multiple clients to consume entries from the map efficiently.

FIFO example

The following code snippet shows a client entering a loop to process entries from the map until the map is exhausted. The loop starts a transaction, then calls the ObjectMap.getNextKey(5000) method. This method returns the key of the next available unlocked entry and locks it. If the transaction is blocked for more than 5000 milliseconds, then the method returns null.

```
Session session = ...;
ObjectMap map = session.getMap("xxx");
// this needs to be set somewhere to stop this loop
boolean timeToStop = false;

while(!timeToStop)
{
  session.begin();
  Object msgKey = map.getNextKey(5000);
  if(msgKey == null)
  {
    // current partition is exhausted, call it again in
    // a new transaction to move to next partition
    session.rollback();
    continue;
  }
  Message m = (Message)map.get(msgKey);
  // now consume the message
  ...
  // need to remove it
  map.remove(msgKey);
  session.commit();
}
```

Local mode versus client mode

If the application is using a local core, that is, it is not a client, then the mechanism works as described previously.
For client mode, if the Java™ virtual machine (JVM) is a client, then the client initially connects to a random partition primary. If no work exists in that partition, then the client moves to the next partition to look for work. The client either finds a partition with entries or loops around to the initial random partition. If the client loops around to the initial partition, then it returns a null value to the application. If the client finds a partition with a map that has entries, then it consumes

entries from there until no entries are available for the timeout period. After the timeout passes, then null is returned. This action means that when null is returned and a partitioned map is used, then it you should start a new transaction and resume listening. The previous code sample fragment has this behavior.

Example

When you are running as a client and a key is returned, that transaction is now bound to the partition with the entry for that key. If you do not want to update any other maps during that transaction, then a problem does not exist. If you do want to update, then you can only update maps from the same partition as the map from which you got the key. The entry that is returned from the getNextKey method needs to give the application a way to discover relevant data in that partition. As an example, if you have two maps; one for events and another for jobs that the events impact. You define the two maps with the following entities:

```
Job.java
package tutorial.fifo;

import com.ibm.websphere.projector.annotations.Entity;
import com.ibm.websphere.projector.annotations.Id;

@Entity
public class Job
{
        @Id String jobId;

        int jobState;
}

JobEvent.java
package tutorial.fifo;

import com.ibm.websphere.projector.annotations.Entity;
import com.ibm.websphere.projector.annotations.Id;
import com.ibm.websphere.projector.annotations.OneToOne;

@Entity
public class JobEvent
{
        @Id String eventId;
        @OneToOne Job job;
}
```

The job has as ID and state, which is an integer. Suppose you want to increment the state whenever an event arrived. The events are stored in the JobEvent Map. Each entry has a reference to the job the event concerns. The code for the listener to do this looks like the following example:

```
JobEventListener.java
package tutorial.fifo;

import com.ibm.websphere.objectgrid.ObjectGridException;
import com.ibm.websphere.objectgrid.ObjectMap;
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.em.EntityManager;

public class JobEventListener
{
        boolean stopListening;

        public synchronized void stopListening()
        {
                stopListening = true;
        }

        synchronized boolean isStopped()
        {
                return stopListening;
        }

        public void processJobEvents(Session session)
                throws ObjectGridException
        {
                EntityManager em = session.getEntityManager();
                ObjectMap jobEvents = session.getMap("JobEvent");
                while(!isStopped())
                {
                        em.getTransaction().begin();

                        Object jobEventKey = jobEvents.getNextKey(5000);
                        if(jobEventKey == null)
                        {
                                em.getTransaction().rollback();
                                continue;
                        }
                        JobEvent event = (JobEvent)em.find(JobEvent.class, jobEventKey);
                        // process the event, here we just increment the
                        // job state
                        event.job.jobState++;
                        em.getTransaction().commit();
                }
        }
}
```

The listener is started on a thread by the application. The listener runs until the stopListening method is called. The processJobEvents method is run on the thread until the stopListening method is called. The loop blocks waiting for an eventKey from the JobEvent Map and then uses the EntityManager to access the event object, dereference to the job and increment the state.

The EntityManager API does not have a getNextKey method, but the ObjectMap does. So, the code uses the ObjectMap for JobEvent to get the key. If a map is used with entities then it does not store objects anymore. Instead, it stores Tuples; a Tuple object for the key and a Tuple object for the value. The EntityManager.find method accepts a Tuple for the key.

The code to create an event looks like the following example:

```
em.getTransaction().begin();
Job job = em.find(Job.class, "Job Key");
JobEvent event = new JobEvent();
event.id = Random.toString();
event.job = job;
em.persist(event); // insert it
em.getTransaction().commit();
```

You find the job for the event, construct an event, point it to the job, insert it in the JobEvent Map and commit the transaction.

Loaders and FIFO maps

If you want to back a map that is used as a FIFO queue with a Loader, then you might need to do some additional work. If the order of the entries in the map is not a concern, you have no extra work. If the order is important, then you need to add a sequence number to all of the inserted records when you are persisting the records to the backend. The preload mechanism should be written to insert the records on startup using this order.

Related concepts:
Caching objects with no relationships involved (ObjectMap API)
Related information:
ObjectMap interface
BackingMap interface
JavaMap interface

Caching objects and their relationships (EntityManager API)

Most cache products use map-based APIs to store data as key-value pairs. The ObjectMap API and the dynamic cache in WebSphere® Application Server, among others, use this approach. However, map-based APIs have limitations. The EntityManager API simplifies the interaction with the data grid by providing an easy way to declare and interact with a complex graph of related objects.

Map-based API limitations

If you are using a map-based API, such as the dynamic cache in WebSphere Application Server or the ObjectMap API, take the following limitations into consideration:

- Indexes and queries must use reflection to query fields and properties in cache objects.
- Custom data serialization is required to achieve performance for complex objects.
- It is difficult to work with graphs of objects. You must develop the application to store artificial references between objects and manually join the objects.

Benefits of the EntityManager API

The EntityManager API uses the existing map-based infrastructure, but it converts entity objects to and from tuples before storing or reading them from the map. An entity object is transformed into a key tuple and a value tuple, which are then stored as key-value pairs. A tuple is an array of primitive attributes.

This set of APIs follows the Plain Old Java™ Object (POJO) style of programming that is adopted by most frameworks.

- Relationship management
  Object-oriented languages such as Java, and relational databases support relationships or associations. Relationships decrease the amount of storage through the use of object references or foreign keys.
- Defining an entity schema
  An ObjectGrid can have any number of logical entity schemas. Entities are defined using annotated Java classes, XML, or a combination of both XML and Java classes. Defined entities are then registered with an eXtreme Scale server and bound to BackingMaps, indexes and other plug-ins.
- Entity manager in a distributed environment
  You can use EntityManager API with a local ObjectGrid or in a distributed eXtreme Scale environment. The main difference is how you connect to this remote environment. After you establish a connection, there is no difference between using a Session object or using the EntityManager API.
- Interacting with EntityManager
  Applications typically first obtain an ObjectGrid reference, and then a Session from that reference for each thread. Sessions cannot be shared between threads. An extra method on Session, the getEntityManager method, is available. This method returns a reference to an entity manager to use for this thread. The EntityManager interface can replace the Session and ObjectMap interfaces for all applications. You can use these EntityManager APIs if the client has access to the defined entity classes.
- EntityManager fetch plan support
  A FetchPlan is the strategy that the entity manager uses for retrieving associated objects if the application needs to access relationships.
- Entity query queues
  Query queues allow applications to create a queue qualified by a query in the server-side or local eXtreme Scale over an entity. Entities from the query result are stored in this queue. Currently, query queue is only supported in a map that is using the pessimistic lock strategy.

- EntityTransaction interface
  You can use the EntityTransaction interface to demarcate transactions.

Related concepts:
Tuning EntityManager interface performance
Entity manager in a distributed environment
Interacting with EntityManager
EntityManager fetch plan support
Entity query queues
Routing cache objects to the same partition
Related tasks:
Tutorial: Storing order information in entities
Collocating multiple cache objects in the same partition
Related reference:
Entity performance instrumentation agent
Defining an entity schema
Entity listeners and callback methods
Entity listener examples
EntityTransaction interface
Related information:
Sample: Running Queries in Parallel using a ReduceGridAgent

Relationship management

Object-oriented languages such as Java™, and relational databases support relationships or associations. Relationships decrease the amount of storage through the use of object references or foreign keys.

When you are using relationships in a data grid, the data must be organized in a constrained tree. One root type must exist in the tree and all children must be associated to only one root. For example: Department can have many Employees and an Employee can have many Projects. But a Project cannot have many Employees that belong to different departments. Once a root is defined, all access to that root object and its descendants are managed through the root. WebSphere® eXtreme Scale uses the hash code of the root object's key to choose a partition. For example:

```
partition = (hashCode MOD numPartitions).
```

When all of the data for a relationship is tied to a single object instance, the entire tree can be collocated in a single partition and can be accessed very efficiently using one transaction. If the data spans multiple relationships, then multiple partitions must be involved which involves additional remote calls, which can lead to performance bottlenecks.

Reference data

Some relationships include look-up or reference data such as: `CountryName`. For look-up or reference data, the data must exist in every partition. The data can be accessed by any root key and the same result is returned. Reference data such as this should only be used in cases where the data is fairly static. Updating this data can be expensive because the data needs to be updated in every partition. The DataGrid API is a common technique to keeping reference data up-to-date.

Costs and benefits of normalization

Normalizing the data using relationships can help reduce the amount of memory used by the data grid since duplication of data is decreased. However, in general, the more relational data that is added, the less it will scale out. When data is grouped together, it becomes more expensive to maintain the relationships and to keep the sizes manageable. Since the grid partitions data based on the key of the root of the tree, the size of the tree isn't taken into account. Therefore, if you have a lot of relationships for one tree instance, the data grid may become unbalanced, causing one partition to hold more data than the others.

When the data is denormalized or flattened, the data that would normally be shared between two objects is instead duplicated and each table can be partitioned independently, providing a much more balanced data grid. Although this increases the amount of memory used, it allows the application to scale since a single row of data can be accessed that has all of the necessary data. This is ideal for read-mostly grids since maintaining the data becomes more expensive.

For more information, see Classifying XTP systems and scaling.

Managing relationships using the data access APIs

The ObjectMap API is the fastest, most flexible and granular of the data access APIs, providing a transactional, session-based approach at accessing data in the grid of maps. The ObjectMap API allows clients to use common CRUD (create, read, update and delete) operations to manage key-value pairs of objects in the distributed data grid.

When using the ObjectMap API, object relationships must be expressed by embedding the foreign key for all relationships in the parent object.

An example follows.

```
public class Department {
        Collection<String> employeeIds;
}
```

The EntityManager API simplifies relationship management by extracting the persistent data from the objects including the foreign keys. When the object is later retrieved from the data grid, the relationship graph is rebuilt, as in the following example.

```
@Entity
public class Department {
        Collection<String> employees;
}
```

The EntityManager API is very similar to other Java object persistence technologies such as JPA and Hibernate in that it synchronizes a graph of managed Java object instances with the persistent store. In this case, the persistent store is an eXtreme Scale data grid, where each entity is represented as a map and the map contains the entity data rather than the object instances.

Related concepts:
Entity manager in a distributed environment
Interacting with EntityManager
EntityManager fetch plan support
Entity query queues
Related reference:
Defining an entity schema
EntityTransaction interface

Defining an entity schema

An ObjectGrid can have any number of logical entity schemas. Entities are defined using annotated Java™ classes, XML, or a combination of both XML and Java classes. Defined entities are then registered with an eXtreme Scale server and bound to BackingMaps, indexes and other plug-ins.

When designing an entity schema, you must complete the following tasks:

1. Define the entities and their relationships.
2. Configure eXtreme Scale.
3. Register the entities.
4. Create entity-based applications that interact with the eXtreme Scale EntityManager APIs.

Entity schema configuration

An entity schema is a set of entities and the relationships between the entities. In an eXtreme Scale application with multiple partitions, the following restrictions and options apply to entity schemas:

- Each entity schema must have a single root defined. This is known as the schema root.
- All the entities for a given schema must be in the same map set, which means that all the entities that are reachable from a schema root with key or non-key relationships must be defined in the same map set as the schema root.
- Each entity can belong to only one entity schema.
- Each eXtreme Scale application can have multiple schemas.

Entities are registered with an ObjectGrid instance before it is initialized. Each defined entity must be uniquely named and is automatically bound to an ObjectGrid BackingMap of the same name. The initialization method varies depending on the configuration you are using:
Local eXtreme Scale configuration

If you are using a local ObjectGrid, you can programmatically configure the entity schema. In this mode, you can use the ObjectGrid.registerEntities methods to register annotated entity classes or an entity metadata descriptor file.
Distributed eXtreme Scale configuration

If you are using a distributed eXtreme Scale configuration, you must provide an entity metadata descriptor file with the entity schema.
For more details, see Entity manager in a distributed environment.

Entity requirements

Entity metadata is configured using Java class files, an entity descriptor XML file or both. At minimum, the entity descriptor XML is required to identify which eXtreme Scale BackingMaps are to be associated with entities. The persistent attributes of the entity and its relationships to other entities are described in either an annotated Java class (entity metadata class) or the entity descriptor XML file. The entity metadata class, when specified, is also used by the EntityManager API to interact with the data in the grid.

An eXtreme Scale grid can be defined without providing any entity classes. This can be beneficial when the server and client are interacting directly with the tuple data stored in the underlying maps. Such entities are defined completely in the entity descriptor XML file and are referred to as classless entities.

Classless entities

Classless entities are useful when it is not possible to include application classes in the server or client classpath. Such entities are defined in the entity metadata descriptor XML file, where the class name of the entity is specified using a classless entity identifier in the form: @<entity identifier>. The @ symbol identifies the entity as classless and is used for mapping associations between entities. See the "Classless entity metadata" figure an example of an entity metadata descriptor XML file with two classless entities defined.

If an eXtreme Scale server or client does not have access to the classes, they can still use the EntityManager API using classless entities. Common use cases include the following:

- The eXtreme Scale container is hosted in a server that does not allow application classes in the classpath. In this case, the clients can still access the grid using the EntityManager API from a client, where the classes are allowed.

- The eXtreme Scale client does not require access to the entity classes because the client is either using a non-Java client, such as the eXtreme Scale REST data service or the client is accessing the tuple data in the grid using the ObjectMap API.

If the entity metadata is compatible between the client and server, entity metadata can be created using entity metadata classes, an XML file, or both.

For example, the "Programmatic entity class" in the following figure is compatible with the classless metadata code in the next section.

```
Programmatic entity class
@Entity
public class Employee {
    @Id long serialNumber;
    @Basic byte[] picture;
    @Version int ver;
    @ManyToOne(fetch=FetchType.EAGER, cascade=CascadeType.PERSIST)
    Department department;
}

@Entity
public static class Department {
    @Id int number;
    @Basic String name;
    @OneToMany(fetch=FetchType.LAZY, cascade=CascadeType.ALL, mappedBy="department")
    Collection<Employee> employees;
}
```

Classless fields, keys, and versions

As previously mentioned, classless entities are configured completely in the entity XML descriptor file. Class-based entities define their attributes using Java fields, properties and annotations. So classless entities need to define key and attribute structure in the entity XML descriptor with the `<basic>` and `<id>` tags.

```
Classless entity metadata
<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings xmlns="http://ibm.com/ws/projector/config/emd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://ibm.com/ws/projector/config/emd ./emd.xsd">

<entity class-name="@Employee" name="Employee">
    <attributes>
        <id name="serialNumber" type="long"/>
        <basic name="firstName" type="java.lang.String"/>
        <basic name="picture" type="[B"/>
        <version name="ver" type="int"/>
        <many-to-one
            name="department"
            target-entity="@Department"
            fetch="EAGER"">
                <cascade><cascade-persist/></cascade>
        </many-to-one>
    </attributes>
</entity>

<entity class-name="@Department" name="Department" >
    <attributes>
        <id name="number" type="int"/>
        <basic name="name" type="java.lang.String"/>
        <version name="ver" type="int"/>
        <one-to-many
            name="employees"
            target-entity="@Employee"
            fetch="LAZY"
            mapped-by="department">
            <cascade><cascade-all/></cascade>
        </one-to-many>
    </attributes>
</entity>
```

Note that each entity above has an <id> element. A classless entity must have either one or more of an <id> element defined, or a single-valued association that represents the key for the entity. The fields of the entity are represented by <basic> elements. The <id>, <version>, and <basic> elements require a name and type in classless entities. See the following supported attribute types section for details on supported types.

Entity class requirements

Class-based entities are identified by associating various metadata with a Java class. The metadata can be specified usingJava Platform, Standard Edition Version 5 annotations, an entity metadata descriptor file, or a combination of annotations and the descriptor file. Entity classes must meet the following criteria:

- The `@Entity` annotation is specified in the entity XML descriptor file.
- The class has a public or protected no-argument constructor.
- It must be a top-level class. Interfaces and enumerated types are not valid entity classes.
- Cannot use the final keyword.
- Cannot use inheritance.
- Must have a unique name and type for each ObjectGrid instance.

Entities all have a unique name and type. The name, if using annotations, is the simple (short) name of the class by default, but can be overridden using the name attribute of the `@Entity` annotation.

Persistent attributes

The persistent state of an entity is accessed by clients and the entity manager by using either fields (instance variables) or Enterprise JavaBeans-style property accessors. Each entity must define either field- or property-based access. Annotated entities are field-access if the class fields are annotated and are property-access if the getter method of the property is annotated. A mixture of field- and property-access is not allowed. If the type cannot be automatically determined, the accessType attribute on the `@Entity` annotation or equivalent XML can be used to identify the access type.

Persistent fields
: Field-access entity instance variables are accessed directly from the entity manager and clients. Fields that are marked with the transient modifier or transient annotation are ignored. Persistent fields must not have final or static modifiers.

Persistent properties
: Property-access entities must adhere to the JavaBeans signature conventions for read and write properties. Methods that do not follow JavaBeans conventions or have the Transient annotation on the getter method are ignored. For a property of type `T`, there must be a getter method getProperty which returns a value of type T and a void setter method setProperty(T). For boolean types, the getter method can be expressed as isProperty, returning true or false. Persistent properties cannot have the static modifier.

Supported attribute types
: The following persistent field and property types are supported:

  - Java primitive types including wrappers
  - java.lang.String
  - java.math.BigInteger
  - java.math.BigDecimal
  - java.util.Date
  - java.util.Calendar
  - java.sql.Date
  - java.sql.Time
  - java.sql.Timestamp
  - byte[]
  - java.lang.Byte[]
  - char[]
  - java.lang.Character[]
  - enum

  User serializable attribute types are supported but have performance, query and change-detection limitations. Persistent data that cannot be proxied, such as arrays and user serializable objects, must be reassigned to the entity if altered.

Serializable attributes are represented in the entity descriptor XML file using the class name of the object. If the object is an array, the data type is represented using the Java internal form. For example, if an attribute data type is java.lang.Byte[][], the string representation is [[Ljava.lang.Byte;

User serializable types should adhere to the following best practices:

- Implement high performance serialization methods. Implement the java.lang.Cloneable interface and public clone method.
- Implement the java.io.Externalizable interface.
- Implement equals and hashCode

Entity associations

Bi-directional and uni-directional entity associations, or relationships between entities can be defined as one-to-one, many-to-one, one-to-many and many-to-many. The entity manager automatically resolves the entity relationships to the appropriate key references when storing the entities.
The eXtreme Scale grid is a data cache and does not enforce referential integrity like a database. Although relationships allow cascading persist and remove operations for child entities, it does not detect or enforce broken links to objects. When removing a child object, the reference to that object must be removed from the parent.

If you define a bi-directional association between two entities, you must identify the owner of the relationship. In a to-many association, the many side of the relationship is always the owning side. If ownership cannot be determined automatically, then the mappedBy attribute of the annotation, or XML equivalent, must be specified. The mappedBy attribute identifies the field in the target entity that is the owner of the relationship. This attribute also helps identify the related fields when there are multiple attributes of the same type and cardinality.

Single-valued associations

One-to-one and many-to-one associations are denoted using the @OneToOne and @ManyToOne annotations or equivalent XML attributes. The target entity type is determined by the attribute type. The following example defines a uni-directional association between Person and Address. The Customer entity has a reference to one Address entity. In this case, the association could also be many-to-one since there is no inverse relationship.

```
@Entity
public class Customer {
  @Id id;
  @OneToOne Address homeAddress;
}

@Entity
public class Address{
  @Id id
  @Basic String city;
}
```

To specify a bi-directional relationship between the Customer and Address classes, add a reference to the Customer class from the Address class and add the appropriate annotation to mark the inverse side of the relationship. Because this association is one-to-one, you have to specify an owner of the relationship using the mappedBy attribute on the @OneToOne annotation.

```
@Entity
public class Address{
  @Id id
  @Basic String city;
  @OneToOne(mappedBy="homeAddress") Customer customer;
}
```

Collection-valued associations

One-to-many and many-to-many associations are denoted using the `@OneToMany` and `@ManyToMany` annotations or equivalent XML attributes. All many relationships are represented using the types: java.util.Collection, java.util.List or java.util.Set. The target entity type is determined by the generic type of the Collection, List or Set or explicitly using the targetEntity attribute on the `@OneToMany` or `@ManyToMany` annotation (or XML equivalent).
In the previous example, it is not practical to have one address object per customer because many customers might share an address or might have multiple addresses. This situation is better solved using a many association:

```
@Entity
public class Customer {
  @Id id;
  @ManyToOne Address homeAddress;
  @ManyToOne Address workAddress;
}

@Entity
public class Address{
  @Id id
  @Basic String city;
  @OneToMany(mappedBy="homeAddress") Collection<Customer> homeCustomers;


  @OneToMany(mappedBy="workAddress", targetEntity=Customer.class)
               Collection workCustomers;
}
```

In this example, two different relationships exist between the same entities: a Home and Work address relationship. A non-generic Collection is used for the workCustomers attribute to demonstrate how to use the targetEntity attribute when generics are not available.
Classless associations

Classless entity associations are defined in the entity metadata descriptor XML file similar to how class-based associations are defined. The only difference is that instead of the target entity pointing to an actual class, it points to the classless entity identifier used for the class name of the entity.

An example follows:

```
<many-to-one name="department" target-entity="@Department" fetch="EAGER">
      <cascade><cascade-all/></cascade>
</many-to-one>
<one-to-many name="employees" target-entity="@Employee" fetch="LAZY">
      <cascade><cascade-all/></cascade>
</one-to-many>
```

Primary keys

All entities must have a primary key, which can be a simple (single attribute) or composite (multiple attribute) key. The key attributes are denoted using the Id annotation or defined in the entity XML descriptor file. Key attributes have the following requirements:

- The value of a primary key cannot change.
- A primary key attribute should be one of the following types: Java primitive type and wrappers, java.lang.String, java.util.Date or java.sql.Date.
- A primary key can contain any number of single-valued associations. The target entity of the primary key association must not have an inverse association directly or indirectly to the source entity.

Composite primary keys can optionally define a primary key class. An entity is associated with a primary key class using the `@IdClass` annotation or the entity XML descriptor file. An `@IdClass` annotation is useful in conjunction with the EntityManager.find method.
Primary key classes have the following requirements:

- It should be public with a no-argument constructor.
- The access type of the primary key class is determined by the entity that declares the primary key class.
- If property-access, the properties of the primary key class must be public or protected.
- The primary key fields or properties must match the key attribute names and types defined in the referencing entity.
- Primary key classes must implement the equals and hashCode methods.

An example follows:

```
@Entity
@IdClass(CustomerKey.class)
public class Customer {
    @Id @ManyToOne Zone zone;
    @Id int custId;
    String name;
    ...
}

@Entity
public class Zone{
    @Id String zoneCode;
    String name;
}
```

```
public class CustomerKey {
    Zone zone;
    int custId;

    public int hashCode() {...}
    public boolean equals(Object o) {...}
}
```

Classless primary keys

Classless entities are required to either have at least one <id> element or an association in the XML file with the attribute `id=true`. An example of both would look like the following:

```
<id name="serialNumber" type="int"/>
<many-to-one name="department" target-entity="@Department"
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsentmgrschema_true">
<cascade><cascade-all/></cascade>
</many-to-one>
```

Remember:
The <id-class> XML tag is not supported for classless entities.

Entity proxies and field interception

Entity classes and mutable supported attribute types are extended by proxy classes for property-access entities and bytecode-enhanced for field-access entities. All access to the entity, even by internal business methods and the equals methods, must use the appropriate field or property access methods. Proxies and field interceptors are used to allow the entity manager to track the state of the entity, determine if the entity has changed, and improve performance.

Attention: When using property-access entities, the equals method should use the instanceof operator for comparing the current instance to the input object. All introspection of the target object should be through the properties of the object, not the fields themselves, because the object instance will be the proxy.
Related concepts:
Relationship management
Entity manager in a distributed environment
Interacting with EntityManager
EntityManager fetch plan support
Entity query queues
Tuning EntityManager interface performance
Caching objects and their relationships (EntityManager API)
Entity manager in a distributed environment
Interacting with EntityManager
EntityManager fetch plan support
Entity query queues
Routing cache objects to the same partition
Related tasks:
Tutorial: Storing order information in entities
Collocating multiple cache objects in the same partition
Related reference:
EntityTransaction interface
Related information:
Sample: Running Queries in Parallel using a ReduceGridAgent

Entity manager in a distributed environment

You can use EntityManager API with a local ObjectGrid or in a distributed eXtreme Scale environment. The main difference is how you connect to this remote environment. After you establish a connection, there is no difference between using a Session object or using the EntityManager API.

Required configuration files

The following XML configuration files are required:

- ObjectGrid descriptor XML file
- Entity descriptor XML file
- Deployment or data grid descriptor XML file

These files specify the entities and BackingMaps that a server hosts.

The entity metadata descriptor file contains a description of the entities that are used. At minimum, you must specify the entity class and name. If you are running in a Java™ Platform, Standard Edition 5 environment, eXtreme Scale automatically reads the entity class and its annotations. You can define additional XML attributes if the entity class has no annotations or if you are required to override the class attributes. If you are registering the entities classless , provide all of entity information in the XML file only.

You can use the following XML configuration snippet to define a data grid with entities. In this snippet, the server creates an ObjectGrid with the name `bookstore` and an associated backing map with the name `order`.The objectgrid.xml file snippet refers to the entity.xml file. In this case, the entity.xml file contains one entity, the Order entity.

```
objectgrid.xml
<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="bookstore" entityMetadataXMLFile="entity.xml">
     <backingMap name="Order"/>
    </objectGrid>
</objectGrids>

</objectGridConfig>
```

This objectgrid.xml file specifies the entity.xml file by including the entityMetadataXMLFile attribute. The value can be a relative directory or an absolute path.

- **For a relative directory:** Specify the location relative to the location of the objectgrid.xml file.
- **For an absolute path:** Specify the location with a URL format, such as file:// or http://.

An example of the entity.xml file follows:

```
entity.xml
<entity-mappings xmlns="http://ibm.com/ws/projector/config/emd"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://ibm.com/ws/projector/config/emd ./emd.xsd">
        <entity class-name="com.ibm.websphere.tutorials.objectgrid.em.
                            distributed.step1.Order" name="Order"/>
</entity-mappings>
```

This example assumes that the Order class would have the orderNumber and desc fields annotated similarly.
An equivalent classless entity.xml file would be as follows:

```
classless entity.xml
<entity-mappings xmlns="http://ibm.com/ws/projector/config/emd"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://ibm.com/ws/projector/config/emd ./emd.xsd">
        <entity class-name="@Order " name="Order">
            <description>"Entity named: Order"</description>
        <attributes>
            <id name="orderNumber" type="int"/>
            <basic name="desc" type="java.lang.String"/>
        </attributes>
        </entity>
</entity-mappings>
```

For information about starting servers, see Starting and stopping stand-alone servers. You use both the deployment.xml and objectgrid.xml files to start the catalog server.

Connecting to a distributed eXtreme Scale server

The following code enables the connect mechanism for a client and server on the same computer:

```
String catalogEndpoints="localhost:2809";
URL clientOverrideURL= new URL("file:etc/emtutorial/distributed/step1/objectgrid.xml");
ClientClusterContext clusterCtx = ogMgr.connect(catalogEndpoints, null, clientOverrideURL);
ObjectGrid objectGrid=ogMgr.getObjectGrid(clusterCtx, "bookstore");
```

In the preceding code snippet, note the reference to the remote eXtreme Scale server. After you establish a connection , you can invoke EntityManager API methods such as persist, update, remove and find.
Attention: When you are using entities, pass the client override ObjectGrid descriptor XML file to the connect method. If a null value is passed to the clientOverrideURL property and the client has a different directory structure than the server, then the client might fail to locate the ObjectGrid or entity descriptor XML files. At minimum, the ObjectGrid and entity XML files for the server can be copied to the client.
Previously, using entities on an ObjectGrid client required you to make the ObjectGrid XML and entity XML available to the client in one of the following two ways:

1. Pass an overriding ObjectGrid XML configuration to the ObjectGridManager.connect(String catalogServerEndpoints, ClientSecurityConfiguration securityProps, URL overRideObjectGridXml) method.

   ```
   String catalogEndpoints="myHost:2809";
   URL clientOverrideURL= new URL("file:etc/emtutorial/distributed/step1/objectgrid.xml");
   ClientClusterContext clusterCtx = ogMgr.connect(catalogEndpoints, null, clientOverrideURL);
   ObjectGrid objectGrid=ogMgr.getObjectGrid(clusterCtx, "bookstore");
   ```

2. Pass null for the override file and ensure that the ObjectGrid XML and referenced entity XML are available to the client on the same path as on the server.

   ```
   String catalogEndpoints="myHost:2809";
   ClientClusterContext clusterCtx = ogMgr.connect(catalogEndpoints, null, null);
   ObjectGrid objectGrid=ogMgr.getObjectGrid(clusterCtx, "bookstore");
   ```

The XML files were required regardless of whether or not you wanted to use subset entities on the client side. These files are no longer required to use the entities as defined by the server. Instead, pass null as the overRideObjectGridXml parameter as in option 2 of the previous section. If the XML file is not found on the same path set on the server, the client uses the entity configuration on the server.

However, if you use subset entities on the client, you must provide an overriding ObjectGrid XML as in option 1.

Client and server-side schema

The server-side schema defines the type of data stored in the maps on a server. The client-side schema is a mapping to application objects from the schema on the server. For example, you might have the following server-side schema:

```
@Entity
class ServerPerson
{
  @Id String ssn;
  String firstName;
  String surname;
  int age;
  int salary;
}
```

A client can have an object annotated as in the following example:

```
@Entity(name="ServerPerson")
class ClientPerson
{
  @Id @Basic(alias="ssn") String socialSecurityNumber;
  String surname;
}
```

This client then takes a server-side entity and projects the subset of the entity into the client object. This projection leads to bandwidth and memory savings on a client because the client has only the information it needs instead of all of the information that is in the server-side entity. Different applications can use their own objects instead of forcing all applications to share a set of classes for data access.
The client-side entity descriptor XML file is required in the following cases: if the server is running with class-based entities while the client side is running classless; or if the server is classless and the client uses class-based entities. A classless client mode allows the client to still run entity queries without having access to the physical classes. Assuming the server has registered the ServerPerson entity above, the client would override the data grid with an entity.xml file such as follows:

```
<entity-mappings xmlns="http://ibm.com/ws/projector/config/emd"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://ibm.com/ws/projector/config/emd ./emd.xsd">
        <entity class-name="@ServerPerson" name="Order">
            <description>"Entity named: Order"</description>
        <attributes>
            <id name="socialSecurityNumber" type="java.lang.String"/>
            <basic name="surname" type="java.lang.String"/>
        </attributes>
        </entity>
</entity-mappings>
```

This file achieves an equivalent subset entity on the client, without requiring the client to provide the actual annotated class. If the server is classless, and the client is not classless, then the client provides an overriding entity descriptor XML file. This entity descriptor XML file contains an override to the class file reference.

Referencing the schema

If your application is running in Java SE 5, then the application can be added to the objects by using annotations. The EntityManager can read the schema from the annotations on those objects. The application provides the eXtreme Scale run time with references to these objects using the entity.xml file, which is referenced from the objectgrid.xml file. The entity.xml file lists all the entities, each of which is associated with either a class or a schema. If a proper class name is specified, then the application attempts to read the Java SE 5 annotations from those classes to determine the schema. If you do not annotate the class file or specify a classless identifier as the class name, then the schema is taken from the XML file. The XML file is used to specify all the attributes, keys, and relationships for each entity.
A local data grid does not need XML files. The program can obtain an ObjectGrid reference and invoke the ObjectGrid.registerEntities method to specify a list of Java SE 5 annotated classes or an XML file.

The run time uses the XML file or a list of annotated classes to find entity names, attribute names and types, key fields and types, and relationships between entities. If eXtreme Scale is running on a server or in stand-alone mode, then it automatically makes a map named after each entity. These maps can be customized further using the objectgrid.xml file or APIs set either by the application or injection frameworks such as Spring.

Entity metadata descriptor file

See emd.xsd file for more information about the metadata descriptor file.
Related concepts:
Relationship management
Interacting with EntityManager
EntityManager fetch plan support
Entity query queues
Tuning EntityManager interface performance
Caching objects and their relationships (EntityManager API)
Interacting with EntityManager
EntityManager fetch plan support
Entity query queues
Routing cache objects to the same partition
Related tasks:
Tutorial: Storing order information in entities
Collocating multiple cache objects in the same partition
Related reference:
Defining an entity schema

EntityTransaction interface
Entity performance instrumentation agent
Defining an entity schema
Entity listeners and callback methods
Entity listener examples
EntityTransaction interface
Related information:
Sample: Running Queries in Parallel using a ReduceGridAgent

Interacting with EntityManager

Applications typically first obtain an ObjectGrid reference, and then a Session from that reference for each thread. Sessions cannot be shared between threads. An extra method on Session, the getEntityManager method, is available. This method returns a reference to an entity manager to use for this thread. The EntityManager interface can replace the Session and ObjectMap interfaces for all applications. You can use these EntityManager APIs if the client has access to the defined entity classes.

Obtaining an EntityManager instance from a session

The getEntityManager method is available on a Session object. The following code example illustrates how to create a local ObjectGrid instance and access the EntityManager. See the EntityManager interface in the API documentation for details about all the supported methods.

```
ObjectGrid og =
ObjectGridManagerFactory.getObjectGridManager().createObjectGrid("intro-grid");
Session s = og.getSession();
EntityManager em = s.getEntityManager();
```

A one-to-one relationship exists between the Session object and EntityManager object. You can use the EntityManager object more than once.

Persisting an entity

Persisting an entity means saving the state of a new entity in an ObjectGrid cache. After the persist method is called, the entity is in the managed state. Persist is a transactional operation, and the new entity is stored in the ObjectGrid cache after the transaction commits.

Every entity has a corresponding BackingMap in which the tuples are stored. The BackingMap has the same name as the entity, and is created when the class is registered. The following code example demonstrates how to create an Order object by using the persist operation.

```
Order order = new Order(123);
em.persist(order);
order.setX();
...
```

The Order object is created with the key 123, and the object is passed to the persist method. You can continue to modify the state of the object before you commit the transaction.
Important: The preceding example does not include any required transactional boundaries, such as begin and commit. See the Tutorial: Storing order information in entities for more information.

Finding an entity

You can locate the entity in the ObjectGrid cache with the find method by providing a key after the entity is stored in the cache. This method does not require any transactional boundary, which is useful for read-only semantics. The following example illustrates that only one line of code is needed to locate the entity.

```
Order foundOrder = (Order)em.find(Order.class, new Integer(123));
```

Removing an entity

The remove method, like the persist method, is a transactional operation. The following example shows the transactional boundary by calling the begin and commit methods.

```
em.getTransaction().begin();
Order foundOrder = (Order)em.find(Order.class, new Integer(123));
em.remove(foundOrder );
em.getTransaction().commit();
```

The entity must first be managed before it can be removed, which you can accomplish by calling the find method within the transactional boundary. Then call the remove method on the EntityManager interface.

Invalidating an entity

The invalidate method behaves much like the remove method, but does not invoke any Loader plug-ins. Use this method to remove entities from the ObjectGrid, but to preserve them in the backend data store.

```
em.getTransaction().begin();
Order foundOrder = (Order)em.find(Order.class, new Integer(123));
em.invalidate(foundOrder );
em.getTransaction().commit();
```

The entity must first be managed before it can be invalidated, which you can accomplish by calling the find method within the transactional boundary. After you call the find method, you can call the invalidate method on the EntityManager interface.

Updating an entity

The update method is also a transactional operation. The entity must be managed before any updates can be applied.

```
em.getTransaction().begin();
Order foundOrder = (Order)em.find(Order.class, new Integer(123));
foundOrder.date = new Date(); // update the date of the order
em.getTransaction().commit();
```

In the preceding example, the persist method is not called after the entity is updated. The entity is updated in the ObjectGrid cache when the transaction is committed.

Queries and query queues

With the flexible query engine, you can retrieve entities by using EntityManager API. Create SELECT type queries over an entity or Object-based schema by using the ObjectGrid query language. Query interface explains in detail how you can run the queries by using the EntityManager API. See the Query API for more information about using queries.

An entity QueryQueue is a queue-like data structure associated with an entity query. It selects all the entities that match the WHERE condition on the query filter and puts the result entities in a queue. Clients can then iteratively retrieve entities from this queue. See Entity query queues for more information.

- Entity listeners and callback methods
  Applications can be notified when the state of an entity transitions from state to state. Two callback mechanisms exist for state change events: lifecycle callback methods that are defined on an entity class and are invoked whenever the entity state changes, and entity listeners, which are more general because the entity listener can be registered on several entities.
- Entity listener examples
  You can write EntityListeners based on your requirements. Several example scripts follow.

Related concepts:
Relationship management
Entity manager in a distributed environment
EntityManager fetch plan support
Entity query queues
Tuning EntityManager interface performance
Caching objects and their relationships (EntityManager API)
Entity manager in a distributed environment
EntityManager fetch plan support
Entity query queues
Routing cache objects to the same partition
Related tasks:
Tutorial: Storing order information in entities
Collocating multiple cache objects in the same partition
Related reference:
Defining an entity schema
EntityTransaction interface
Entity performance instrumentation agent
Defining an entity schema
Entity listeners and callback methods
Entity listener examples
EntityTransaction interface
Related information:
Sample: Running Queries in Parallel using a ReduceGridAgent

Entity listeners and callback methods

Applications can be notified when the state of an entity transitions from state to state. Two callback mechanisms exist for state change events: lifecycle callback methods that are defined on an entity class and are invoked whenever the entity state changes, and entity listeners, which are more general because the entity listener can be registered on several entities.

Lifecycle of an entity instance

An entity instance has the following states:

- **New**: A newly created entity instance that does not exist in the eXtreme Scale cache.
- **Managed**: The entity instance exists in the eXtreme Scale cache and is retrieved or persisted using the entity manager. An entity must be associated with an active transaction to be in the managed state.
- **Detached**: The entity instance exists in the eXtreme Scale cache, but is no longer associated with an active transaction.
- **Removed**: The entity instance is removed, or is scheduled to be removed, from the eXtreme Scale cache when the transaction is flushed or committed.
- **Invalidated**: The entity instance is invalidated, or is scheduled to be invalidated, from the eXtreme Scale cache when the transaction is flushed or committed.

When entities change from state to state, you can invoke life-cycle, call-back methods.

The following sections further describe the meanings of New, Managed, Detached, Removed and Invalidated states as the states apply to an entity.

Entity lifecycle callback methods

Entity lifecycle callback methods can be defined on the entity class and are invoked when the entity state changes. These methods are useful for validating entity fields and updating transient state that is not usually persisted with the entity. Entity lifecycle callback methods can also be defined on classes that are not using entities. Such classes are entity listener classes, which can be associated with multiple entity types. Lifecycle callback methods can be defined using both metadata annotations and a entity metadata XML descriptor file:

- **Annotations**: lifecycle callback methods can be denoted using the PrePersist, PostPersist, PreRemove, PostRemove, PreUpdate, PostUpdate, and PostLoad annotations in an entity class.
- **Entity XML descriptor** : lifecycle callback methods can be described using XML when annotations are not available.

Entity listeners

An entity listener class is a class that does not use entities that defines one or more entity lifecycle callback methods. Entity listeners are useful for general purpose auditing or logging applications. Entity listeners can be defined using both metadata annotations and a entity metadata XML descriptor file:

- **Annotation**: The EntityListeners annotation can be used to denote one or more entity listener classes on an entity class. If multiple entity listeners are defined, the order in which they are invoked is determined by the order in which they are specified in the EntityListeners annotation.
- **Entity XML descriptor**: The XML descriptor can be used as an alternative to specify the invocation order of entity listeners or to override the order that is specified in metadata annotations.

Callback method requirements

Any subset or combination of annotations can be specified on an entity class or a listener class. A single class cannot have more than one lifecycle callback method for the same lifecycle event. However, the same method can be used for multiple callback events. The entity listener class must have a public no-arg constructor. Entity listeners are stateless. The lifecycle of an entity listener is unspecified. eXtreme Scale does not support entity inheritance, so callback methods can only be defined in the entity class, but not in the superclass.

Callback method signature

Entity lifecycle callback methods can be defined on an entity listener class, directly on an entity class, or both. Entity lifecycle callback methods can be defined using both metadata annotations and the entity XML descriptor. The annotations used for callback methods on the entity class and on the entity listener class are the same. The signatures of the callback methods are different when defined on an entity class versus an entity listener class. Callback methods defined on an entity class or mapped superclass have the following signature:

```
void <METHOD>()
```

Callback methods that are defined on an entity listener class have the following signature:

```
void <METHOD>(Object)
```

The Object argument is the entity instance for which the callback method is invoked. The Object argument can be declared as a java.lang.Object object or the actual entity type.
Callback methods can have public, private, protected, or package level access, but must not be static or final.

The following annotations are defined to designate lifecycle event callback methods of the corresponding types:

- com.ibm.websphere.projector.annotations.PrePersist
- com.ibm.websphere.projector.annotations.PostPersist
- com.ibm.websphere.projector.annotations.PreRemove
- com.ibm.websphere.projector.annotations.PostRemove
- com.ibm.websphere.projector.annotations.PreUpdate
- com.ibm.websphere.projector.annotations.PostUpdate
- com.ibm.websphere.projector.annotations.PostLoad

See the API Documentation for more details. Each annotation has an equivalent XML attribute defined in the entity metadata XML descriptor file.

Lifecycle callback method semantics

Each of the different lifecycle callback methods has a different purpose and is called in different phases of the entity lifecycle:

PrePersist
: Invoked for an entity before the entity has been persisted to the store, which includes entities that have been persisted due to a cascading operation. This method is invoked on the thread of the EntityManager.persist operation.

PostPersist
: Invoked for an entity after the entity has been persisted to the store, which includes entities that have been persisted due to a cascading operation. This method is invoked on the thread of the EntityManager.persist operation. It is called after the EntityManager.flush or EntityManager.commit is called.

PreRemove
: Invoked for an entity before the entity has been removed, which includes entities that have been removed due to a cascading operation. This method is invoked on the thread of the EntityManager.remove operation.

PostRemove

Invoked for an entity after the entity has been removed, which includes entities that have been removed due to a cascading operation. This method is invoked on the thread of the EntityManager.remove operation. It is called after the EntityManager.flush or EntityManager.commit is called.

PreUpdate
: Invoked for an entity before the entity has been updated to the store. This method is invoked on the thread of the transaction flush or commit operation.

PostUpdate
: Invoked for an entity after the entity has been updated to the store. This method is invoked on the thread of the transaction flush or commit operation.

PostLoad
: Invoked for an entity after the entity has been loaded from the store which includes any entities that are loaded through an association. This method is invoked on the thread of the loading operation, such as EntityManager.find or a query.

Duplicate lifecycle callback methods

If multiple callback methods are defined for an entity lifecycle event, the ordering of the invocation of these methods is as follows:

1. **Lifecycle callback methods defined in the entity listeners**: The lifecycle callback methods that are defined on the entity listener classes for an entity class are invoked in the same order as the specification of the entity listener classes in the EntityListeners annotation or the XML descriptor.
2. **Listener super class**: Callback methods defined in the super class of the entity listener are invoked before the children.
3. **Entity lifecycle methods**: WebSphere® eXtreme Scale does not support entity inheritance, so the entity lifecycle methods can only be defined in the entity class.

Exceptions

Lifecycle callback methods might result in run time exceptions. If a lifecycle callback method results in a run time exception within a transaction, the transaction is rolled back. No further lifecycle callback methods are invoked after a runtime exception results.

Related concepts:
Tuning EntityManager interface performance
Caching objects and their relationships (EntityManager API)
Entity manager in a distributed environment
Interacting with EntityManager
EntityManager fetch plan support
Entity query queues
Routing cache objects to the same partition

Related tasks:
Tutorial: Storing order information in entities
Collocating multiple cache objects in the same partition

Related information:
Sample: Running Queries in Parallel using a ReduceGridAgent

Entity listener examples

You can write EntityListeners based on your requirements. Several example scripts follow.

EntityListeners example using annotations

The following example shows the life-cycle callback method invocations and order of the invocations. Assume an entity class Employee and two entity listeners exist: EmployeeListener and EmployeeListener2.

```
@Entity
@EntityListeners({EmployeeListener.class, EmployeeListener2.class})
public class Employee {
    @PrePersist
    public void checkEmployeeID() {
        ....
    }
}

public class EmployeeListener {
    @PrePersist
    public void onEmployeePrePersist(Employee e) {
        ....
    }
}

public class PersonListener {
    @PrePersist
    public void onPersonPrePersist(Object person) {
        ....
    }
}

public class EmployeeListener2 extends PersonListener {
    @PrePersist
    public void onEmployeePrePersist2(Object employee) {
        ....
    }
}
```

If a PrePersist event occurs on an Employee instance, the following methods are called in order:

1. onEmployeePrePersist method
2. onPersonPrePersist method
3. onEmployeePrePersist2 method
4. checkEmployeeID method

Entity listeners example using XML

The following example shows how to set an entity listener on an entity using the entity descriptor XML file:

```
<entity
    class-name="com.ibm.websphere.objectgrid.sample.Employee"
    name="Employee" access="FIELD">
    <attributes>
        <id name="id" />
        <basic name="value" />
    </attributes>
    <entity-listeners>
        <entity-listener
            class-name="com.ibm.websphere.objectgrid.sample.EmployeeListener">
            <pre-persist method-name="onListenerPrePersist" />
            <post-persist method-name="onListenerPostPersist" />
        </entity-listener>
    </entity-listeners>
    <pre-persist method-name="checkEmployeeID" />
</entity>
```

The entity Employee is configured with a com.ibm.websphere.objectgrid.sample.EmployeeListener entity listener class , which has two life-cycle callback methods defined. The onListenerPrePersist method is for the PrePersist event, and the onListenerPostPersist method is for the PostPersist event. Also, the checkEmployeeID method in the Employee class is configured to listen for the PrePersist event.

Related concepts:
Tuning EntityManager interface performance
Caching objects and their relationships (EntityManager API)
Entity manager in a distributed environment
Interacting with EntityManager
EntityManager fetch plan support
Entity query queues
Routing cache objects to the same partition

Related tasks:
Tutorial: Storing order information in entities
Collocating multiple cache objects in the same partition

Related information:
Sample: Running Queries in Parallel using a ReduceGridAgent

EntityManager fetch plan support

A FetchPlan is the strategy that the entity manager uses for retrieving associated objects if the application needs to access relationships.

Example

Assume for example that your application has two entities: Department and Employee. The relationship between the Department entity and the Employee entity is a bi-directional one-to-many relationship: One department has many employees, and one employee belongs to only one department. Since most of the time, when Department entity is fetched, its employees are likely to be fetched, the fetch type of this one-to-many relationship is set to be EAGER.

Here is a snippet of the Department class.

```
        @Entity
public class Department {

    @Id
    private String deptId;

    @Basic
    String deptName;

                @OneToMany(fetch = FetchType.EAGER, mappedBy="department", cascade = {CascadeType.PERSIST})
                public Collection<Employee> employees;

}
```

In a distributed environment, when an application calls `em.find(Department.class, "dept1")` to find a Department entity with key "dept1", this find operation will get the Department entity and all its eager-fetched relations. In the case of the preceding snippet, these are all the employees of department "dept1".

Prior to WebSphere® eXtreme Scale 6.1.0.5, the retrieval of one Department entity and N Employee entities incurred N+1 client-server trips because the client retrieved one entity for one client-server trip. You can improve performance if you retrieve these N+1 entities in one trip.

Fetch plan

A fetch plan can be used to customize how to fetch eager relationships by customizing the maximum depth of the relationships. The fetch depth overrides eager relations greater than the specified depth to lazy relations. By default, the fetch depth is the maximum fetch depth. This means that eager relationships of all levels that are eager-navigable from the root entity will be fetched. An EAGER relationship is eager-navigable from a root entity if and only if all the relations starting from the root entity to it are configured as eager-fetched.

In the previous example, the Employee entity is eager-navigable from the Department entity because the Department-Employee relationship is configured as eager-fetched.

If the Employee entity has another eager relationship to an Address entity for instance, then the Address entity is also eager-navigable from the Department entity. However, if the Department-Employee relationships were configured as lazy-fetch, then the Address entity is not eager-navigable from the Department entity because the Department-Employee relationship breaks the eager fetch chain.

A FetchPlan object can be retrieved from the EntityManager instance. The application can use the setMaxFetchDepth method to change the maximum fetch depth.

A fetch plan is associated with an EntityManager instance. The fetch plan applies to any fetch operation, more specifically as follows.

- EntityManager `find(Class class, Object key)` and `findForUpdate(Class class, Object key)` operations
- Query operations
- QueryQueue operations

The FetchPlan object is mutable. Once changed, the changed value will be applied to the fetch operations executed afterward.

A fetch plan is important for a distributed deployment because it decides whether the eager-fetched relationship entities are retrieved with the root entity in one client-server trip or more than one.

Continuing with the previous example, consider further that the fetch plan has maximum depth set to infinity. In that case, when an application calls `em.find(Department.class, "dept1")` to find a Department, this find operation will get one Department entity and N employee entities in one client-server trip. However, for a fetch plan with maximum fetch depth set to zero, only the Department object will be retrieved from the server, while the Employee entities are retrieved from the server only when the employees collection of the Department object is accessed.

Different fetch plans

You have several different fetch plans based on your requirements, explained in the following sections.

Impact on a distributed grid

- *Infinite-depth fetch plan:* An infinite-depth fetch plan has its maximum fetch depth set to `FetchPlan.DEPTH_INFINITE`.
  In a client-server environment, if an infinite-depth fetch plan is used, then all the relations that are eager-navigable from the root entity will be retrieved in one client-server trip.

  Example: If the application is interested in all the Address entities of all employees of a particular Department, then it uses an infinite-depth fetch plan to retrieve all the associated Address entities. The following code only incurs one client-server trip.

  ```
  em.getFetchPlan().setMaxFetchDepth(FetchPlan.DEPTH_INFINITE);

  tran.begin();
  Department dept = (Department) em.find(Department.class, "dept1");
  // do something with Address object.
  for (Employee e: dept.employees) {
      for (Address addr: e.addresses) {
      // do something with addresses.
      }
  }
  tran.commit();
  ```

- *Zero-depth fetch plan:* A zero-depth fetch plan has its maximum fetch depth set to 0.
  In a client-server environment, if a zero fetch plan is used, then only the root entity will be retrieved in the first client-server trip. All the eager relationships are treated as if they were lazy.

  Example: In this example, the application is only interested in the Department entity attribute. It does not need to access its employees, so the application sets the fetch plan depth to 0.

  ```
  Session session = objectGrid.getSession();
  EntityManager em = session.getEntityManager();
  EntityTransaction tran = em.getTransaction();
  em.getFetchPlan().setMaxFetchDepth(0);

  tran.begin();
  Department dept = (Department) em.find(Department.class, "dept1");
  // do something with dept object.
  tran.commit();
  ```

- *Fetch plan with depth k* :
  A *k*-depth fetch plan has its maximum fetch depth set to *k*.

  In a client-server eXtreme Scale environment, if a *k*-depth fetch plan is used, then all the relationships eager-navigable from the root entity within *k* steps will be retrieved in the first client-server trip.

  The infinite-depth fetch plan (*k* = infinity) and zero-depth fetch plan (*k* = 0) are just two examples of the *k*-depth fetch plan.

To continue expanding on the previous example, assume there is another eager relationship from the entity Employee to the entity Address. If the fetch plan has maximum fetch depth set to 1, then the `em.find(Department.class, "dept1")` operation will retrieve the Department entity and all its Employee entities in one client-server trip. However, the Address entities will not be retrieved because they are not eager-navigable to the Department entity within 1 step, but 2 steps.

If you use a fetch plan with depth set to 2, then the `em.find(Department.class, "dept1")` operation will retrieve the Department entity, all its Employee entities, and all Address entities associated with the Employee entities in one client-server trip.

Tip: The default fetch plan has maximum fetch depth set to infinity, so the default behavior of a fetch operation can change. All the eager-navigable relationships from the root entity are retrieved. Instead of multiple trips, now the fetch operation only incurs one client-server trip with the default fetch plan. To keep the settings for the product from the prior version, set the fetch depth to 0.

- *Fetch plan used on query:*
  If you execute an entity query you can also use a fetch plan to customize relationship retrieval.

  For example, the query `SELECT d FROM Department d WHERE "d.deptName='Department'"` result has a relationship to the Department entity. Notice the fetch plan depth starts with the query result association: In this case, the Department entity, not the query result itself. That is, the Department entity is on fetch-depth level 0. Therefore a fetch plan with maximum fetch depth 1 will retrieve the Department entity and its Employee entities in one client-server trip.

  Example: In this example, the fetch plan depth is set to 1, so the Department entity and its Employee entities are retrieved in one client-server trip, but the Address entities will not be retrieved in the same trip.

Important: If a relationship is ordered, using either OrderBy annotation or configuration, then it is considered an eager relationship even if it is configured as lazy-fetch.

Performance considerations in a distributed environment

By default, all relationships that are eager-navigable from the root entity will be retrieved in one client-server trip. This can improve performance if all the relationships are going to be used. However, in certain usage scenarios, not all relationships eager-navigable from the root entity are used, so they incur both run-time overhead and bandwidth overhead by retrieving those unused entities.

For such cases, the application can set the maximum fetch depth to a small number to decrease the depth of entities to be retrieved by making all the eager relations after that certain depth lazy. This setting can improve performance.

Proceeding still further with the previous Department-Employee-Address example, by default, all the Address entities associated with employees of the Department "dept1" will be retrieved when `em.find(Department.class, "dept1")` is called. If the application does not use Address entities, it can set the maximum fetch depth to 1, so the Address entities will not be retrieved with the Department entity.

Related concepts:
Relationship management
Entity manager in a distributed environment
Interacting with EntityManager
Entity query queues
Tuning EntityManager interface performance
Caching objects and their relationships (EntityManager API)
Entity manager in a distributed environment
Interacting with EntityManager
Entity query queues
Routing cache objects to the same partition
Related tasks:
Tutorial: Storing order information in entities
Collocating multiple cache objects in the same partition
Related reference:
Defining an entity schema
EntityTransaction interface
Entity performance instrumentation agent
Defining an entity schema
Entity listeners and callback methods
Entity listener examples
EntityTransaction interface
Related information:
Sample: Running Queries in Parallel using a ReduceGridAgent

Entity query queues

Query queues allow applications to create a queue qualified by a query in the server-side or local eXtreme Scale over an entity. Entities from the query result are stored in this queue. Currently, query queue is only supported in a map that is using the pessimistic lock strategy.

A query queue is shared by multiple transactions and clients. After the query queue becomes empty, the entity query associated with this queue is rerun and new results are added to the queue. A query queue is uniquely identified by the entity query string and parameters. There is only one instance for each unique query queue in one ObjectGrid instance. See the EntityManager API documentation for additional information.

Query queue example

The following example shows how query queue can be used.

```
/**
 * Get a unassigned question type task
 */
private void getUnassignedQuestionTask() throws Exception {
    EntityManager em = og.getSession().getEntityManager();
    EntityTransaction tran = em.getTransaction();

    QueryQueue queue = em.createQueryQueue("SELECT t FROM Task t
                                WHERE t.type=?1 AND t.status=?2", Task.class);
    queue.setParameter(1, new Integer(Task.TYPE_QUESTION));
    queue.setParameter(2, new Integer(Task.STATUS_UNASSIGNED));

    tran.begin();
    Task nextTask = (Task) queue.getNextEntity(10000);
    System.out.println("next task is " + nextTask);
    if (nextTask != null) {
        assignTask(em, nextTask);
    }
    tran.commit();
}
```

The previous example first creates a QueryQueue with a entity query string, "SELECT t FROM Task t WHERE t.type=?1 AND t.status=?2". Then it sets the parameters for the QueryQueue object. This query queue represents all "unassigned" tasks of the type "question". The QueryQueue object is very similar to an entity Query object.

After the QueryQueue is created, an entity transaction is started and the getNextEntity method is invoked, which retrieves the next available entity with a timeout value set to 10 seconds. After the entity is retrieved, it is processed in the assignTask method. The assignTask modifies the Task entity instance and changes the status to "assigned" which effectively removes it from the queue since it no longer matches the QueryQueue's filter. Once assigned, the transaction is committed.

From this simple example, you can see a query queue is similar to an entity query. However, they differ in the following ways:

1. Entities in the query queue can be retrieved in an iterative manner. The user application decides the number of entities to be retrieved. For example, if QueryQueue.getNextEntity(timeout) is used, only one entity is retrieved, and if QueryQueue.getNextEntities(5, timeout) is used, 5 entities are retrieved. In a distributed environment, the number of entities directly decides the number of bytes to be transferred from the server to client.
2. When an entity is retrieved from the query queue, a U lock is placed on the entity so no other transactions can access it.

Retrieve entities in a loop

You can retrieve entities in a loop. An example that illustrates how to get all the unassigned, question type tasks completed follows.

```
/**
 * Get all unassigned question type tasks
 */
private void getAllUnassignedQuestionTask() throws Exception {
    EntityManager em = og.getSession().getEntityManager();
    EntityTransaction tran = em.getTransaction();

    QueryQueue queue = em.createQueryQueue("SELECT t FROM Task t WHERE
                        t.type=?1 AND t.status=?2", Task.class);
    queue.setParameter(1, new Integer(Task.TYPE_QUESTION));
    queue.setParameter(2, new Integer(Task.STATUS_UNASSIGNED));

    Task nextTask = null;

    do {
        tran.begin();
        nextTask = (Task) queue.getNextEntity(10000);
        if (nextTask != null) {
            System.out.println("next task is " + nextTask);
        }
        tran.commit();
    } while (nextTask != null);
}
```

If there are 10 unassigned question-type tasks in the entity map, you might expect that you will have 10 entities printed to the console. However, if this example is run, you will see the program never exits, which might be contrary to what you assumed.

When a query queue is created and the getNextEntity is called, the entity query associated with the queue is executed and the 10 results are populated into the queue. When getNextEntity is called, an entity is taken off the queue. After 10 getNextEntity calls are executed, the queue is empty. The entity query will automatically re-run. Since these 10 entities still exist and match the query queue's filter criteria, they are populated into the queue again.

If the following line is added after the println() statement, you will see only 10 entities printed.

```
em.remove(nextTask);
```

For information on using SessionHandle with QueryQueue in a per-container placement deployment, read about SessionHandle integration.

Query queues deployed to all partitions

In a distributed eXtreme Scale, a query queue can be created for one partition or all partitions. If a query queue is created for all partitions, there will be one query queue instance in each partition.

When a client tries to get the next entity using the QueryQueue.getNextEntity or QueryQueue.getNextEntities method, the client sends a request to one of the partitions. A client sends peek and pin requests to the server:

- With a peek request, the client sends a request to one partition and the server returns immediately. If there is an entity in the queue, the server sends a response with the entity; if there is not, the server sends a response with no entity. In either case, the server will return immediately.
- With a pin request, the client sends a request to one partition and the server waits until an entity is available. If there is an entity in the queue, the server sends a response with the entity immediately; if there is not, the server waits on the queue until either an entity is available or the request times out.

An example of how an entity is retrieved for a query queue which is deployed to all partitions (n) follows:

1. When a QueryQueue.getNextEntity or QueryQueue.getNextEntities method is called, the client picks a random partition number from 0 to n-1.
2. The client sends peek request to the random partition.
    - If an entity is available, the QueryQueue.getNextEntity or QueryQueue.getNextEntities method exits by returning the entity.
    - If an entity is not available and is not the last unvisited partition, the client sends a peek request to the next partition.
    - If an entity is not available and it is the last unvisited partition, the client instead sends a pin request.
    - If the pin request to the last partition times-out and there is still no data available, the client will make a last effort by sending peek request to all partitions serially one more round. Therefore, if any entity is available in the previous partitions, the client will be able to get it.

Subset entity and no-entity support

The method to create a QueryQueue object in the entity manager follows:

```
public QueryQueue createQueryQueue(String qlString, Class entityClass);
```

The result in the query queue should be projected to the object defined by the second parameter to the method, Class entityClass.

If this parameter is specified, the class must have the same entity name as specified in the query string. This is useful if you want to project an entity into a subset entity. If a null value is used as the entity class, then the result will not be projected. The value stored in the map will be in a entity tuple format.

Client-side key collision

In distributed eXtreme Scale environment, query queue is only supported for eXtreme Scale maps with pessimistic locking mode. Therefore, there is no near cache on the client side. However, a client could have data (key and value) in the transactional map. This potentially could lead to a key collision when an entity retrieved from the server share the same key as an entry already in the transactional map.

When a key collision happens, the eXtreme Scale client run time uses the following rule to either throw an exception or silently override the data.

1. If the collided key is the key of the entity specified in the entity query associated with the query queue, then an exception is thrown. In this case, the transaction is rolled back, and the U lock on this entity key will be released on the server side.
2. Otherwise, if the collided key is the key of the entity association, the data in the transactional map will be overridden without warning.

The key collision only happens when there is a data in the transactional map. In other words, it only happens when a getNextEntity or getNextEntities call is called in a transaction which has already been dirtied (a new data has been inserted or a data has been updated). If an application does not want a key collision happen, it should always call getNextEntity or getNextEntities in a transaction which has not been dirtied.

Client failures

After a client sends a getNextEntity or getNextEntities request to the server, the client could fail as follows:

1. The client sends a request to the server and then goes down.
2. The client gets one or more entities from the server and then goes down.

In the first case, the server discovers that the client is going down when it tries to send back the response to the client. In the second case, when the client gets one or more entities from the server, an X lock is placed on these entities. If the client goes down, the transaction will eventually time out, and the X lock will be released.

Query with ORDER BY clause
Generally, query queues do not honor the ORDER BY clause. If you call getNextEntity or getNextEntities from the query queue, there is no guarantee the entities are returned according to the order. The reason is that the entities cannot be ordered across partitions. In the case that the query queue is deployed to all partitions, when a getNextEntity or getNextEntities call is executed, a random partition is picked to process the request. Therefore, the order is not guaranteed.

ORDER BY is honored if a query queue is deployed to a single partition.

For more information see EntityManager Query API.

One call per transaction

Each QueryQueue.getNextEntity or QueryQueue.getNextEntities call retrieves matched entities from one random partition. Applications should call exactly one QueryQueue.getNextEntity or QueryQueue.getNextEntities on one transaction. Otherwise eXtreme Scale could end up touching entities from multiple partitions, causing an exception to be thrown at the commit time.

Related concepts:
Relationship management
Entity manager in a distributed environment
Interacting with EntityManager
EntityManager fetch plan support
Tuning EntityManager interface performance

Caching objects and their relationships (EntityManager API)
Entity manager in a distributed environment
Interacting with EntityManager
EntityManager fetch plan support
Routing cache objects to the same partition
Related tasks:
Tutorial: Storing order information in entities
Collocating multiple cache objects in the same partition
Related reference:
Defining an entity schema
EntityTransaction interface
Entity performance instrumentation agent
Defining an entity schema
Entity listeners and callback methods
Entity listener examples
EntityTransaction interface
Related information:
Sample: Running Queries in Parallel using a ReduceGridAgent

EntityTransaction interface

You can use the EntityTransaction interface to demarcate transactions.

Purpose

To demarcate a transaction, you can use the EntityTransaction interface, which is associated with an entity manager instance. Use the EntityManager.getTransaction method to retrieve the EntityTransaction instance for the entity manager. Each EntityManager and EntityTransaction instance are associated with the Session. You can demarcate transactions with either the EntityTransaction or Session. Methods on the EntityTransaction interface do not have any checked exceptions. Only runtime exceptions of type PersistenceException or its subclasses result.

For more information about the EntityTransaction interface, see the API documentation.

Related concepts:
Relationship management
Entity manager in a distributed environment
Interacting with EntityManager
EntityManager fetch plan support
Entity query queues
Tuning EntityManager interface performance
Caching objects and their relationships (EntityManager API)
Entity manager in a distributed environment
Interacting with EntityManager
EntityManager fetch plan support
Entity query queues
Routing cache objects to the same partition
Related tasks:
Tutorial: Storing order information in entities
Collocating multiple cache objects in the same partition
Related reference:
Defining an entity schema
Related information:
Sample: Running Queries in Parallel using a ReduceGridAgent

Retrieving entities and objects (Query API)

WebSphere® eXtreme Scale provides a flexible query engine for retrieving entities using the EntityManager API and Java™ objects using the ObjectQuery API.

WebSphere eXtreme Scale query capabilities

With the eXtreme Scale query engine, you can perform SELECT type queries over an entity or object-based schema using the eXtreme Scale query language.

This query language provides the following capabilities:

- Single and multi-valued results
- Aggregate functions
- Sorting and grouping
- Joins
- Conditional expressions with subqueries
- Named and positional parameters
- eXtreme Scale index use

- Path expression syntax for object navigation
- Pagination

Query interface

Use the query interface to control entity query execution.

Use the EntityManager.createQuery(String) method to create a Query. You can use each query instance multiple times with the EntityManager instance in which it was retrieved.

Each query result produces an entity, where the entity key is the row ID (of type long) and the entity value contains the field results of the SELECT clause. You can use each query result in subsequent queries.

The following methods are available on the com.ibm.websphere.objectgrid.em.Query interface.

public ObjectMap getResultMap()
The getResultMap method runs a SELECT query and returns the results in an ObjectMap object with the results in query-specified order. The resulting ObjectMap is valid only for the current transaction.

The map key is the result number, of type long, starting at 1. The map value is of type com.ibm.websphere.projector.Tuple where each attribute and association is named based on its ordinal position within the select clause of the query. Use the method to retrieve the EntityMetadata for the Tuple object that is stored within the map.

The getResultMap method is the fastest method for retrieving query result data where multiple results can exist. You can retrieve the name of the resulting entity using the ObjectMap.getEntityMetadata() and EntityMetadata.getName() methods.

Example: The following query returns two rows.

```
String ql = SELECT e.name, e.id, d from Employee e join e.dept d WHERE d.number=5
 Query q = em.createQuery(ql);
 ObjectMap resultMap = q.getResultMap();
 long rowID = 1; // starts with index 1
 Tuple tResult = (Tuple) resultMap.get(new Long(rowID));
 while(tResult != null) {
     // The first attribute is name and has an attribute name of 1
     // But has an ordinal position of 0.
     String name = (String)tResult.getAttribute(0);
     Integer id = (String)tResult.getAttribute(1);

     // Dept is an association with a name of 3, but
     // an ordinal position of 0 since it's the first association.
     // The association is always a OneToOne relationship,
     // so there is only one key.
     Tuple deptKey = tResult.getAssociation(0,0);
     ...
     ++rowID;
     tResult = (Tuple) resultMap.get(new Long(rowID));

 }
```

public Iterator getResultIterator

The getResultIterator method runs a SELECT query and returns the query results using an Iterator where each result is either an Object for a single-valued query, or an Object array for a multiple-valued query. The values in the Object[] result are stored in query order. The result Iterator is valid for the current transaction only.

This method is preferred for retrieving query results within the EntityManager context. You can use the optional setResultEntityName(String) method to name the resulting entity so that it can be used in further queries.

Example: The following query returns two rows.

```
String ql = SELECT e.name, e.id, e.dept from Employee e WHERE e.dept.number=5
 Query q = em.createQuery(ql);
 Iterator results = q.getResultIterator();
 while(results.hasNext()) {
     Object[] curEmp = (Object[]) results.next();
     String name = (String) curEmp[0];
     Integer id = (Integer) curEmp[1];
     Dept d = (Dept) curEmp[2];
     ...
 }
```

public Iterator getResultIterator(Class resultType)

The getResultIterator(Class resultType) method runs a SELECT query and returns the query results using an entity Iterator. The entity type is determined by the resultType parameter. The result Iterator is valid only for the current transaction.

Use this method when you want to use the EntityManager APIs to access the resulting entities.

Example: The following query returns all of the employees and the department to which they belong for one division, ordering by salary. To print out the five employees with the highest salaries and then select work with employees from only one department in the same working set, use the following code.

```
String string_ql = "SELECT e.name, e.id, e.dept from Employee e WHERE
                e.dept.division='Manufacturing' ORDER BY e.salary DESC";
Query query1 = em.createQuery(string_ql);
```

```
query1.setResultEntityName("AllEmployees");
Iterator results1 = query1.getResultIterator(EmployeeResult.class);
int curEmployee = 0;
System.out.println("Highest paid employees");
while (results1.hasNext() && curEmployee++ < 5) {
        EmployeeResult curEmp = (EmployeeResult) results1.next();
        System.out.println(curEmp);
        // Remove the employee from the resultset.
        em.remove(curEmp);
}

// Flush the changes to the result map.
em.flush();

// Run a query against the local working set without the employees we
// removed
String string_q2 = "SELECT e.name, e.id, e.dept from AllEmployees e
                WHERE e.dept.name='Hardware'";
Query query2 = em.createQuery(string_q2);
Iterator results2 = query2.getResultIterator(EmployeeResult.class);
System.out.println("Subset list of Employees");
while (results2.hasNext()) {
        EmployeeResult curEmp = (EmployeeResult) results2.next();
        System.out.println(curEmp);
}
```

public Object getSingleResult

The getSingleResult method runs a SELECT query that returns a single result.

If the SELECT clause has more than one field defined, then the result is an object array, where each element in the array is based on its ordinal position within the SELECT clause of the query.

```
String ql = SELECT e from Employee e WHERE e.id=100"
 Employee e = em.createQuery(ql).getSingleResult();

 String ql = SELECT e.name, e.dept from Employee e WHERE e.id=100"
 Object[] empData = em.createQuery(ql).getSingleResult();
 String empName= (String) empData[0];
 Department empDept = (Department) empData[1];
```

public Query setResultEntityName(String entityName)

The setResultEntityName(String entityName) method specifies the name of the query result entity.

Each time the getResultIterator or getResultMap methods are invoked, an entity with an ObjectMap is dynamically created to hold the results of the query. If the entity is not specified, or null, the entity and ObjectMap name are automatically generated.

Because all query results are available for the duration of a transaction, a query name cannot be reused in a single transaction.

public Query setPartition(int partitionId)

Set the partition to where the query routes.

This method is required if the maps in the query are partitioned and if the entity manager does not have affinity to a single schema root entity partition.

Use the PartitionManager Interface to determine the number of partitions for the backing map of a given entity.

The following table provides descriptions of the other methods that are available through the query interface.

Table 1. Other methods.

Method	Result
public Query setMaxResults(int maxResult)	Set the maximum number of results to retrieve.
public Query setFirstResult(int startPosition)	Set the position of the first result to retrieve.
public Query setParameter(String name, Object value)	Bind an argument to a named parameter.
public Query setParameter(int position, Object value)	Bind an argument to a positional parameter.
public Query setFlushMode(FlushModeType flushMode)	Set the flush mode type to be used when the query runs, overriding the flush mode type set on the EntityManager.

eXtreme Scale query elements

With the eXtreme Scale query engine, you can use a single query language for searching the eXtreme Scale cache. This query language can query Java objects that are stored in ObjectMap objects and Entity objects. Use the following syntax for creating a query string.

An eXtreme Scale query is a string that contains the following elements:

- A SELECT clause that specifies the objects or values to return.
- A FROM clause that names the object collections.
- An optional WHERE clause that contains search predicates over the collections.

- An optional GROUP BY and HAVING clause (see eXtreme Scale query aggregation functions).
- An optional ORDER BY clause that specifies the ordering of the result collection.

Collections of Java objects are identified in queries through the use of their name in the query FROM clause.

The elements of query language are discussed in more detail in the following related topics:

- ObjectGrid query Backus-Naur Form syntax
- Reference for eXtreme Scale queries

The following topics describe the means to use the Query API:

- EntityManager Query API
- Using the ObjectQuery API

- Querying data in multiple time zones
  In a distributed scenario, queries actually run on servers. When querying data with predicates of type calendar, java.util.Date and timestamp, the specified date time value in a query is based on the local time zone of the server.
- Data for different time zones
  When inserting data with calendar, java.util.Date, and timestamp attributes into an ObjectGrid, you must ensure these date time attributes are created based on same time zone, especially when deployed into multiple servers in various time zones. Using the same time zone based date time objects can ensure the application is time-zone safe and data can be queried by calendar, java.util.Date and timestamp predicates.
- Using the ObjectQuery API
  The ObjectQuery API provides methods for querying data in the ObjectGrid that is stored using the ObjectMap API. When a schema is defined in the ObjectGrid instance, the ObjectQuery API can be used to create and run queries over the heterogeneous objects stored in the object maps.
- EntityManager Query API
  The EntityManager API provides methods for querying data in the ObjectGrid that is stored using the EntityManager API. The EntityManager Query API is used to create and run queries over one or more entities defined in eXtreme Scale.
- Reference for eXtreme Scale queries
  WebSphere eXtreme Scale has its own language by which the user can query data.

Querying data in multiple time zones

In a distributed scenario, queries actually run on servers. When querying data with predicates of type calendar, java.util.Date and timestamp, the specified date time value in a query is based on the local time zone of the server.

In a single time-zone system where all clients and servers run on same time zone, you do not need to consider issues related to predicate types with calendar, java.util.Date and timestamp. However, when clients and servers are in different time zones, the specified date time value in queries is based on the server time zone and may return unwanted data back to client. Without knowing the server time zone, the specified date time value is meaningless. So the specified date time value should consider the time zone offset difference between the target time zone and the server time zone.

Time zone offset

For example, assume that a client is in [GMT-0] time zone and the server is in [GMT-6] time zone. The server time zone is 6 hours behind the client. The client would like to run the following query:

```
SELECT e FROM Employee e WHERE e.birthDate='1999-12-31 06:00:00'
```

Assuming the entity Employee has a birthDate attribute that is of type java.util.Date, the client is in [GMT-0] time zone and wants to retrieve Employees with birthDate value as '1999-12-31 06:00:00 [GMT-0]' based on its time zone.

The query will run on the server and the birthDate value used by the query engine will be '1999-12-31 06:00:00 [GMT-6]' that equals to '1999-12-31 12:00:00 [GMT-0]'. Employees with birthDate value equal to '1999-12-31 12:00:00 [GMT-0]' will be returned to the client. Thus, the client will not get wanted Employees with birthDate value '1999-12-31 06:00:00 [GMT-0]'.

The problem described occurs because of the time zone difference between client and server. To solve this problem, one approach is to calculate the time zone offset between client and server and apply the time zone offset on the target date time value in the query. In the previous query example, the time zone offset is -6 hours, and the adjusted birthDate predicate should be “birthDate='1999-12-31 00:00:00'” if the client intends to retrieve Employees with birthDate value '12-31 06:00:00 [GMT-0]'. With the adjusted birthDate value, the server will use '1999-12-31 00:00:00 [GMT-6]' that equals to target value '12-31 06:00:00 [GMT-0]', and the required Employees will be returned to the client.

Distributed deployment in multiple time zones

If the distributed eXtreme Scale grid is deployed into multiple ObjectGrid servers in various time zones, the adjusting time zone offset approach will not work because the client will not know which server will run the query and thus cannot determine the time zone offset to use. The only solution is to use suffix ‘Z' (not case sensitive) on JDBC date and time escape format to indicate using GMT time zone based date time value. The suffix ‘Z' (not case sensitive) indicates to use GMT time zone based date time value. Without the suffix ‘Z', the local time zone based date time value will be used in the process that runs the query.

The following query is equivalent to the previous example, but uses the suffix ‘Z' instead:

```
SELECT e FROM Employee e WHERE e.birthDate='1999-12-31 06:00:00Z'
```

The query should find Employees with birthDate value ‘1999-12-31 06:00:00'. The suffix ‘Z' indicates the specified birthDate value is GMT time zone based, so the GMT time zone based birthDate value ‘1999-12-31 06:00:00 [GMT-0]' will be used by the query engine for matching criteria value. Employees with birthDate attribute value equal to this GMT based birthDate value ‘1999-12-31 06:00:00 [GMT-0]' will be included in query result. Using the suffix ‘Z' on JDBC date time

escape format in any query is crucial to make applications time zone safe. Without this approach, the date time value is server time zone based and is meaningless from the client perspective when clients and servers are in different time zones.

For more information, see Data for different time zones.

Related concepts:
Data for different time zones
Using the ObjectQuery API
EntityManager Query API
Reference for eXtreme Scale queries
Data for different time zones

Data for different time zones

When inserting data with calendar, java.util.Date, and timestamp attributes into an ObjectGrid, you must ensure these date time attributes are created based on same time zone, especially when deployed into multiple servers in various time zones. Using the same time zone based date time objects can ensure the application is time-zone safe and data can be queried by calendar, java.util.Date and timestamp predicates.

Without explicitly specifying a time zone when creating date time objects, Java™ uses the local time zone and may cause inconsistent date time values in clients and servers.

Consider an example in a distributed deployment in which client1 is in time zone [GMT-0] and client2 is in [GMT-6] and both want to create a java.util.Date object with value '1999-12-31 06:00:00'. Then client1 will create java.util.Date object with value '1999-12-31 06:00:00 [GMT-0]' and client2 will create java.util.Date object with value '1999-12-31 06:00:00 [GMT-6]'. Both java.util.Date objects are not equal because the time zone is different. A similar problem occurs when preloading data into partitions residing in servers in different time zones if local time zone is used to create date time objects.

To avoid the described problem, the application can choose a time zone such as [GMT-0] as the base time zone for creating calendar, java.util.Date, and timestamp objects.

Related concepts:
Querying data in multiple time zones
Using the ObjectQuery API
EntityManager Query API
Reference for eXtreme Scale queries
Querying data in multiple time zones

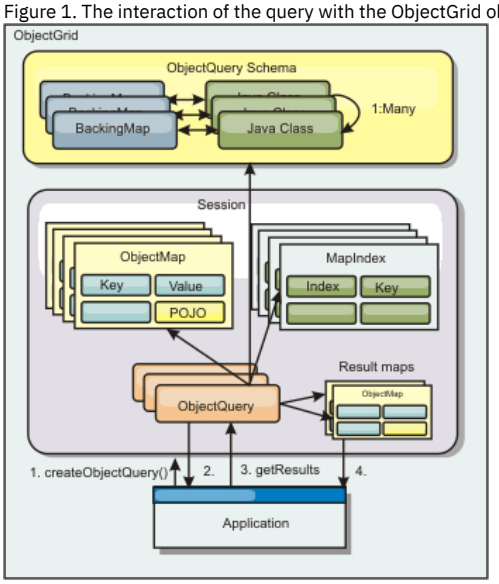
Using the ObjectQuery API

The ObjectQuery API provides methods for querying data in the ObjectGrid that is stored using the ObjectMap API. When a schema is defined in the ObjectGrid instance, the ObjectQuery API can be used to create and run queries over the heterogeneous objects stored in the object maps.

Query and object maps

You can use an enhanced query capability for objects that are stored using the ObjectMap API. These queries allow retrieval of objects using non-key attributes and performs simple aggregations such as sum, avg, min, and max against all the data that matches a query. Applications can construct a query using the Session.createObjectQuery method. This method returns an ObjectQuery object which can then be interrogated to obtain the query results. The query object also allows the query to be customized before running the query. The query is run automatically when any method returning the result is called.

Figure 1. The interaction of the query with the ObjectGrid object maps and how a schema is defined for classes and associated with an ObjectGrid map



Defining an ObjectMap schema

Object maps are used to store objects in various forms and are largely unaware of the format. A schema must be defined in the ObjectGrid that defines the format of the data. A schema is composed of the following pieces:

- The type of object stored in the ObjectMap
- Relationships between ObjectMaps
- The method for which each query should access the data attributes in the objects (fields or property methods)
- The primary key attribute name in the object.

See Configuring an ObjectQuery schema for details.

For an example on creating a schema programmatically or using the ObjectGrid descriptor XML file, see ObjectQuery tutorial - step 3.

Querying objects with the ObjectQuery API

The ObjectQuery interface allows the querying of non-entity objects, which are heterogeneous objects that are stored directly in the ObjectGrid ObjectMaps. The ObjectQuery API provides an easy way to find ObjectMap objects without using the index mechanism directly.

There are two methods for retrieving results from an ObjectQuery: getResultIterator and getResultMap.

Retrieving query results using getResultIterator

Query results are basically a list of attributes. Suppose the query was select a,b,c from X where y=z. This query returns a list of rows containing a, b and c. This list is actually stored in a transaction scoped Map, which means that you must associate an artificial key with each row and use an integer that increases with each row. This map is obtained using the ObjectQuery.getResultMap() method. You can access the elements of each row using code similar to the following:

```
ObjectQuery q = session.createQuery(
    "select c.id, c.firstName, c.surname from Customer c where c.surname=?1");

  q.setParameter(1, "Claus");

  Iterator iter = q.getResultIterator();
  while(iter.hasNext())
  {
    Object[] row = (Object[])iter.next();
    System.out.println("Found a Claus with id "
      + row[objectgrid: 0 ] + ", firstName: "
      + row[objectgrid: 1 ] + ", surname: "
      + row[objectgrid: 2 ]);
  }
```

Retrieving query results using getResultMap

Query results can also be retrieved using the result map directly. The following example shows a query retrieving specific parts of the matching Customers and demonstrates how to access the resulting rows. Notice that if you use the ObjectQuery object to access the data, then the generated long row identifier is hidden. The long row is only visible when using the ObjectMap to access the result.

When the transaction is completed this map disappears. The map is also only visible to the session used, that is, normally to just the thread that created it. The map uses a key of type Long which represents the row ID. The values stored in the map either are of type Object or Object[], where each element matches the type of the element in the select clause of query.

```
ObjectQuery q = em.createQuery(
      "select c.id, c.firstName, c.surname from Customer c where c.surname=?1");
  q.setParameter(1, "Claus");
  ObjectMap qmap = q.getResultMap();
  for(long rowId = 0; true; ++rowId)
  {
    Object[] row = (Object[]) qmap.get(new Long(rowId));
    if(row == null) break;
    System.out.println(" I Found a Claus with id " + row[0]
      + ", firstName: " + row[1]
      + ", surname: " + row[2]);
  }
```

For examples on using the ObjectQuery, see Tutorial: Querying a local in-memory data grid.

- Configuring an ObjectQuery schema
  ObjectQuery relies on schema or shape information to perform semantic checking and to evaluate path expressions. This section describes how to define the schema in XML or programmatically.

Related concepts:
Querying data in multiple time zones
Data for different time zones
EntityManager Query API
Reference for eXtreme Scale queries

Configuring an ObjectQuery schema

ObjectQuery relies on schema or shape information to perform semantic checking and to evaluate path expressions. This section describes how to define the schema in XML or programmatically.

Defining the schema

The ObjectMap schema is defined in the ObjectGrid deployment descriptor XML or programmatically using the normal eXtreme Scale configuration techniques. For an example on how to create a schema, see ObjectQuery tutorial - step 4.

Schema information describes plain old Java™ objects (POJOs): which attributes they consist of and what types of attributes there might be, whether the attributes are primary key fields, single-valued or multi-valued relationships, or bidirectional relationships. Schema information directs ObjectQuery to use field access or property access.

Queryable attributes

When the schema is defined in the ObjectGrid, the objects in the schema are introspected using reflection to determine which attributes are available for querying. You can query the following attribute types:

- Java primitive types including wrappers
- java.lang.String
- java.math.BigInteger
- java.math.BigDecimal
- java.util.Date
- java.sql.Date
- java.sql.Time
- java.sql.Timestamp
- java.util.Calendar
- byte[]
- java.lang.Byte[]
- char[]
- java.lang.Character[]
- J2SE enum

Embedded serializable types other than those stated previously can also be included in a query result, but cannot be included in the WHERE or FROM clause of the query. Serializable attributes are not navigable.

Attribute types can be excluded from the schema if the type is not serializable, the field or property is static, or the field is transient. Since all map objects must be serializable, the ObjectGrid only includes attributes that can be persisted from the object. Other objects are ignored.

Field attributes

When the schema is configured to access the object using fields, all serializable, non-transient fields are automatically incorporated into the schema. To select a field attribute in a query, use the field identifier name as it exists in the class definition.

All public, private, protected and package protected fields are included in the schema.

Property attributes

When the schema is configured to access the object using properties, all serializable methods that follow the JavaBeans property naming conventions will automatically be incorporated into the schema. To select a property attribute for the query, use the JavaBeans style property name conventions.

All public, private, protected and package protected properties are included in the schema.

In the following class, the following attributes are added to the schema: name, birthday, valid.

```
public class Person {
  public String getName(){}
  private java.util.Date getBirthday(){}
  boolean isValid(){}
  public NonSerializableObject getData(){}
}
```

When using a CopyMode of COPY_ON_WRITE, the query schema must always use property-based access. COPY_ON_WRITE creates proxy objects whenever objects are retrieved from the map and can only access those objects using property methods. Failure to do so will result in each query result being set to null.

Relationships

Each relationship must be explicitly defined in the schema configuration. The cardinality of the relationship is automatically determined by the type of the attribute. If the attribute implements the java.util.Collection interface, then the relationship is either a one-to-many or many-to-many relationship.

Unlike entity queries, attributes that refer to other cached objects must not store direct references to the object. References to other objects are serialized as part of the containing object's data. Instead, store the key to the related object.

For example, if there is a many-to-one relationship between a Customer and Order:

```
Incorrect. Storing an object reference.

public class Customer {
  String customerId;
  Collection<Order> orders;
}
```

```
public class Order {
  String orderId;
  Customer customer;
}

Correct. The key to the related object.

public class Customer {
  String customerId;
  Collection<String> orders;
}

public class Order {
  String orderId;
  String customer;
}
```

When you run a query that joins two object maps together, the key is automatically inflated. For example, the following query would return Customer objects:

```
SELECT c FROM Order o JOIN Customer c WHERE orderId=5
```

Using indexes

ObjectGrid uses index plugins to add indexes to maps. The query engine automatically incorporates any indexes that are defined on a schema map element of the type: com.ibm.websphere.objectgrid.plugins.index.HashIndex and the rangeIndex property is set to true. If the index type is not HashIndex and the rangeIndex property is not set to true, then the index is ignored by the query. See ObjectQuery tutorial - step 2 for an example on how to add an index to the schema.

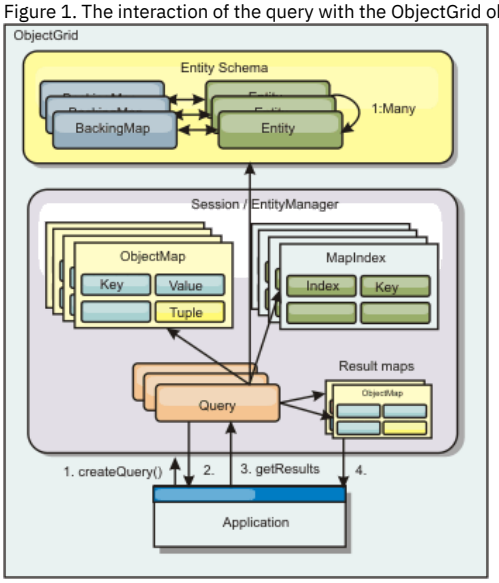
EntityManager Query API

The EntityManager API provides methods for querying data in the ObjectGrid that is stored using the EntityManager API. The EntityManager Query API is used to create and run queries over one or more entities defined in eXtreme Scale.

Query and ObjectMaps for entities

WebSphere® Extended Deployment v6.1 introduced an enhanced query capability for entities stored in eXtreme Scale. These queries allow objects to be retrieved using non-key attributes and to perform simple aggregations such as the sum, average, minimum, and maximum against all the data that matches a query. Applications construct a query using the EntityManager.createQuery API. This returns a Query object and can then be interrogated to obtain the query results. The query object also allows the query to be customized before running the query. The query is run automatically when any method returning the result is called.

Figure 1. The interaction of the query with the ObjectGrid object maps and how the entity schema is defined and associated with an ObjectGrid map.



Retrieving query results using the getResultIterator method

Query results are a list of attributes. If the query was select a,b,c from X where y=z, then a list of rows containing a, b and c is returned. This list is stored in a transaction scoped Map, which means that you must associated an artificial key with each row and use an integer that increases with each row. This map is obtained using the Query.getResultMap method. The map has EntityMetaData, which describes each row in the Map associated with it. You can access the elements of each row using code similar to the following:

```
Query q = em.createQuery("select c.id, c.firstName, c.surname from Customer c where c.surname=?1");

  q.setParameter(1, "Claus");
```

```
  Iterator iter = q.getResultIterator();
  while(iter.hasNext())
  {
    Object[] row = (Object[])iter.next();
    System.out.println("Found a Claus with id "  + row[objectgrid: 0 ]
      + ", firstName: " + row[objectgrid: 1 ]
      + ", surname: " + row[objectgrid: 2 ]);
  }
```

Retrieving query results using getResultMap

The following code shows the retrieval of specific parts of the matching Customers and shows how to access the resulting rows. If you use the Query object to access the data, then the generated long row identifier is hidden. The long is only visible when using the ObjectMap to access the result. When the transaction is completed, then this Map disappears. The Map is only visible to the Session used, that is, normally to just the thread that created it. The Map uses a Tuple for the key with a single attribute, a long with the row ID. The value is another tuple with an attribute for each column in the result set.

The following sample code demonstrates this:

```
Query q = em.createQuery("select c.id, c.firstName, c.surname from
Customer c where c.surname=?1");
q.setParameter(1, "Claus");
ObjectMap qmap = q.getResultMap();
Tuple keyTuple = qmap.getEntityMetadata().getKeyMetadata().createTuple();
for(long i = 0; true; ++i)
{
  keyTuple.setAttribute(0, new Long(i));
  Tuple row = (Tuple)qmap.get(keyTuple);
  if(row == null) break;
  System.out.println(" I Found a Claus with id "  + row.getAttribute(0)
    + ", firstName: " + row.getAttribute(1)
    + ", surname: " + row.getAttribute(2));
}
```

Retrieving query results using an entity result iterator

The following code shows the query and the loop that retrieves each result row using the normal Map APIs. The key for the Map is a Tuple. So, construct one of the correct types using the createTuple method result in keyTuple. Try to retrieve all rows with rowIds from 0 onwards. When you get returns null (indicating key not found), then the loop finishes. Set the first attribute of keyTuple to be the long that you want to find. The value returned by get is also a Tuple with an attribute for each column in the query result. Then, pull each attribute from the value Tuple using getAttribute.

Following is the next code fragment:

```
Query q2 = em.createQuery("select c.id, c.firstName, c.surname from Customer c where c.surname=?1");
q2.setResultEntityName("CustomerQueryResult");
q2.setParameter(1, "Claus");

Iterator iter2 = q2.getResultIterator(CustomerQueryResult.class);
while(iter2.hasNext())
{
  CustomerQueryResult row = (CustomerQueryResult)iter2.next();
  // firstName is the id not the firstName.
  System.out.println("Found a Claus with id " + row.id
    + ", firstName: " + row.firstName
    + ", surname: " + row.surname);
}

em.getTransaction().commit();
```

Specified is a ResultEntityName value on the query. This value tells the query engine that you want to project each row to a specific object, CustomerQueryResult in this case. The class follows:

```
@Entity
public class CustomerQueryResult {
        @Id long rowId;
        String id;
        String firstName;
        String surname;
};
```

In the first snippet, notice that the each query row is returned as a CustomerQueryResult object rather than an Object[]. The result columns of the query are projected to the CustomerQueryResult object. Projecting the result is slightly slower at run time but more readable. Query result Entities should not be registered with eXtreme Scale at startup. If the entities are registered, then a global Map with the same name is created, and the query fails with an error indicating duplicate Map name.

- Simple queries with EntityManager
  WebSphere eXtreme Scale comes with EntityManager query API.

Related concepts:
Querying data in multiple time zones
Data for different time zones
Using the ObjectQuery API
Reference for eXtreme Scale queries

Simple queries with EntityManager

WebSphere® eXtreme Scale comes with EntityManager query API.

The EntityManager query API is very similar to SQL other query engines that query over objects. A query is defined, then the result is retrieved from the query using various getResult methods.

The following examples refer to the entities used in the EntityManager tutorial in the Product Overview.

Running a simple query

In this example, customers with the surname of Claus are queried:

```
em.getTransaction().begin();

  Query q = em.createQuery("select c from Customer c where c.surname='Claus'");

  Iterator iter = q.getResultIterator();
  while(iter.hasNext())
  {
    Customer c = (Customer)iter.next();
    System.out.println("Found a claus with id " + c.id);
  }

  em.getTransaction().commit();
```

Using parameters

Since you want to find all customers with a surname of Claus, a parameter to specify the surname is used since you might may want to use this query more than once.

Positional Parameter Example

```
Query q = em.createQuery("select c from Customer c where c.surname=?1");
  q.setParameter(1, "Claus");
```

Using parameters is very important when the query is used more than once. The EntityManager needs to parse the query string and build a plan for the query, which is expensive. By using a parameter, the EntityManager caches the plan for the query, thereby reducing the time it takes to run a query.

Both positional and named parameters are used:

Named Parameter Example

```
Query q = em.createQuery("select c from Customer c where c.surname=:name");
  q.setParameter("name", "Claus");
```

Using an index to improve performance

If there are millions of customers, then the previous query needs to scan over all rows in the Customer Map. This is not very efficient. But eXtreme Scale provides a mechanism for defining indexes over individual attributes in an entity. The query automatically uses this index when appropriate, which can speed up queries dramatically.

You can specify which attributes to index very simply by using the @Index annotation on the entity attribute:

```
@Entity
public class Customer
{
  @Id String id;
  String firstName;
  @Index String surname;
  String address;
  String phoneNumber;
}
```

The EntityManager creates an appropriate ObjectGrid index for the surname attribute in the Customer entity and the query engine automatically uses the index, which greatly decreases the query time.

Using pagination to improve performance

If there are a million customers named Claus, then it is not likely that you would want to display a page displaying a million customers. It is more likely that you would want to display 10 or 25 customers at a time.

The Query setFirstResult and setMaxResults methods helps by only returning a subset of the results.

Pagination Example

```
Query q = em.createQuery("select c from Customer c where c.surname=:name");
  q.setParameter("name", "Claus");
  // Display the first page
  q.setFirstResult=1;
```

```
q.setMaxResults=25;
displayPage(q.getResultIterator());

// Display the second page
q.setFirstResult=26;
displayPage(q.getResultIterator());
```

Reference for eXtreme Scale queries

WebSphere® eXtreme Scale has its own language by which the user can query data.

ObjectGrid query FROM clause

The FROM clause specifies the collections of objects to which to apply the query. Each collection is identified either by an abstract schema name and an identification variable, called a range variable, or by a collection member declaration that identifies either a single or multi-valued relationship and an identification variable.

Conceptually, the semantics of the query is to first form a temporary collection of tuples, referred to as R. Tuples are composed of elements from the collections that are identified in the FROM clause. Each tuple contains one element from each of the collections in the FROM clause. All possible combinations are formed subject to the constraints that are imposed by the collection member declarations. If any schema name identifies a collection for which there are no records in the persistent store, then the temporary collection R is empty.

Examples using FROM
The DeptBean object contains records 10, 20 and 30. The EmpBean object contains records 1, 2 and 3 that are related to department 10 and records 4 and 5 that are related to department 20. Department 30 has no related employees.

```
FROM DeptBean d, EmpBean e
```

This clause forms a temporary collection R that contains 15 tuples.

```
FROM DeptBean d, DeptBean d1
```

This clause forms a temporary collection R that contains 9 tuples.

```
FROM DeptBean d, IN (d.emps) AS e
```

This clause forms a temporary collection R that contains 5 tuples. Department 30 is not in the R temporary collection because it contains no employees. Department 10 is contained in the R temporary collection three times and department 20 is contained in R twice.

Instead of using IN(d.emps) as e, you can use a JOIN predicate:

```
FROM DeptBean d JOIN d.emps as e
```

After forming the temporary collection, the search conditions of the WHERE clause are applied to the R temporary collection, yielding a new temporary collection R1. The ORDER BY and SELECT clauses are applied to R1 to yield the final result set.

An identification variable is a variable that is declared in the FROM clause using the IN operator or the optional AS operator.

```
FROM DeptBean AS d, IN (d.emps) AS e
```

is equivalent to:

```
FROM DeptBean d, IN (d.emps) e
```

An identification variable that is declared to be an abstract schema name is called a range variable. In the previous query, "d" is a range variable. An identification variable that is declared to be a multi-valued path expression is called a collection member declaration. The "d" and "e" values in the previous example are collection member declarations.

An example of using a single-valued path expression in the FROM clause follows:

```
FROM EmpBean e, IN(e.dept.mgr) as m
```

ObjectGrid query SELECT clause

The syntax of the SELECT clause is illustrated in the following example:

```
SELECT { ALL | DISTINCT } [ selection , ]* selection

selection  ::= {single_valued_path_expression |
                identification_variable |
                OBJECT ( identification_variable) |
               aggregate_functions } [[ AS ] id ]
```

The SELECT clause consists of one or more of the following elements: a single identification variable that is defined in the FROM clause, a single-valued path expression that evaluates to object references or values, and an aggregate function. You can use the DISTINCT keyword to eliminate duplicate references.

A scalar-subselect is a subselect that returns a single value.

Examples using SELECT
Find all employees that earn more than the John employee:

```
SELECT OBJECT(e) FROM EmpBean ej, EmpBean eWHERE ej.name = 'John' and e.salary > ej.salary
```

Find all departments that have one or more employees who earn less than 20000:

```
SELECT DISTINCT e.dept FROM EmpBean e where e.salary < 20000
```

A query can have a path expression that evaluates to an arbitrary value:

```
SELECT e.dept.name FROM EmpBean e where e.salary < 20000
```

The previous query returns a collection of name values for the departments that have employees who earn less than 20000.

A query can return an aggregate value:

```
SELECT avg(e.salary) FROM EmpBean e
```

A query that retrieves the names and object references for underpaid employees follows:

```
SELECT e.name as name , object(e) as emp from EmpBean e where e.salary < 50000
```

ObjectGrid query WHERE clause

The WHERE clause contains search conditions that are composed of the elements presented below. When a search condition evaluates to TRUE, the tuple is added to the result set.

ObjectGrid query literals

A string literal is enclosed in single quotes. A single quotation mark that occurs within a string literal is represented by two single quotes, for example: 'Tom''s'.

A numeric literal can be any of the following values:

- An exact value such as 57, -957, or +66
- Any value supported by Java™ long type
- A decimal literal such as 57.5 or -47.02
- An approximate numeric value such as 7E3 or -57.4E-2
- Float types must include the "F" qualifier, for example 1.0F
- Long types must include the "L" qualifier, for example 123L

Boolean literals are TRUE and FALSE.

Temporal literals follow JDBC escape syntax based on the type of attribute:

- java.util.Date: yyyy-mm-ss
- java.sql.Date: yyyy-mm-ss
- java.sql.Time: hh-mm-ss
- java.sql.Timestamp: yyyy-mm-dd hh:mm:ss.f...
- java.util.Calendar: yyyy-mm-dd hh:mm:ss.f...

Enum literals are expressed using Java enum literal syntax using the fully qualified enum class name.

ObjectGrid query input parameters

You can specify input parameters by either using an ordinal position or by using a variable name. Writing queries that use input parameters is strongly encouraged, because using input parameters increases performance by allowing the ObjectGrid to catch the query plan between running actions.

An input parameter can be any of the following types: Byte, Short, Integer, Long, Float, Double, BigDecimal, BigInteger, String, Boolean, Char, java.util.Date, java.sql.Date, java.sql.Time, java.sql.Timestamp, java.util.Calendar, a Java SE 5 enum, an Entity or POJO Object, or a binary data string in the form of Java byte[].

An input parameter must not have a NULL value. To search for the occurrence of a NULL value, use the NULL predicate.

Positional Parameters

Positional input parameters are defined by using question mark followed by a positive number:

```
?[positive integer].
```

Positional input parameters are numbered starting at 1 and correspond to the arguments of the query; therefore, a query must not contain an input parameter that exceeds the number of input arguments.

Example: `SELECT e FROM Employee e WHERE e.city = ?1 and e.salary >= ?2`

Named Parameters

Named input parameters are defined using a variable name in the format: :[parameter name].

Example: `SELECT e FROM Employee e WHERE e.city = :city and e.salary >= :salary`

ObjectGrid query BETWEEN predicate

The BETWEEN predicate determines whether a given value lies between two other given values.

```
expression [NOT] BETWEEN expression-2 AND expression-3
```

Example 1

```
e.salary BETWEEN 50000 AND 60000
```

is equivalent to:

```
e.salary >= 50000 AND e.salary <= 60000
```

Example 2

```
e.name NOT BETWEEN 'A' AND 'B'
```

is equivalent to:

```
e.name < 'A' OR e.name > 'B'
```

ObjectGrid query IN predicate

The IN predicate compares a value to a set of values. You can use the IN predicate in one of two forms:

```
expression [NOT] IN ( subselect )expression [NOT] IN ( value1, value2, .... )
```

The ValueN value can either be a literal value or an input parameter. The expression cannot evaluate to a reference type.

Example 1

```
e.salary IN ( 10000, 15000 )
```

is equivalent to

```
( e.salary = 10000 OR e.salary = 15000 )
```

Example 2

```
e.salary IN ( select e1.salary from EmpBean e1 where e1.dept.deptno = 10)
```

is equivalent to

```
e.salary = ANY ( select e1.salary from EmpBean e1 where e1.dept.deptno = 10)
```

Example 3

```
e.salary NOT IN ( select e1.salary from EmpBean e1 where e1.dept.deptno = 10)
```

is equivalent to

```
e.salary <> ALL ( select e1.salary from EmpBean e1 where e1.dept.deptno = 10)
```

ObjectGrid query LIKE predicate

The LIKE predicate searches a string value for a certain pattern.

```
string-expression [NOT] LIKE pattern [ ESCAPE escape-character ]
```

The pattern value is a string literal or parameter marker of type string in which the underscore (_) stands for any single character and percent (%) stands for any sequence of characters, including an empty sequence. Any other character stands for itself. The escape character can be used to search for character _ and %. The escape character can be specified as a string literal or as an input parameter.

If the string-expression is null, then the result is unknown.

If both string-expression and pattern are empty, then the result is true.

Example

```
'' LIKE '' is true
'' LIKE '%' is true
e.name LIKE '12%3' is true for '123' '12993' and false for '1234'
e.name LIKE 's_me' is true for 'some' and 'same', false for 'soome'
e.name LIKE '/_foo' escape '/' is true for '_foo', false for 'afoo'
e.name LIKE '//_foo' escape '/' is true for '/afoo' and for '/bfoo'
e.name LIKE '///_foo' escape '/' is true for '/_foo' but false for '/afoo'
```

ObjectGrid query NULL predicate

The NULL predicate tests for null values.

```
{single-valued-path-expression | input_parameter} IS [NOT] NULL
```

Example

```
e.name IS NULL
e.dept.name IS NOT NULL
e.dept IS NOT NULL
```

ObjectGrid query EMPTY collection predicate

Use the EMPTY collection predicate to test for an empty collection.

To test if a multi-valued relationship is empty, use the following syntax:

```
collection-valued-path-expression IS [NOT] EMPTY
```

Example

Empty collection predicate To find all the departments that have no employees:

```
SELECT OBJECT(d) FROM DeptBean d WHERE d.emps IS EMPTY
```

ObjectGrid query MEMBER OF predicate

The following expression tests whether the object reference that is specified by the single valued path expression or input parameter is a member of the designated collection. If the collection valued path expression designates an empty collection, then the value of the MEMBER OF expression is FALSE.

```
{ single-valued-path-expression | input_parameter } [ NOT ] MEMBER [ OF ] collection-valued-path-expression
```

Example

Find employees that are not members of a given department number:

```
SELECT OBJECT(e) FROM EmpBean e , DeptBean d
WHERE e NOT MEMBER OF d.emps AND d.deptno = ?1
```

Find employees whose manager is a member of a given department number:

```
SELECT OBJECT(e) FROM EmpBean e, DeptBean d
WHERE e.dept.mgr MEMBER  OF d.emps  and d.deptno=?1
```

ObjectGrid query EXISTS predicate

The EXISTS predicate tests for the presence or absence of a condition that specified by a subselect.

```
EXISTS ( subselect )
```

The result of EXISTS is true if the subselect returns at least one value, otherwise the result is false.

To negate an EXISTS predicate, precede the predicate with the NOT logical operator.

Example

Return departments that have at least one employee that earns more than 1000000:

```
SELECT  OBJECT(d) FROM  DeptBean d
WHERE EXISTS ( SELECT  e  FROM IN (d.emps) e WHERE  e.salary > 1000000 )
```

Return departments that have no employees:

```
SELECT OBJECT(d) FROM DeptBean d
WHERE NOT EXISTS  ( SELECT e FROM IN (d.emps) e)
```

You can also rewrite the previous query like in the following example:

```
SELECT OBJECT(d) FROM DeptBean d WHERE SIZE(d.emps)=0
```

ObjectGrid query ORDER BY clause

The ORDER BY clause specifies an ordering of the objects in the result collection. An example follows:

ORDER BY [order_element ,]* order_element order_element ::={ path-expression }[ASC | DESC]

The path expression must specify a single-valued field that is a primitive type of byte, short, int, long, float, double, char, or a wrapper type of Byte, Short, Integer, Long, Float, Double, BigDecimal, String, Character, java.util.Date, java.sql.Date, java.sql.Time, java.sql.Timestamp and java.util.Calendar. The ASC order element specifies that the results are displayed in ascending order, which is the default. A DESC order element specifies that the results are displayed in descending order.

Example

Return department objects. Display the department numbers in decreasing order:

```
SELECT OBJECT(d) FROM DeptBean d ORDER BY d.deptno DESC
```

Return employee objects, sorted by department number and name:

```
SELECT OBJECT(e) FROM EmpBean e ORDER BY e.dept.deptno ASC, e.name DESC
```

ObjectGrid query aggregation functions

Aggregation functions operate on a set of values to return a single scalar value. You can use these functions in the select and subselect methods. The following example illustrates an aggregation:

```
SELECT SUM (e.salary) FROM EmpBean e WHERE e.dept.deptno =20
```

This aggregation computes the total salary for department 20.

The aggregation functions are: AVG, COUNT, MAX, MIN, and SUM. The syntax of an aggregation function is illustrated in the following example:

```
aggregation-function ( [ ALL | DISTINCT ] expression )
```

or:

```
COUNT( [ ALL | DISTINCT ] identification-variable )
```

The DISTINCT option eliminates duplicate values before applying the function. The ALL option is the default option, and does not eliminate duplicate values. Null values are ignored in computing the aggregate function except when you use the COUNT(identification-variable) function, which returns a count of all the elements in the set.

Defining return type

The MAX and MIN functions can apply to any numeric, string or date-time data type and return the corresponding data type. The SUM and AVG functions take a numeric type as input. The AVG function returns a double type. The SUM function returns a long type if the input type is an integer type, except when the input is a Java BigInteger type, then the function returns a Java BigInteger type. The SUM function returns a double type if the input type is not an integer type, except when the input is a Java BigDecimal type, then the function returns a Java BigDecimal type. The COUNT function can take any data type except collections, and returns a long type.

When applied to an empty set, the SUM, AVG, MAX, and MIN functions can return a null value. The COUNT function returns zero (0) when it is applied to an empty set.

Using GROUP BY and HAVING clauses

The set of values that is used for the aggregate function is determined by the collection that results from the FROM and WHERE clause of the query. You can divide the set into groups and apply the aggregation function to each group. To perform this action, use a GROUP BY clause in the query. The GROUP BY clause defines grouping members, which comprise a list of path expressions. Each path expression specifies a field that is a primitive type of byte, short, int, long, float, double, boolean, char, or a wrapper type of Byte, Short, Integer, Long, Float, Double, BigDecimal, String, Boolean, Character, java.util.Date, java.sql.Date, java.sql.Time, java.sql.Timestamp, java.util.Calendar or a Java SE 5 enum.

The following example illustrates the use of the GROUP BY clause in a query that computes the average salary for each department:

```
SELECT e.dept.deptno, AVG ( e.salary) FROM EmpBean e GROUP BY e.dept.deptno
```

In division of a set into groups, a NULL value is considered equal to another NULL value.

Groups can be filtered using a HAVING clause that tests group properties before involving aggregate functions or grouping members. This filtering is similar to how the WHERE clause filters tuples (that is, records of the return collection values) from the FROM clause. An example of the HAVING clause follows:

```
SELECT e.dept.deptno, AVG ( e.salary) FROM EmpBean e
GROUP BY e.dept.deptno
HAVING COUNT(e) > 3 AND e.dept.deptno > 5
```

This query returns the average salary for departments that have more than three employees and the department number is greater than five.
You can use a HAVING clause without a GROUP BY clause. In this case, the entire set is treated as a single group, to which the HAVING clause is applied.

- ObjectGrid query Backus-Naur Form
  A summary of the ObjectGrid Query Backus-Naur Form (BNF) Notation follows.

Related concepts:
Querying data in multiple time zones
Data for different time zones
Using the ObjectQuery API
EntityManager Query API

ObjectGrid query Backus-Naur Form

A summary of the ObjectGrid Query Backus-Naur Form (BNF) Notation follows.

Table 1. Key to BNF summary

Representation	Description
`{...}`	Grouping
`[...]`	Optional constructs
bold	Keywords
*	Zero or more
|	Alternates

```
ObjectGrid QL ::=select_clause from_clause [where_clause] [group_by_clause]
                 [having_clause] [order_by_clause]

from_clause ::=FROM identification_variable_declaration
                 [,identification_variable_declaration]*

identification_variable_declaration ::=collection_member_declaration |
                 range_variable_declaration

collection_member_declaration ::=IN ( collection_valued_path_expression |
                 single_valued_navigation) [AS] identifier | [LEFT [OUTER]
                 | INNER] JOIN collection_valued_path_expression |
                 single_valued_navigation [AS] identifier

range_variable_declaration ::=abstract_schema_name [AS] identifier
```

```
single_valued_path_expression ::={single_valued_navigation | identification_variable}.
                { state_field | state_field.value_object_attribute } | single_valued_navigation

single_valued_navigation ::=identification_variable.[ single_valued_association_field. ]*
                single_valued_association_field

collection_valued_path_expression ::=identification_variable.[
                single_valued_association_field. ]* collection_valued_association_field

select_clause ::= SELECT [DISTINCT] [ selection , ]* selection

selection ::= {single_valued_path_expression |identification_variable | OBJECT
                ( identification_variable) |aggregate_functions } [[ AS ] id ]

order_by_clause ::= ORDER BY [ {identification_variable.[ single_valued_association_field.
                ]*state_field} [ASC|DESC],]* {identification_variable.[
                single_valued_association_field. ]*state_field}[ASC|DESC]

where_clause ::= WHERE conditional_expression

conditional_expression ::= conditional_term | conditional_expression OR conditional_term

conditional_term ::= conditional_factor | conditional_term AND conditional_factor

conditional_factor ::= [NOT] conditional_primary

conditional_primary ::= simple_cond_expression | (conditional_expression)

simple_cond_expression ::= comparison_expression | between_expression | like_expression |
                in_expression | null_comparison_expression | empty_collection_comparison_expression |
                exists_expression | collection_member_expression

between_expression ::= numeric_expression [NOT] BETWEEN numeric_expression
                AND numeric_expression | string_expression [NOT] BETWEEN
                string_expression AND string_expression | datetime_expression [NOT]
                BETWEEN datetime_expression AND datetime_expression

in_expression ::= identification_variable.[ single_valued_association_field. ]state_field
                [*NOT] IN { (subselect) | ( atom ,]* atom) }

atom ::= { string_literal | numeric_literal | input_parameter }

like_expression ::=string_expression [NOT] LIKE {string_literal | input_parameter}
                [ESCAPE {string_literal | input_parameter}]

null_comparison_expression ::= {single_valued_path_expression | input_parameter} IS
                [ NOT ] NULL

empty_collection_comparison_expression ::= collection_valued_path_expression IS
                [NOT] EMPTY

collection_member_expression ::={ ssingle_valued_path_expression | input_parameter }[
                NOT ] MEMBER    [ OF ]collection_valued_path_expression

exists_expression ::= EXISTS {(subselect)}

subselect ::= SELECT [{ ALL | DISTINCT }] subselection from_clause
                [where_clause] [group_by_clause] [having_clause]

subselection ::= {single_valued_path_expression |identification_variable |
                aggregate_functions }

group_by_clause ::= GROUP BY[single_valued_path_expression,]*
                single_valued_path_expression

having_clause ::= HAVING conditional_expression

comparison_expression ::= numeric_ expression comparison_operator { numeric_expression
                | {SOME | ANY | ALL} (subselect) } | string_expression
                comparison_operator {

string_expression | {SOME | ANY | ALL}(subselect) } |

datetime_expression comparison_operator {

datetime_expression {SOME | ANY | ALL}(subselect) } |

boolean_expression {=|<>} {

boolean_expression {SOME | ANY | ALL}(subselect) } |

entity_expression {=|<>} {

entity_expression {SOME| ANY | ALL}(subselect) }

comparison_operator ::= = | > | >= | < | <= | <>

string_expression ::= string_primary | (subselect)

string_primary ::=state_field_path_expression |string_literal | input_parameter |
                functions_returning_strings

datetime_expression ::= datetime_primary |(subselect)
```

```
datetime_primary ::=state_field_path_expression | string_literal | long_literal
                | input_parameter | functions_returning_datetime

boolean_expression ::= boolean_primary |(subselect)

boolean_primary ::=state_field_path_expression | boolean_literal | input_parameter

entity_expression ::=single_valued_association_path_expression |
                identification_variable | input_parameter

numeric_expression ::= simple_numeric_expression |(subselect)

simple_numeric_expression ::= numeric_term | numeric_expression {+|-} numeric_term

numeric_term ::= numeric_factor | numeric_term {*|/} numeric_factor

numeric_factor ::= {+|-} numeric_primary

numeric_primary ::= single_valued_path_expression | numeric_literal |
                ( numeric_expression ) | input_parameter | functions

aggregate_functions :=

AVG([ALL|DISTINCT] identification_variable.
                [ single_valued_association_field. ]*state_field) |

COUNT([ALL|DISTINCT] {single_valued_path_expression |
                identification_variable}) |

MAX([ALL|DISTINCT] identification_variable.[
                single_valued_association_field. ]*state_field) |

MIN([ALL|DISTINCT] identification_variable.[
                single_valued_association_field. ]*state_field) |

SUM([ALL|DISTINCT] identification_variable.[
                single_valued_association_field. ]*state_field)

functions ::=

ABS (simple_numeric_expression) |

CONCAT (string_primary , string_primary) |

LOWER (string_primary) |

LENGTH(string_primary) |

LOCATE(string_primary, string_primary [, simple_numeric_expression]) |

MOD (simple_numeric_expression, simple_numeric_expression) |

SIZE (collection_valued_path_expression) |

SQRT (simple_numeric_expression) |

SUBSTRING (string_primary, simple_numeric_expression[, simple_numeric_expression]) |

UPPER (string_primary) |

TRIM ([[LEADING | TRAILING | BOTH] [trim_character]
                FROM] string_primary)
```

Programming for transactions in Java applications

When you write a Java™ application that requires transactions, you must consider issues such as lock handling, collision handling, and transaction isolation.

- Interacting with data in a transaction for Java applications
  Use sessions to interact with data, including insert and update operations.
- Using locking
  Locks have life cycles and different types of locks are compatible with others in various ways. Locks must be handled in the correct order to avoid deadlock scenarios.

Transaction processing overview

WebSphere® eXtreme Scale uses transactions as its mechanism for interaction with data.

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.

- 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.
- 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.
- 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.
- 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.
- Deadlocks
  Deadlocks can occur when two transactions try to update the same cache entry.
- 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.
- 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.
- 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.
- JMS for distributed transaction changes
  Use Java™ Message Service (JMS) for distributed transaction changes between different tiers or in environments on mixed platforms.

Related tasks:
Resolving lock timeout exceptions

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.

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.

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. 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:

- 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.

- 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 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.

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.

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.

Related tasks:
Interacting with data in a transaction for Java applications

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. 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.

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.

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.

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

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.

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 WebSphereTranscationCallback plug-in.
8.5+

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.

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.

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.

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"/>
```

Related concepts:
Tuning the copy mode
Improving performance with byte array maps
Related reference:
ObjectGrid descriptor XML file

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:

No locking
: Running without the locking setting is the fastest. If you are using read-only data, then you might not need locking.

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.

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.

Optimistic no versioning locking
: This locking strategy allows you to disable version control. This is important because near cache is only enabled if you are doing Optimistic locking. With the current implementation, you need a plug-in or callback handler to handle version control. However, using the OPTIMISTIC_NO_VERSIONING locking strategy to disable version control on the client and only enable it on the server, is an additional performance savings.

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.

8.5 To avoid a java.lang.IllegalStateException exception, you must call the setNumberOfLockBuckets method before the initialize or getSession methods on the ObjectGrid instance. The setNumberOfLockBuckets method parameter is a Java primitive integer that specifies the number of lock buckets to use. Using a prime number can allow for a uniform distribution of map entries over the lock buckets. A good starting point for best performance is to set the number of lock buckets to about 10 percent of the expected number of BackingMap entries.

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.
- 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.

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.
- 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.

Optimistic no versioning

You can enable OPTIMISTIC_NO_VERSIONING locking either through the client override XML file or programmatically. See the following examples of both approaches:
Client override XML file example

```
<objectGrid name="lockStrategyGrid">
            <backingMap name="opt_with_noversion" lockStrategy="OPTIMISTIC_NO_VERSIONING"/>
            <backingMap name="opt_with_none" lockStrategy="NONE"/>
            <backingMap name="optnoversion_with_opt" lockStrategy="OPTIMISTIC"/>
            <backingMap name="optnoversion_with_none" lockStrategy="NONE"/>
        </objectGrid>
```

Programmatic example

```
ObjectGridConfiguration lsConfig = ObjectGridConfigFactory.createObjectGridConfiguration("lockStrategyGrid");
                BackingMapConfiguration oMapWithOVConfig =
ObjectGridConfigFactory.createBackingMapConfiguration("opt_with_noversion");
                oMapWithOVConfig.setLockStrategy(LockStrategy.OPTIMISTIC_NO_VERSIONING);
                lsConfig.addBackingMapConfiguration(oMapWithOVConfig);
```

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.
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.

Related tasks:
Configuring a locking strategy in the ObjectGrid descriptor XML file
Configuring and implementing locking in Java applications
Configuring the lock timeout value in the ObjectGrid descriptor XML file

JMS for distributed transaction changes

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 Configuring Java Message Service (JMS)-based client synchronization and Configuring peer-to-peer replication with JMS 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 Distributing changes between peer JVMs.

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.

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.

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.

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.

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.

Developing eXtreme Scale client components to use transactions

8.5+ 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.LocalTransction 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.

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 = ((XSConnection)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 = ((XSConnection)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 = ((XSConnection)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 = ((XSConnection)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 = ((XSConnection)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(); }
            }
        }
    /**
         /**
```

Previous topic: 8.5+ Securing J2C client connections
Next topic: 8.5+ Administering J2C client connections
Related information:
Resource reference benefits

Using locking

Locks have life cycles and different types of locks are compatible with others in various ways. Locks must be handled in the correct order to avoid deadlock scenarios.

- Configuring and implementing locking in Java applications
  You can define an optimistic, a pessimistic, or no locking strategy on each BackingMap in the WebSphere® eXtreme Scale configuration.
- Example: flush method lock ordering
  Invoking the flush method on the ObjectMap interface before a commit can introduce additional lock ordering considerations. The flush method is typically used to force changes that are made to the map out to the backend through the Loader plug-in.
- Implementing exception handling in locking scenarios for Java applications
  To prevent locks from being held for excessive amounts of time when a LockTimeoutException exception or a LockDeadlockException exception occurs, your application must catch unexpected exceptions and call the rollback method when an unexpected event occurs.
- Map entry locks with query and indexes
  The eXtreme Scale Query APIs and the MapRangeIndex indexing plug-in interact with locks. You can increase concurrency and decrease deadlocks when you are using the pessimistic locking strategy for maps.
- Java examples for transaction isolation
  Transaction isolation defines how the changes that are made by one operation become visible to other concurrent operations. You can use the following examples to define the transaction isolation level in your Java application.
- Optimistic collision exception
  You can receive an OptimisticCollisionException directly, or receive it with an ObjectGridException.
- Running parallel business logic on the data grid (DataGrid API)
  The DataGrid API provides a simple programming interface to run business logic over all or a subset of the data grid in parallel with where the data is located.

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.
In Java applications, locks are also acquired when the applications uses the find methods on an index, or does a query.

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 modes and the behavior of equivalent methods:

Table 1. Lock modes and ObjectMap method equivalents

Lock mode	Method equivalent
S (shared)	get() and getAll()
U (upgradeable)	getForUpdate(), getAllForUpdate()
X (exclusive)	getNextKey() and commit()

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 2. 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

Related tasks:
Troubleshooting deadlocks

Implementing exception handling in locking scenarios for Java™ applications

To prevent locks from being held for excessive amounts of time when a LockTimeoutException exception or a LockDeadlockException exception occurs, your application must catch unexpected exceptions and call the rollback method when an unexpected event occurs.

Procedure

1. Catch the exception, and display resulting message.

   ```
   try {
   ...
   } catch (ObjectGridException oe) {
   System.out.println(oe);
   }
   ```

   The following exception displays as a result:

   ```
   com.ibm.websphere.objectgrid.plugins.LockDeadlockException: Message
   ```

   This message represents the string that is passed as a parameter when the exception is created and thrown.
2. Roll back the transaction after an exception:

   ```
   Session sess = ...;
   ObjectMap person = sess.getMap("PERSON");
   boolean activeTran = false;
   try
   {
       sess.begin();
       activeTran = true;
       Person p = (IPerson)person.get("Lynn");
       // Lynn had a birthday, so we make her 1 year older.
       p.setAge( p.getAge() + 1 );
       person.put( "Lynn", p );
       sess.commit();
       activeTran = false;
   }
   finally
   {
       if ( activeTran ) sess.rollback();
   }
   ```

   The `finally` block in the snippet of code ensures that a transaction is rolled back when an unexpected exception occurs. It not only handles a LockDeadlockException exception, but any other unexpected exception that might occur. The `finally` block handles the case where an exception occurs during a commit method invocation. This example is not the only way to deal with unexpected exceptions, and there might be cases where an application wants to catch some of the unexpected exceptions that can occur and display one of its application exceptions. You can add catch blocks as appropriate, but the application must ensure that the snippet of code does not exit without completing the transaction.

Configuring a locking strategy in the ObjectGrid descriptor XML file

You can define an optimistic, a pessimistic, or no locking strategy on each BackingMap in the WebSphere® eXtreme Scale configuration.

Before you begin

Decide which locking strategy you want to use. For more information, see Locking strategies.

About this task

You can configure each BackingMap instance to use one of the following locking strategies:

- Optimistic locking mode (default)
- Pessimistic locking mode

- None

Procedure

- **Configure a pessimistic locking strategy**
  - With the lockStrategy attribute in the ObjectGrid descriptor 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="test">
            <backingMap name="pessimisticMap"
                lockStrategy="PESSIMISTIC"/>
        </objectGrid>
    </objectGrids>
</objectGridConfig>
```

- **Configure an optimistic locking strategy**
  - With the lockStrategy attribute in the ObjectGrid descriptor 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="test">
            <backingMap name="optimisticMap"
                lockStrategy="OPTIMISTIC"/>
        </objectGrid>
    </objectGrids>
</objectGridConfig>
```

- **Configure a no locking strategy**
  - With the lockStrategy attribute in the ObjectGrid descriptor 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="test">
            <backingMap name="noLockingMap"
                lockStrategy="NONE"/>
        </objectGrid>
    </objectGrids>
</objectGridConfig>
```

Related concepts:
Distributing changes between peer JVMs
JMS event listener
Locking strategies
Related tasks:
Configuring and implementing locking in Java applications
Configuring the lock timeout value in the ObjectGrid descriptor XML file
Related reference:
ObjectGrid descriptor XML file

Configuring the lock timeout value in the ObjectGrid descriptor XML file

The lock timeout value on a BackingMap instance is used to ensure that an application does not wait endlessly for a lock mode to be granted because of a deadlock condition that occurs due to an application error.

Before you begin

To configure the lock timeout value, the locking strategy must be set to either OPTIMISTIC or PESSIMISTIC. See Configuring a locking strategy in the ObjectGrid descriptor XML file for more information.

About this task

When a LockTimeoutException exception occurs, the application must determine if the timeout is occurring because the application is running slower than expected, or if the timeout occurred because of a deadlock condition. If an actual deadlock condition occurred, then increasing the lock wait timeout value does not eliminate the exception. Increasing the timeout results in the exception taking longer to occur. However, if increasing the lock wait timeout value does eliminate the exception, then the problem occurred because the application was running slower than expected. The application in this case must determine why performance is slow.

To prevent deadlocks from occurring, the lock manager has a default timeout value of 15 seconds. If the timeout limit is exceeded, a LockTimeoutException exception occurs. If your system is heavily loaded, the default timeout value might cause the LockTimeoutException exceptions to occur when no deadlock exists. In this situation, you can increase the lock timeout value programmatically or in the ObjectGrid descriptor XML file.

Procedure

Configure the lock timeout value with the lockTimeout attribute in the ObjectGrid descriptor 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="test">
            <backingMap name="optimisticMap"
                lockStrategy="OPTIMISTIC"
                lockTimeout="60"/>
        </objectGrid>
    </objectGrids>
</objectGridConfig>
```

Related concepts:
Locking strategies
Related tasks:
Configuring a locking strategy in the ObjectGrid descriptor XML file
Configuring and implementing locking in Java applications

Map entry locks with query and indexes

The eXtreme Scale Query APIs and the MapRangeIndex indexing plug-in interact with locks. You can increase concurrency and decrease deadlocks when you are using the pessimistic locking strategy for maps.

Overview

The ObjectGrid Query API allows SELECT queries over ObjectMap cache objects and entities. When a query is run, the query engine uses a MapRangeIndex when possible to find matching keys that match values in the query's WHERE clause or to bridge relationships. When an index isn't available, the query engine will scan each entry in one or more maps to find the appropriate entries. Both the query engine and index plug-ins acquire locks to verify consistent data, depending on the locking strategy, transaction isolation level, and transaction state.

Locking with the HashIndex plug-in

The eXtreme Scale HashIndex plug-in allows finding keys that are based on a single attribute that is stored in the cache entry value. The index stores the indexed value in a separate data structure from the cache map. The index validates the keys against map entries before returning to the user to try to achieve an accurate result set. When you are using the pessimistic lock strategy and the index against a local ObjectMap instance (versus a client/server ObjectMap), the index acquires locks for each matching entry. When you are using optimistic locking or a remote ObjectMap, the locks are always immediately released.

The type of lock that is acquired depends upon the forUpdate argument that is passed to the ObjectMap.getIndex method. The forUpdate argument specifies the type of lock that the index should acquire. If false, a shareable (S) lock is acquired and if true, an upgradeable (U) lock is acquired.

If the lock type is shareable, the transaction isolation setting for the session is applied and affects the duration of the lock. See the transaction isolation topic for details on how transaction isolation is used to add concurrency to applications.

Shared locks with query

The eXtreme Scale query engine acquires S locks when needed to introspect the cache entries to discover if they satisfy the query's filter criteria. When you are using repeatable read transaction isolation with pessimistic locking, the S locks are only retained for the elements that are included in the query result. The locks are released for any entries that are not included in the result. If you are using a lower transaction isolation level or optimistic locking, the S locks are not retained.

Shared locks with client to server query

When you are using the eXtreme Scale query from a client, the query typically runs on the server unless all of the maps or entities referenced in the query are local to the client, for example, a client-replicated map or a query result entity. All queries that run in a read/write transaction retain S locks. If the transaction is not a read/write transaction, then a session is not retained on the server and the S locks are released.

A read/write transaction is only routed to a primary partition and a session is maintained on the server for the client session. A transaction can be promoted to read/write under the following conditions:

1. Any map that is configured to use pessimistic locking is accessed with the ObjectMap get and getAll API methods or the EntityManager.find methods.
2. The transaction is flushed, causing updates to be sent to the server.
3. Any map that is configured to use optimistic locking is accessed with the ObjectMap.getForUpdate or EntityManager.findForUpdate method.

Upgradeable locks with query

Shareable locks are useful when concurrency and consistency are important. It guarantees that an entry's value does not change for the life of the transaction. No other transaction can change the value while any other S locks are held, and only one other transaction can establish an intent to update the entry.

Upgradeable locks are used to identify the intent to update a cache entry when using the pessimistic lock strategy. It allows synchronization between transactions that want to modify a cache entry. Transactions can still view the entry using an S lock, but other transactions are prevented from acquiring a U lock or an X lock. In many scenarios, acquiring a U lock without first acquiring an S lock is necessary to avoid deadlocks. See the Pessimistic Locking Mode topic for common deadlock examples.

The ObjectQuery and EntityManager Query interfaces provide the setForUpdate method to identify the intended use for the query result. Specifically, the query engine acquires U locks instead of S locks for each map entry that is involved in the query result:

```
ObjectMap orderMap = session.getMap("Order");
ObjectQuery q = session.createQuery("SELECT o FROM Order o WHERE o.orderDate=?1");
q.setParameter(1, "20080101");
q.setForUpdate(true);
session.begin();
// Run the query.  Each order has  U lock
Iterator result = q.getResultIterator();
// For each order, update the status.
while(result.hasNext()) {
    Order o = (Order) result.next();
    o.status = "shipped";
    orderMap.update(o.getId(), o);
}
// When committed, the
session.commit();
```

```
Query q = em.createQuery("SELECT o FROM Order o WHERE o.orderDate=?1");
q.setParameter(1, "20080101");
q.setForUpdate(true);
emTran.begin();
// Run the query.  Each order has  U lock
Iterator result = q.getResultIterator();
// For each order, update the status.
while(result.hasNext()) {
    Order o = (Order) result.next();
    o.status = "shipped";
}
tmTran.commit();
```

When the setForUpdate attribute is enabled, the transaction is automatically converted to a read/write transaction and the locks are held on the server as expected. If the query cannot use any indexes, then the map must be scanned which results in temporary U locks for map entries that do not satisfy the query result. U locks are held for entries that are included in the result.

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:

- With the txIsolation attribute in the ObjectGrid descriptor XML file. For more information, see ObjectGrid descriptor XML file.
- 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.

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.

Related reference:
Java examples for transaction isolation

Optimistic collision exception

You can receive an OptimisticCollisionException directly, or receive it with an ObjectGridException.

The following code is an example of how to catch the exception and then display its message:

```
try {
...
} catch (ObjectGridException oe) {
    System.out.println(oe);
}
```

Exception cause

OptimisticCollisionException is created in a situation in which two different clients try to update the same map entry at relatively the same time. For example, if one client attempts to commit a session and update the map entry after another client reads the data before the commit, that data is then incorrect. The exception is created when the other client attempts to commit the incorrect data.

Retrieving the key that triggered the exception

It might be useful, when troubleshooting such an exception, to retrieve the key corresponding to the entry that triggered the exception. The benefit of the OptimisticCollisionException is it contains the getKey method, which returns the object representing that key. The following example demonstrates how to retrieve and print the key when catching OptimisticCollisionException:

```
try {
...
} catch (OptimisticCollisionException oce) {
    System.out.println(oce.getKey());
}
```

ObjectGridException causes an OptimisticCollisionException

OptimisticCollisionException might be the cause of ObjectGridException displaying. If this is the case, you can use the following code to determine the exception type and print out the key. The following code uses the findRootCause utility method as described in the section below.

```
try {
...
}
catch (ObjectGridException oe) {
    Throwable Root = findRootCause( oe );
    if (Root instanceof OptimisticCollisionException) {
        OptimisticCollisionException oce = (OptimisticCollisionException)Root;
        System.out.println(oce.getKey());
    }
}
```

General exception handling technique

Knowing the root cause of a Throwable object is helpful in isolating the source of an error. The following example demonstrates how an exception handler uses a utility method to find the root cause of the Throwable object.

Example:

```
static public Throwable findRootCause( Throwable t )
{
     // Start with Throwable that occurred as the root.
     Throwable root = t;

     // Follow cause chain until last Throwable in chain is found.
     Throwable cause = root.getCause();
     while ( cause != null )
     {
        root = cause;
```

```
            cause = root.getCause();
        }

        // Return last Throwable in the chain as the root cause.
        return root;
    }
```

Running parallel business logic on the data grid (DataGrid API)

The DataGrid API provides a simple programming interface to run business logic over all or a subset of the data grid in parallel with where the data is located.

- DataGrid APIs and partitioning
  With the DataGrid APIs, a client can send requests to one partition, a subset of partitions, or all the partitions in a data grid. The client can specify a list of keys, and WebSphere® eXtreme Scale determines the set of partitions that are hosting the keys. The request is then sent to all the partitions in the set in parallel and the client waits for the results. The client can also send requests without specifying keys, therefore, requests are sent to all partitions.
- DataGrid agents and entity-based Maps
  A map contains key objects and value objects. The key object is a generated tuple as is the value.
- DataGrid API example
  The DataGrid APIs support two common grid programming patterns: parallel map and parallel reduce.

Related information:
DataGrid API

DataGrid APIs and partitioning

With the DataGrid APIs, a client can send requests to one partition, a subset of partitions, or all the partitions in a data grid. The client can specify a list of keys, and WebSphere® eXtreme Scale determines the set of partitions that are hosting the keys. The request is then sent to all the partitions in the set in parallel and the client waits for the results. The client can also send requests without specifying keys, therefore, requests are sent to all partitions.

Agents that are deployed to the data grid do not work in client mode. These agents work directly against the primary shard. Working directly against the primary shard results in maximum performance, allowing tens of thousands or more transactions per second because the agent works with the data at full memory speeds. Working directly with the primary shard also means that an agent can only see data that is within that shard. This provides some interesting opportunities that cannot be done on a client.

A typical eXtreme Scale client must be able determine the partition from the transaction, because the client needs to route the request. If an agent is directly attached to a shard, then no routing is needed. All requests go against that shard. Because the agent is directly attached to a shard, data in other maps in the shard can be accessed without worrying about common partitioning keys, and so on, because no routing occurs.

Related information:
DataGrid API

DataGrid agents and entity-based Maps

A map contains key objects and value objects. The key object is a generated tuple as is the value.

The key object is a generated tuple as is the value. An agent is normally provided with the application specified key objects. This will be the key objects used by the application or Tuples if it is an entity Map. An application using Entities will not want to deal with Tuples directly and would prefer to work with the Java™ objects mapped to the Entity.

Therefore, an Agent class can implement the EntityAgentMixin interface. This forces the class to implement one more method, getClassForEntity(). This returns the entity class to use with the agent on the server side. The keys are converted to this Entity before invoking the process and reduce methods.

This is a different semantic to a non EntityAgentMixin agent where those methods are provided with just the keys. An agent implementing EntityAgentMixin receives the Entity object which includes keys and values in one object.
Note: If the entity does not exist on the server, the keys are the raw Tuple format of the key instead of the managed entity.
Related information:
DataGrid API

DataGrid API example

The DataGrid APIs support two common grid programming patterns: parallel map and parallel reduce.

Parallel Map

The parallel map allows the entries for a set of keys to be processed and returns a result for each entry processed. The application makes a list of keys and receives a Map of key/result pairs after invoking a Map operation. The result is the result of applying a function to the entry of each key. The function is supplied by the application.

MapGridAgent call flow

When the AgentManager.callMapAgent method is invoked with a collection of keys, the MapGridAgent instance is serialized and sent to each primary partition that the keys resolve to. This means that any instance data stored in the agent can be sent to the server. Each primary partition therefore has one instance of the agent. The process method is invoked for each instance one time for each key that resolves to the partition. The result of each process method is then serialized back to the client and returned to the caller in a Map instance, where the result is represented as the value in the map.

When the AgentManager.callMapAgent method is invoked without a collection of keys, the MapGridAgent instance is serialized and sent to every primary partition. This means that any instance data stored in the agent can be sent to the server. Each primary partition therefore has one instance (partition) of the agent. The processAllEntries method is invoked for each partition. The result of each processAllEntries method is then serialized back to the client and returned to the caller in a Map instance. The following example assumes that a Person entity exists with the following shape:

```
import com.ibm.websphere.projector.annotations.Entity;
import com.ibm.websphere.projector.annotations.Id;
@Entity
public class Person
{
  @Id String ssn;
  String firstName;
  String surname;
  int age;
}
```

The application supplied function is written as a class that implements the MapAgentGrid interface. An example agent that shows a function to return the age of a Person multiplied by two.

```
public class DoublePersonAgeAgent implements MapGridAgent, EntityAgentMixin
{
  private static final long serialVersionUID = -2006093916067992974L;

  int lowAge;
  int highAge;

  public Object process(Session s, ObjectMap map, Object key)
  {
    Person p = (Person)key;
    return new Integer(p.age * 2);
  }

  public Map processAllEntries(Session s, ObjectMap map)
  {
    EntityManager em = s.getEntityManager();
    Query q = em.createQuery("select p from Person p where p.age > ?1 and p.age < ?2");
    q.setParameter(1, lowAge);
    q.setParameter(2, highAge);
    Iterator iter = q.getResultIterator();
    Map<Person, Interger> rc = new HashMap<Person, Integer>();
         while(iter.hasNext())
    {
                        Person p = (Person)iter.next();
                        rc.put(p, (Integer)process(s, map, p));
                }
                return rc;
        }
        public Class getClassForEntity()
        {
                return Person.class;
        }
}
```

The previous example shows the Map agent for doubling a Person. The first process method is supplied with the Person to work with and returns double the age of that entry. The second process method is called for each partition and finds all Person objects with an age between lowAge and highAge and returns their ages doubled.

```
Session s = grid.getSession();
ObjectMap map = s.getMap("Person");
AgentManager amgr = map.getAgentManager();

DoublePersonAgeAgent agent = new DoublePersonAgeAgent();

// make a list of keys
ArrayList<Person> keyList = new ArrayList<Person>();
Person p = new Person();
p.ssn = "1";
keyList.add(p);
p = new Person ();
p.ssn = "2";
keyList.add(p);

// get the results for those entries
Map<Tuple, Object> = amgr.callMapAgent(agent, keyList);
// Close the session (optional in Version 7.1.1 and later) for improved performance
s.close();
```

The previous example shows a client obtaining a Session and a reference to the Person Map. The agent operation is performed against a specific Map. The AgentManager interface is retrieved from that Map. An instance of the agent to invoke is created and any necessary state is added to the object by setting attributes, there are none in this case. A list of keys are then constructed. A Map with the values for person 1 doubled, and the same values for person 2 are returned.

The agent is then invoked for that set of keys. The agents process method is invoked on each partition with some of the specified keys in the grid in parallel. A Map is returned providing the merged results for the specified key. In this case, a Map with the values holding the age for person 1 doubled and the same for person 2 is returned.

If the key does not exist, the agent is still invoked. This invocation gives the agent the opportunity to create the map entry. If you are using an EntityAgentMixin, the key to process is not the entity, but the actual Tuple key value for the entity. If the keys are unknown, then you can ask all partitions to find Person objects of a certain shape and return their ages doubled. An example follows:

```
Session s = grid.getSession();
  ObjectMap map = s.getMap("Person");
  AgentManager amgr = map.getAgentManager();

  DoublePersonAgeAgent agent = new DoublePersonAgeAgent();
  agent.lowAge = 20;
  agent.highAge = 9999;

  Map m = amgr.callMapAgent(agent);
```

The previous example shows the AgentManager being obtained for the Person Map, and the agent constructed and initialized with the low and high ages for Persons of interest. The agent is then invoked using the callMapAgent method. Notice, no keys are supplied. As a result, the ObjectGrid invokes the agent on every partition in the grid in parallel and returns the merged results to the client. This set of returns contains all Person objects in the grid with an age between low and high and calculates the age of those Person objects doubled. This example shows how the grid APIs can be used to run a query to find entities that match a certain query. The agent is serialized and transported by the ObjectGrid to the partitions with the needed entries. The results are similarly serialized for transport back to the client. Care needs to be taken with the Map APIs. If the ObjectGrid was hosting terabytes of objects and running on many servers, then potentially this processing would overwhelm client machines. Use Map APIs to process a small subset. If a large subset needs processing, use a reduce agent to do the processing out in the data grid rather than on a client.

Parallel Reduction or aggregation agents

This style of programming processes a subset of the entries and calculates a single result for the group of entries. Examples of such a result would be:

- Minimum value
- Maximum value
- Some other business-specific function

A reduce agent is coded and invoked in a similar manner to the Map agents.

ReduceGridAgent call flow

When the AgentManager.callReduceAgent method is invoked with a collection of keys, the ReduceGridAgent instance is serialized and sent to each primary partition that the keys resolve to. This means that any instance data stored in the agent can be sent to the server. Each primary partition therefore has one instance of the agent. The reduce(Session s, ObjectMap map, Collection keys) method is invoked once per instance (partition) with the subset of keys that resolves to the partition. The result of each reduce method is then serialized back to the client. The reduceResults method is invoked on the client ReduceGridAgent instance with the collection of each result from each remote reduce invocation. The result from the reduceResults method is returned to the caller of the callReduceAgent method.

When the AgentManager.callReduceAgent method is invoked without a collection of keys, the ReduceGridAgentinstance is serialized and sent to each primary partition. This means that any instance data stored in the agent can be sent to the server. Each primary partition therefore has one instance of the agent. The reduce(Session s, ObjectMap map) method is invoked once per instance (partition). The result of each reduce method is then serialized back to the client. The reduceResults method is invoked on the client ReduceGridAgent instance with the collection of each result from each remote reduce invocation. The result from the reduceResults method is returned to the caller of the callReduceAgent method. Here is an example of a reduce agent that simply adds the ages of the matching entries.

```
package com.ibm.ws.objectgrid.test.agent.jdk5;

import java.util.Collection;
import java.util.Iterator;

import com.ibm.websphere.objectgrid.ObjectMap;
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.datagrid.EntryErrorValue;
import com.ibm.websphere.objectgrid.datagrid.ReduceGridAgent;
import com.ibm.websphere.objectgrid.query.ObjectQuery;
import com.ibm.websphere.samples.objectgrid.entityxmlgen.PersonFeature1Entity.PersonKey;

public class SumAgeReduceAgent implements ReduceGridAgent {
        private static final long serialVersionUID = 2521080771723284899L;

        /**
         * Invoked on the server if a collection of keys is passed to
         * AgentManager.callReduceAgent(). This is invoked on each primary shard
         * where the key applies.
         */
        public Object reduce(Session s, ObjectMap map, Collection keyList) {
                try {
                        int sum = 0;
                        Iterator<PersonKey> iter = keyList.iterator();
                        while (iter.hasNext()) {
                                Object nextKey = iter.next();
                                PersonKey pk = (PersonKey) nextKey;
                                Person p = (Person) map.get(pk);
                                sum += p.age;
                        }

                        return sum;
                } catch (Exception e) {
                        throw new RuntimeException(e.getMessage(), e);
```

```
                }
        }

        /**
         * Invoked on the server if a collection of keys is NOT passed to
         * AgentManager.callReduceAgent(). This is invoked on every primary shard.
         */
        public Object reduce(Session s, ObjectMap map) {
                ObjectQuery q = s
                                .createObjectQuery("select p from Person p where p.age > -1");
                Iterator<Person> iter = q.getResultIterator();
                int sum = 0;
                while (iter.hasNext()) {
                        Object nextKey = iter.next();
                        Person p = (Person) nextKey;
                        sum += p.age;
                }
                return sum;
        }

        /**
         * Invoked on the client to reduce the results from all partitions.
         */
        public Object reduceResults(Collection results) {
                // If we encounter an EntryErrorValue, then throw a RuntimeException
                // to indicate that there was at least one failure and include each
                // EntryErrorValue
                // as part of the thrown exception.
                Iterator<Integer> iter = results.iterator();
                int sum = 0;
                while (iter.hasNext()) {
                        Object nextResult = iter.next();
                        if (nextResult instanceof EntryErrorValue) {
                                EntryErrorValue eev = (EntryErrorValue) nextResult;
                                throw new RuntimeException(
                                                "Error encountered on one of the partitions: "
                                                                + nextResult, eev.getException());
                        }

                        sum += ((Integer) nextResult).intValue();
                }
                return new Integer(sum);
        }
}
```

The previous example shows the agent. The agent has three important parts. The first allows a specific set of entries to be processed without a query. It iterates over the set of entries, adding the ages. The sum is returned from the method. The second uses a query to select the entries to be aggregated. It then sums all the matching Person ages. The third method is used to aggregate the results from each partition to a single result. The ObjectGrid performs the entry aggregation in parallel across the grid. Each partition produces an intermediate result that must be aggregated with other partition intermediate results. This third method performs that task. In the following example the agent is invoked, and the ages of all Persons with ages 10 - 20 exclusively are aggregated:

```
Session s = grid.getSession();
ObjectMap map = s.getMap("Person");
AgentManager amgr = map.getAgentManager();

SumAgeReduceAgent agent = new SumAgeReduceAgent();

Person p = new Person();
p.ssn = "1";
ArrayList<Person> list = new ArrayList<Person>();
list.add(p);
p = new Person ();
p.ssn = "2";
list.add(p);
Integer v = (Integer)amgr.callReduceAgent(agent, list);
// Close the session (optional in Version 7.1.1 and later) for improved performance
s.close();
```

Agent functions

The agent is free to do ObjectMap or EntityManager operations within the local shard where it is running. The agent receives a Session and can add, update, query, read, or remove data from the partition the Session represents. Some applications query only data from the grid, but you can also write an agent to increment all the Person ages by 1 that match a certain query. There is a transaction on the Session when the agent is called, and is committed when the agent returns unless an exception is thrown

Error handling

If a map agent is invoked with an unknown key then the value that is returned is an error object that implements the EntryErrorValue interface.

Transactions

A map agent runs in a separate transaction from the client. Agent invocations may be grouped into a single transaction. If an agent fails and throws an exception, the transaction is rolled back. Any agents that ran successfully in a transaction rolls back with the failed agent. The AgentManager reruns the rolled-back agents that ran successfully in a new transaction.

Related information:
DataGrid API

Java client overrides

You can configure a WebSphere® eXtreme Scale client based on your requirements by overriding the server settings. You can override several plug-ins and attributes.

To override settings on a client, you can use either XML or programmatic configuration. For more information about overriding client settings, see Configuring Java clients with an XML configuration and Configuring Java clients programmatically.
You can override the following plug-ins on a client:

- **BackingMap plug-ins**
  - Evictor plug-in
  - MapEventListener plug-in
  - BackingMapLifecycleListener plug-in
  - MapSerializerPlugin plug-in
- **BackingMap attributes**
  - numberOfBuckets attribute
  - timeToLive attribute
  - ttlEvictorType attribute
  - evictionTriggers attribute
  - readOnly
  - nearCacheCopyMode
  - lockStrategy
- **ObjectGrid plug-ins**
  - TransactionCallback plug-in
  - ObjectGridEventListener plug-in
  - ObjectGridLifecycleListener plug-in
- **ObjectGrid attributes**
  - entityMetadataXMLFile attribute
  - txTimeout attribute
  - txIsolation attribute

Related tasks:
Configuring Java clients with an XML configuration
Configuring clients in the Spring framework
Related reference:
ObjectGrid descriptor XML file
Client properties file

Enabling client-side map replication

You can also enable replication of maps on the client side to make data available faster.

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.

Accessing data with the REST data service

Develop applications that perform operations using REST data service protocols.

Related concepts:
Operations with the REST data service
REST data services overview
Related reference:
Optimistic concurrency in the REST data service
Request protocols for the REST data service
Retrieve requests with the REST data service
Retrieving non-entities with REST data services
Insert requests with REST data services
Update requests with REST data services
Delete requests with REST data services

Operations with the REST data service

After you start the eXtreme Scale REST data service, you can use any HTTP client to interact with it. A Web browser, PHP client, Java™ client or WCF Data Services client can be used to issue any of the supported request operations.

The REST service implements a subset of the Microsoft Atom Publishing Protocol: Data Services URI and Payload Extensions specification, Version 1.0 which is part of OData protocol. This topic describes which of the features of the specification are supported and how they are mapped to eXtreme Scale.

Service root URI

Microsoft WCF Data Services typically defines a service per data source or entity model. The eXtreme Scale REST data service defines a service per defined ObjectGrid. Each ObjectGrid that is defined in the eXtreme Scale ObjectGrid client override XML file is automatically exposed as a separate REST service root.

The URI for the service root is:

`http://`*`host`*`:`*`port`*`/`*`contextroot`*`/restservice/`*`gridname`*

Where:

- *contextroot* is defined when you deploy the REST data service application, and depends on the application server
- *gridname* is the name of the ObjectGrid

Request types

The following list describes the Microsoft WCF Data Services request types which the eXtreme Scale REST data service supports. For details about each request type that WCF Data Services supports, see: MSDN: Request Types.

Insert request types
: Clients can insert resources using the POST HTTP verb with the following limitations:

  - InsertEntity Request: Supported.
  - InsertLink request: Supported.
  - InsertMediaResource request: Not supported due to media resource support restriction.

  For additional information, see: MSDN: Insert Request Types.

Update request types
: Clients can update resources using the PUT and MERGE HTTP verbs with the following limitations:

  Note: 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.

  - UpdateEntity Request: Supported.
  - UpdateComplexType Request: Not Supported due to complex type restriction.
  - UpdatePrimitiveProperty Request: Supported.
  - UpdateValue Request: Supported.
  - UpdateLink Request: Supported.
  - UpdateMediaResource Request: Not supported due to media resource support restriction.

  For additional information, see: MSDN: Insert Request types.

Delete request types
: Clients can delete resources using the DELETE HTTP verb with the following limitations:

  - DeleteEntity Request: Supported.
  - DeleteLink Request: Supported.
  - DeleteValue request: Supported.

  For additional information, see: MSDN: Delete Request Types.

Retrieve request types
: Clients can retrieve resources using the GET HTTP verb with the following limitations:

  - RetrieveEntitySet Request: Supported.
  - RetrieveEntity Request: Supported.
  - RetrieveComplexType Request: Not supported due to complex type restriction.
  - RetrievePrimitiveProperty Request: Supported.
  - RetrieveValue Request: Supported.
  - RetrieveServiceMetadata Request: Supported.
  - RetrieveServiceDocument Request: Supported.
  - RetrieveLink Request: Supported.
  - Retrieve Request Containing a Customizable Feed Mapping: Not supported
  - RetrieveMediaResource: Not supported due to media resource restriction.

  For additional information, see: MSDN: Retrieve Request Types.

System query options
: Queries are supported which allow clients to identify a collection of entities or a single entity. System query options are specified in a data service URI and are supported with the following limitations:

- $expand: Supported
- $filter: Supported.
- $orderby: Supported.
- $format: Not supported. The acceptable format is identified in the HTTP Accept request header.
- $skip: Supported
- $top: Supported

For additional information, see: MSDN: System Query Options.

Partition routing

Partition routing is based on the root entity. A request URI infers a root entity if its resource path starts with a root entity or with an entity that has a direct or indirect association to the entity. In a partitioned environment, any request that cannot infer a root entity will be rejected. Any request that infers a root entity will be routed to the correct partition.

For additional information on defining a schema with associations and root entities, see Scalable data model in eXtreme Scale and Partitioning.

Invoke request

Invoke requests are not supported. For additional information, see MSDN: Invoke Request.

Batch request

Clients can batch multiple Change Sets or Query Operations within a single request. This can reduce the number of round trips to the server and allows multiple requests to participate in a single transaction. For additional information, see MSDN: Batch Request.

Tunneled requests

Tunneled requests are not supported. For additional information, see MSDN: Tunneled Requests.

Related concepts:
REST data services overview
Related tasks:
Accessing data with the REST data service
Related reference:
Optimistic concurrency in the REST data service
Request protocols for the REST data service
Retrieve requests with the REST data service
Retrieving non-entities with REST data services
Insert requests with REST data services
Update requests with REST data services
Delete requests with REST data services

Optimistic concurrency in the REST data service

The eXtreme Scale REST data service uses an optimistic locking model by using native HTTP headers: If-Match, If-None-Match, and ETag. These headers are sent in request and response messages to relay an entity's version information from the server to client and client to server.

For more details on optimistic concurrency, refer to MSDN Library: Optimistic Concurrency (ADO.NET).

The eXtreme Scale REST data service enables optimistic concurrency for an entity if a version attribute is defined in the entity schema for that entity. A version property can be defined in the entity schema by a @Version annotation for Java™ classes or a <version/> attribute for entities defined using an entity descriptor XML file. The eXtreme Scale REST data service automatically propagates the value of the version property to the client in the ETag header for single entity responses using an m:etag attribute in the payload for multiple entity XML responses, and an etag attribute in the payload for multiple entity JSON responses.

For more details on defining an eXtreme Scale entity schema, see Defining an entity schema.

Related concepts:
Operations with the REST data service
REST data services overview
Related tasks:
Accessing data with the REST data service

Request protocols for the REST data service

In general, the protocols for interacting with the REST service are the same as those described in the WCF Data Services AtomPub protocol. However, eXtreme Scale does provide additional details, from eXtreme Scale Entity Model perspective. Users are expected to be familiar with the WCF Data Services protocols before reading this section. Alternatively, users can read this section with the WCF Data Services protocol section.

Examples are provided to illustrate the request and response. These examples apply to both the eXtreme Scale REST data service and WCF Data Services. Because Web browsers can only retrieve data, the CUD (create, update and delete) operations must be performed by another client such as Java™, JavaScript, RUBY or PHP.

Related concepts:
Operations with the REST data service
REST data services overview
Related tasks:
Accessing data with the REST data service

Retrieve requests with the REST data service

A RetrieveEntity Request is used by a client to retrieve an eXtreme Scale entity. The response payload contains the entity data in AtomPub or JSON format. Also, the system operator $expand can be used to expand the relations. The relations are represented in line within the data service response as an Atom Feed Document, which is a to-many relation, or an Atom Entry Document which is a to-one relation.

Tip: For more details on the RetrieveEntity protocol defined in WCF Data Services, refer to MSDN: RetrieveEntity Request.

Retrieving an entity

The following RetrieveEntity example retrieves a Customer entity with key.

AtomPub

- Method
  GET
- Request URI:
  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('ACME')
- Request Header:
  Accept: application/atom+xml
- Request Payload:
  None
- Response Header:
  Content-Type: application/atom+xml
- Response Payload:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<entry xml:base = "http://localhost:8080/wxsrestservice/
restservice" xmlns:d= "http://schemas.microsoft.com/ado/2007/
08/dataservices" xmlns:m = "http://schemas.microsoft.com/ado/2007/
08/dataservices/metadata" xmlns = "http://www.w3.org/2005/Atom">

<category term = "NorthwindGridModel.Customer" scheme = "http://
schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>
<id>http://localhost:8080/wxsrestservice/restservice/
                        NorthwindGrid/Customer('ACME')</id>
    <title type = "text"/>
    <updated>2009-12-16T19:52:10.593Z</updated>
    <author>
        <name/>
    </author>
    <link rel = "edit" title = "Customer" href = "Customer(
                                        'ACME')"/>
    <link rel = "http://schemas.microsoft.com/ado/2007/08/
                                       dataservices/related/
orders" type = "application/atom+xml;type=feed" title =
                "orders" href ="Customer('ACME')/orders"/>
    <content type = "application/xml">
        <m:properties>
            <d:customerId>ACME</d:customerId>
            <d:city m:null = "true"/>
            <d:companyName>RoaderRunner</d:companyName>
            <d:contactName>ACME</d:contactName>
            <d:country m:null = "true"/>
            <d:version m:type = "Edm.Int32">3</d:version>
        </m:properties>
    </content>
</entry>
```

- Response Code:
  200 OK

JSON

- Method
  GET
- Request URI:
  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('ACME')

- Request Header:
  Accept: application/json
- Request Payload:
  None
- Response Header:
  Content-Type: application/json
- Response Payload:

```
{"d":{"__metadata":{"uri":"http://localhost:8080/wxsrestservice/
                        restservice/NorthwindGrid/Customer('ACME')",
                        "type":"NorthwindGridModel.Customer"},
                        "customerId":"ACME",
                        "city":null,
                        "companyName":"RoaderRunner",
                        "contactName":"ACME",
                        "country":null,
                        "version":3,
                        "orders":{"__deferred":{"uri":"http://localhost:8080/
                                                wxsrestservice/restservice/
                        NorthwindGrid/Customer('ACME')/orders"}}}}
```

- Response Code:
  200 OK

Queries

A query can also be used with a RetrieveEntitySet or RetrieveEntity request. A query is specified by the system $filter operator.

For details on the $filter operator, refer to: MSDN: Filter System Query Option ($filter)

The OData protocol supports several common expressions. The eXtreme Scale REST data service supports a subset of the expressions defined in the specification:

- Boolean expressions:
  - eq, ne, lt, le, gt, ge
  - negate
  - not
  - parenthesis
  - and, or
- Arithmetic expressions:
  - add
  - sub
  - mul
  - div
- Primitive literals
  - String
  - date-time
  - decimal
  - single
  - double
  - int16
  - int32
  - int64
  - binary
  - null
  - byte

The following expressions are *not* available:

- Boolean expressions:
  - isof
  - cast
- Method call expressions
- Arithmetic expressions:
  - mod
- Primitive literals:
  - Guid
- Member expressions

For a complete list and description of the expressions that are available in Microsoft WCF Data Services, see section 2.2.3.6.1.1 : Common Expression Syntax.

The following example demonstrates a RetrieveEntity request with a query. In this example, all customers whose contact name is “RoadRunner” are retrieved. The only customer which matches this filter is Customer('ACME') as shown in the response payload.

Restriction: This query will only work for non-partitioned entities. If Customer is partitioned, then the key belonging to the customer is required.
AtomPub

- Method: GET
- Request URI: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer?$filter=contactName eq 'RoadRunner'
- Request Header: Accept: application/atom+xml
- Input Payload: None
- Response Header: Content-Type: application/atom+xml
- Response Payload:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<feed
        xml:base="http://localhost:8080/wxsrestservice/restservice"
        xmlns:d="http://schemas.microsoft.com/ado/2007/08/
                        dataservices"
        xmlns:m="http://schemas.microsoft.com/ado/2007/08/
                        dataservices/metadata"
        xmlns="http://www.w3.org/2005/Atom">
        <title type="text">Customer</title>
        <id>            http://localhost:8080/wxsrestservice/restservice/
                        NorthwindGrid/Customer  </id>
        <updated>2009-09-16T04:59:28.656Z</updated>
        <link rel="self" title="Customer"
href="#_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_Customer" />
        <entry>
                <category term="NorthwindGridModel.Customer"
                        scheme="http://schemas.microsoft.com/ado/2007/08/
                        dataservices/scheme" />
                <id>
http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
                Customer('ACME')</id>
                <title type="text" />
                <updated>2009-09-16T04:59:28.656Z</updated>
                <author>
                        <name />
                </author>
                <link rel="edit" title="Customer"
href="#_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_Customer('ACME')" />
                <link
                        rel="http://schemas.microsoft.com/ado/2007/08/dataservices/
                                                related/orders"
                        type="application/atom+xml;type=feed" title="orders"
href="#_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_Customer('ACME')_orders" />
                <content type="application/xml">
                          <m:properties>
            <d:customerId>ACME</d:customerId>
            <d:city m:null = "true"/>
            <d:companyName>RoaderRunner</d:companyName>
            <d:contactName>ACME</d:contactName>
            <d:country m:null = "true"/>
            <d:version m:type = "Edm.Int32">3</d:version>
        </m:properties>
                </content>
        </entry>
</feed>
```

- Response Code: 200 OK

JSON

- Method: GET
- Request URI:
  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer?$filter=contactName eq 'RoadRunner'

- Request Header: Accept: application/json
- Request Payload: None
- Response Header: Content-Type: application/json
- Response Payload:

```
{"d":[{"__metadata":{"uri":"http://localhost:8080/wxsrestservice/
                        restservice/NorthwindGrid/Customer('ACME')",
                        "type":"NorthwindGridModel.Customer"},
                        "customerId":"ACME",
                        "city":null,
                        "companyName":"RoaderRunner",
                        "contactName":"ACME",
                        "country":null,
                        "version":3,
                        "orders":{"__deferred":{"uri":"http://localhost:8080/
                                        wxsrestservice/restservice/NorthwindGrid/
                                        Customer('ACME')/orders"}}}]}
```

- Response Code: 200 OK

System operator $expand

The system operator $expand can be used to expand associations. The associations are represented in line in the data service response. Multi-valued (to-many) associations are represented as an Atom Feed Document or JSON array. Single-valued (to-one) associations, are represented as n Atom Entry Document or

JSON object.

For more details on the $expand system operator, refer to Expand System Query Option ($expand).

Here is an example of using the $expand system operator. In this example, we retrieve the entity Customer('IBM')which has an Orders 5000, 5001 and others associated with it. The $expand clause is set to “orders”, so the order collection is expand as inline in the response payload. Only orders 5000 and 5001 are displayed here.

AtomPub

- Method: GET
- Request URI: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')?$expand=orders
- Request Header: Accept: application/atom+xml
- Request Payload: None
- Response Header: Content-Type: application/atom+xml
- Response Payload:

```
<?xml version="1.0" encoding="utf-8"?>
<entry xml:base = "http://localhost:8080/wxsrestservice/restservice"
                        xmlns:d ="http://schemas.microsoft.com/ado/2007/08/dataservices"
                        xmlns:m = "http://schemas.microsoft.com/ado/2007/08/dataservices/
                        metadata" xmlns = "http://www.w3.org/2005/Atom">
<category term = "NorthwindGridModel.Customer" scheme = "http://schemas.

microsoft.com/ado/2007/08/dataservices/scheme"/>
    <id>http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
                                Customer('IBM')</id>
    <title type = "text"/>
    <updated>2009-12-16T22:50:18.156Z</updated>
    <author>
        <name/>
    </author><link rel = "edit" title = "Customer" href =
                                "Customer('IBM')"/>
    <link rel = "http://schemas.microsoft.com/ado/2007/08/dataservices/
                                related/orders" type = "application/atom+xml;type=feed" title =
                                "orders" href = "Customer('IBM')/orders">
        <m:inline>
            <feed>
                <title type = "text">orders</title>
                <id>http://localhost:8080/wxsrestservice/restservice/
                                                                        NorthwindGrid/Customer('IBM')/orders</id>
                <updated>2009-12-16T22:50:18.156Z</updated>
                <link rel = "self" title = "orders" href = "Customer
                                                                                ('IBM')/orders"/>
                <entry>
                    <category term = "NorthwindGridModel.Order" scheme =

"http://schemas.microsoft.com/ado/2007/08/
                                                                                dataservices/scheme"/>
                    <id>http://localhost:8080/wxsrestservice/restservice/

NorthwindGrid/Order(orderId=5000,customer_customerId=
                                                                                'IBM')</id>
                    <title type = "text"/>
                    <updated>2009-12-16T22:50:18.156Z</updated>
                    <author>
                        <name/>
                    </author>
                    <link rel = "edit" title = "Order" href =

"Order(orderId=5000,customer_customerId='IBM')"/>
                    <link rel = "http://schemas.microsoft.com/ado/2007/08/
                                                                        dataservices/related/customer" type =
"application/
                                                                        atom+xml;type=entry" title ="customer"
href =

"Order(orderId=5000,customer_customerId='IBM')/customer"/>
                    <link rel = "http://schemas.microsoft.com/ado/2007/08/
                                                                        dataservices/related/orderDetails" type =
"application/
                                                                                atom+xml;type=feed" title
="orderDetails" href =

"Order(orderId=5000,customer_customerId='IBM')/orderDetails"/>
                    <content type = "application/xml">
                        <m:properties>
                            <d:orderId m:type = "Edm.Int32">5000</d:orderId>
                            <d:customer_customerId>IBM</d:customer_customerId>
                            <d:orderDate m:type = "Edm.DateTime">

2009-12-16T19:46:29.562</d:orderDate>
                            <d:shipCity>Rochester</d:shipCity>
                            <d:shipCountry m:null = "true"/>
                            <d:version m:type = "Edm.Int32">0</d:version>
                        </m:properties>
                    </content>
                </entry>
                <entry>
```

```
                    <category term = "NorthwindGridModel.Order" scheme =
"http://schemas.microsoft.com/ado/2007/08/
                                                                    dataservices/scheme"/>
                    <id>http://localhost:8080/wxsrestservice/restservice/
NorthwindGrid/Order(orderId=5001,customer_customerId=
                                                                    'IBM')</id>
                    <title type = "text"/>
                    <updated>2009-12-16T22:50:18.156Z</updated>
                    <author>
                        <name/></author>
                    <link rel = "edit" title = "Order" href = "Order(
orderId=5001,customer_customerId='IBM')"/>
                    <link rel = "http://schemas.microsoft.com/ado/2007/
08/dataservices/related/customer" type =
"application/atom+xml;type=entry" title =
                                                                    "customer" href =
"Order(orderId=5001,customer_customerId=
                                                                    'IBM')/customer"/>
                    <link rel = "http://schemas.microsoft.com/ado/2007/08/
dataservices/related/orderDetails" type =
"application/atom+xml;type=feed" title =
                                                                "orderDetails" href
= "Order(orderId=5001,
customer_customerId='IBM')/orderDetails"/>
                    <content type = "application/xml">
                        <m:properties>
                            <d:orderId m:type = "Edm.Int32">5001</d:orderId>
                            <d:customer_customerId>IBM</d:customer_customerId>
                            <d:orderDate m:type = "Edm.DateTime">2009-12-16T19:
50:11.125</d:orderDate>
                            <d:shipCity>Rochester</d:shipCity>
                            <d:shipCountry m:null = "true"/>
                            <d:version m:type = "Edm.Int32">0</d:version>
                        </m:properties>
                    </content>
                </entry>
            </feed>
        </m:inline>
    </link>
    <content type = "application/xml">
        <m:properties>
            <d:customerId>IBM</d:customerId>
            <d:city m:null = "true"/>
            <d:companyName>IBM Corporation</d:companyName>
            <d:contactName>John Doe</d:contactName>
            <d:country m:null = "true"/>
            <d:version m:type = "Edm.Int32">4</d:version>
        </m:properties>
    </content>
</entry>
```

- Response Code: 200 OK

JSON

- Method: GET
- Request URI: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')?$expand=orders
- Request Header: Accept: application/json
- Request Payload: None
- Response Header: Content-Type: application/json
- Response Payload:

```
{"d":{"__metadata":{"uri":"http://localhost:8080/wxsrestservice/
                        restservice/NorthwindGrid/Customer('IBM')",
       "type":"NorthwindGridModel.Customer"},
"customerId":"IBM",
"city":null,
"companyName":"IBM Corporation",
"contactName":"John Doe",
"country":null,
"version":4,
"orders":[{"__metadata":{"uri":"http://localhost:8080/
                             wxsrestservice/restservice/NorthwindGrid/Order(
                             orderId=5000,customer_customerId='IBM')",
"type":"NorthwindGridModel.Order"},
"orderId":5000,
"customer_customerId":"IBM",
"orderDate":"\/Date(1260992789562)\/",
"shipCity":"Rochester",
"shipCountry":null,
"version":0,
```

```
"customer":{"__deferred":{"uri":"http://localhost:8080/
                        wxsrestservice/restservice/NorthwindGrid/Order(
                        orderId=5000,customer_customerId='IBM')/customer"}},
"orderDetails":{"__deferred":{"uri":"http://localhost:
                                        8080/wxsrestservice/restservice/NorthwindGrid/
                                        Order(orderId=5000,customer_customerId='IBM')/
                                        orderDetails"}}},
{"__metadata":{"uri":"http://localhost:8080/wxsrestservice/
                                        restservice/NorthwindGrid/Order(orderId=5001,
                                        customer_customerId='IBM')","type":
                                        "NorthwindGridModel.Order"},
"orderId":5001,
"customer_customerId":"IBM",
"orderDate":"\/Date(1260993011125)\/",
"shipCity":"Rochester",
"shipCountry":null,
"version":0,
"customer":{"__deferred":{"uri":"http://localhost:
                        8080/wxsrestservice/restservice/
NorthwindGrid/Order(orderId=5001,customer_customerId
                        ='IBM')/customer"}},
"orderDetails":{"__deferred":{"uri":"http://localhost:8080/
                        wxsrestservice/restservice/NorthwindGrid/Order(
                        orderId=5001,   customer_customerId='IBM')/
                        orderDetails"}}}]}}
```

- Response Code: 200 OK

Related concepts:
Operations with the REST data service
REST data services overview
Related tasks:
Accessing data with the REST data service

Retrieving non-entities with REST data services

The REST data service allows you to retrieve more than only entities, such as entity collections and properties.

Retrieve an entity collection

A RetrieveEntitySet Request can be used by a client to retrieve a set of eXtreme Scale entities. The entities are represented as an Atom Feed Document or JSON array in the response payload. For more details on the RetrieveEntitySet protocol defined in WCF Data Services, refer to MSDN: RetrieveEntitySet Request.

The following RetrieveEntitySet request example retrieves all the Order entities associated with the Customer('IBM') entity. Only orders 5000 and 5001 are displayed here.

AtomPub

- Method: GET
- Request URI: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/orders
- Request Header: Accept: application/atom+xml
- Request Payload: None
- Response Header: Content-Type: application/atom+xml
- Response Payload:

```
<?xml version="1.0" encoding="utf-8"?>
<feed xml:base = "http://localhost:8080/wxsrestservice/restservice"
                        xmlns:d = "http://schemas.microsoft.com/ado/2007/08/dataservices"
                        xmlns:m = "http://schemas.microsoft.com/ado/2007/08/dataservices/
                        metadata" xmlns =       "http://www.w3.org/2005/Atom">
    <title type = "text">Order</title>
    <id>http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
                                        Order</id>
    <updated>2009-12-16T22:53:09.062Z</updated>
    <link rel = "self" title = "Order" href = "Order"/>
    <entry>
        <category term = "NorthwindGridModel.Order" scheme = "http://
                                            schemas.microsoft.com/
                                            ado/2007/08/dataservices/scheme"/>
        <id>http://localhost:8080/wxsrestservice/restservice/
                                            NorthwindGrid/Order(orderId=5000,customer_customerId=
                                            'IBM')</id>
        <title type = "text"/>
        <updated>2009-12-16T22:53:09.062Z</updated>
        <author>
            <name/>
        </author>
        <link rel = "edit" title = "Order" href = "Order(orderId=5000,
                                            customer_customerId='IBM')"/>
        <link rel = "http://schemas.microsoft.com/ado/2007/08/
                                            dataservices/related/customer"
                                            type =  "application/atom+xml;type=entry"
```

```
                                            title = "customer" href = "Order(orderId=5000,
                                            customer_customerId='IBM')/customer"/>
        <link rel = "http://schemas.microsoft.com/ado/2007/08/
                                            dataservices/related/orderDetails"
                                            type = "application/atom+xml;type=feed"
                                            title = "orderDetails" href = "Order(orderId=5000,
                                                                    customer_customerId='IBM')/
                                                                orderDetails"/>
        <content type = "application/xml">
            <m:properties>
                <d:orderId m:type = "Edm.Int32">5000</d:orderId>
                <d:customer_customerId>IBM</d:customer_customerId>
                <d:orderDate m:type = "Edm.DateTime">2009-12-16T19:
                                                                        46:29.562</d:orderDate>
                <d:shipCity>Rochester</d:shipCity>
                <d:shipCountry m:null = "true"/>
                <d:version m:type = "Edm.Int32">0</d:version>
            </m:properties>
        </content>
    </entry>
    <entry>
        <category term = "NorthwindGridModel.Order" scheme = "http://
                                            schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>
        <id>http://localhost:8080/wxsrestservice/restservice/
                                            NorthwindGrid/Order(orderId=5001,       customer_customerId='IBM')
                                </id>
        <title type = "text"/>
        <updated>2009-12-16T22:53:09.062Z</updated>
        <author>
            <name/>
        </author>
        <link rel = "edit" title = "Order" href = "Order(orderId=5001,
                                                    customer_customerId='IBM')"/>
        <link rel = "http://schemas.microsoft.com/ado/2007/08/
                                                    dataservices/related/customer"
                                                    type =  "application/atom+xml;type=entry"
                                                    title = "customer" href = "Order(orderId=5001,
                                                                customer_customerId='IBM')/customer"/>
        <link rel = "http://schemas.microsoft.com/ado/2007/08/
                                                                dataservices/related/orderDetails"
                                                    type = "application/atom+xml;type=feed"
                                                    title = "orderDetails" href = "Order(orderId=5001,
                                                                customer_customerId='IBM')/orderDetails"/>
        <content type = "application/xml">
            <m:properties>
                <d:orderId m:type = "Edm.Int32">5001</d:orderId>
                <d:customer_customerId>IBM</d:customer_customerId>
                <d:orderDate m:type = "Edm.DateTime">2009-12-16T19:50:
                                                                        11.125</d:orderDate>
                <d:shipCity>Rochester</d:shipCity>
                <d:shipCountry m:null = "true"/>
                <d:version m:type = "Edm.Int32">0</d:version>
            </m:properties>
        </content>
    </entry>
</feed>
```

- Response Code: 200 OK

JSON

- Method: GET
- Request URI: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/orders
- Request Header: Accept: application/json
- Request Payload: None
- Response Header: Content-Type: application/json
- Response Payload:

```
{"d":[{"__metadata":{"uri":"http://localhost:8080/wxsrestservice/
                        restservice/NorthwindGrid/Order(orderId=5000,
                        customer_customerId='IBM')",
"type":"NorthwindGridModel.Order"},
"orderId":5000,
"customer_customerId":"IBM",
"orderDate":"\/Date(1260992789562)\/",
"shipCity":"Rochester",
"shipCountry":null,
"version":0,
"customer":{"__deferred":{"uri":"http://localhost:8080/
                        wxsrestservice/restservice/NorthwindGrid/Order(orderId=
                        5000,customer_customerId='IBM')/customer"}},
"orderDetails":{"__deferred":{"uri":"http://localhost:8080/
                        wxsrestservice/restservice/NorthwindGrid/Order(orderId=
                        5000,customer_customerId='IBM')/orderDetails"}}},
{"__metadata":{"uri":"http://localhost:8080/wxsrestservice/
                                    restservice/NorthwindGrid/
                        Order(orderId=5001,
                        customer_customerId='IBM')",
"type":"NorthwindGridModel.Order"},
```

```
"orderId":5001,
"customer_customerId":"IBM",
"orderDate":"\/Date(1260993011125)\/",
"shipCity":"Rochester",
"shipCountry":null,
"version":0,
"customer":{"__deferred":{"uri":"http://localhost:8080/
                          wxsrestservice/restservice/NorthwindGrid/Order(orderId=
                          5001,customer_customerId='IBM')/customer"}},
"orderDetails":{"__deferred":{"uri":"http://localhost:8080/
                          wxsrestservice/restservice/NorthwindGrid/Order(orderId=
                          5001,customer_customerId='IBM')/orderDetails"}}}]}
```

- Response Code: 200 OK

Retrieve a property

A RetrievePrimitiveProperty request can be used to get the value of a property of an eXtreme Scale entity instance. The property value is represented as XML format for AtomPub requests and a JSON object for JSON requests in the response payload. For more details on RetrievePrimitiveProperty request, refer to MSDN: RetrievePrimitiveProperty Request.

The following RetrievePrimitiveProperty request example retrieves the contactName property of the Customer('IBM') entity.

AtomPub

- Method: GET
- Request URI: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/contactName
- Request Header: Accept: application/xml
- Request Payload: None
- Response Header: Content-Type: application/atom+xml
- Response Payload:

```
<contactName xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices">
    John Doe
</contactName>
```

- Response Code: 200 OK

JSON

- Method: GET
- Request URI: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/contactName
- Request Header: Accept: application/json
- Request Payload: None
- Response Header: Content-Type: application/json
- Response Payload: `{"d":{"contactName":"John Doe"}}`
- Response Code: 200 OK

Retrieve a property value

A RetrieveValue request can be used to get the raw value of a property on an eXtreme Scale entity instance. The property value is represented as a raw value in the response payload. If the entity type is one of the following, then the media type of the response is “text/plain." Otherwise the response' media type is “application/octet-stream." These types are:

- Java™ primitive types and their respective wrappers
- java.lang.String
- byte[]
- Byte[]
- char[]
- Character[]
- enums
- java.math.BigInteger
- java.math.BigDecimal
- java.util.Date
- java.util.Calendar
- java.sql.Date
- java.sql.Time
- java.sql.Timestamp

For more details on the RetrieveValue request, refer to MSDN: RetrieveValue Request.

The following RetrieveValue request example retrieves the raw value of the contactName property of the Customer('IBM') entity.

- Request Method: GET
- Request URI: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/contactName/$value
- Request Header: Accept: text/plain
- Request Payload: None
- Response Header: Content-Type: text/plain
- Response Payload: John Doe
- Response Code: 200 OK

Retrieve a link

A RetrieveLink Request can be used to get the link(s) representing a to-one association or to-many association. For the to-one association, the link is from one eXtreme Scale Entity instance to another, and the link is represented in the response payload. For the to-many association, the links are from one eXtreme Scale Entity instance to all others in a specified eXtreme Scale entity collection, and the response is represented as a set of links in the response payload. For more details on RetrieveLink request, refer to MSDN: RetrieveLink Request.

Here is a RetrieveLink request example. In this example, we retrieve the association between entity Order(orderId=5000,customer_customerId='IBM') and its customer. The response shows the Customer entity URI.

AtomPub

- Method: GET
- Request URI: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=5000,customer_customerId='IBM')/$links/customer
- Request Header: Accept: application/xml
- Request Payload: None
- Response Header: Content-Type: application/xml
- Response Payload:

```
<?xml version="1.0" encoding="utf-8"?>
<uri>http://localhost:8080/wxsrestservice/restservice/
                        NorthwindGrid/Customer('IBM')</uri>
```

- Response Code: 200 OK

JSON

- Method: GET
- Request URI: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=5000,customer_customerId='IBM')/$links/customer
- Request Header: Accept: application/json
- Request Payload: None
- Response Header: Content-Type: application/json
- Response Payload: {"d":{"uri":"http:\/\/localhost:8080\/wxsrestservice\/restservice\/NorthwindGrid\/Customer('IBM')"}}

Retrieve service metadata

A RetrieveServiceMetadata Request can be used to get the conceptual schema definition language (CSDL) document, which describes the data model associated with the eXtreme Scale REST data service. For more details on RetrieveServiceMetadata request, refer to MSDN: RetrieveServiceMetadata Request.

Retrieve service document

A RetrieveServiceDocument Request can be used to retrieve the Service Document describing the collection of resources exposed by the eXtreme Scale REST data service. For more details on RetrieveServiceDocument request, refer to MSDN: RetrieveServiceDocument Request.

Related concepts:
Operations with the REST data service
REST data services overview
Related tasks:
Accessing data with the REST data service

Insert requests with REST data services

An InsertEntity Request can be used to insert a new eXtreme Scale entity instance, potentially with new related entities, into the eXtreme Scale REST data service.

Insert entity request

An InsertEntity Request can be used to insert a new eXtreme Scale entity instance, potentially with new related entities, into the eXtreme Scale REST data service. When inserting an entity, the client may specify if the resource or entity should be automatically linked to other existing entities in the data service.

The client must include the required binding information in the representation of the associated relation in the request payload.

In addition to supporting the insertion of a new EntityType instance (E1), the InsertEntity request also allows inserting new entities related to E1 (described by an entity relation) in a single Request. For example, when inserting a Customer('IBM'), we can insert all the orders with Customer('IBM'). This form of an InsertEntity Request is also known as a *deep insert*. With a deep insert, the related entities must be represented using the inline representation of the relation associated with E1 that identifies the link to the to-be-inserted related entities.

The properties of the entity to be inserted are specified in the request payload. The properties are parsed by the REST data service and then set to the correspondent property on the entity instance. For the AtomPub format, the property is specified as a <d:PROPERTY_NAME> XML element. For JSON, the property is specified as a property of a JSON object.

If a property is missing in the request payload, then the REST data service sets the entity property value to the java default value. However, the database backend might reject such a default value, for example, if the column is not nullable in the database. Then a 500 response code will be returned to indicate an Internal Server error.

If there are duplicate properties specified in the payload, the last property will be used. All the previous values for the same property name are ignored by the REST data service.

If the payload contains a non-existent property, then the REST data service returns a 400 (Bad Request) response code to indicate the request sent by the client was syntactically incorrect.

If the key properties are missing, then the REST data service returns a response code of 400 (Bad Request) to indicate a missing key property.

If the payload contains a link to a related entity with a non-existent key, then the REST data service returns a 404 (Not Found) response code to indicate the linked entity cannot be found.

If the payload contains a link to a related entity with an incorrect association name, then the REST data service returns a 400 (Bad Request) response code to indicate the link cannot be found.

If the payload contains more than one link to a one-to-one relation, the last link will be used. All the previous links for the same association are ignored.

For more details on the InsertEntity request, see MSDN Library: InsertEntity Request.

An InsertEntity request inserts a Customer entity with key 'IBM'.

AtomPub

- Method: POST
- Request URI: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')
- Request Header: Accept: application/atom+xml Content-Type: application/atom+xml
- Request Payload:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<entry xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
 xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
 xmlns="http://www.w3.org/2005/Atom">
 <category term="NorthwindGridModel.Customer"
  scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
<content type="application/xml">
  <m:properties>
   <d:customerId>Rational</d:customerId>
   <d:city>Rochester</d:city>
   <d:companyName>Rational</d:companyName>
   <d:contactName>John Doe</d:contactName>
   <d:country>USA</d:country>
  </m:properties>
 </content>
</entry>
```

- Response Header: Content-Type: application/atom+xml
- Response Payload:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<entry xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
 xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
 xmlns="http://www.w3.org/2005/Atom">
 <category term="NorthwindGridModel.Customer"
  scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
<content type="application/xml">
  <m:properties>
   <d:customerId>Rational</d:customerId>
   <d:city>Rochester</d:city>
   <d:companyName>Rational</d:companyName>
   <d:contactName>John Doe</d:contactName>
   <d:country>USA</d:country>
  </m:properties>
 </content>
</entry>
Response Header:
Content-Type: application/atom+xml
Response Payload:
<?xml version="1.0" encoding="utf-8"?>
<entry xml:base = "http://localhost:8080/wxsrestservice/restservice" xmlns:d =
                       "http://schemas.microsoft.com/ado/2007/08/dataservices" xmlns:m =
                                    "http://schemas.microsoft.com/
                       ado/2007/08/dataservices/metadata" xmlns = "http://www.w3.org/2005/Atom">
    <category term = "NorthwindGridModel.Customer" scheme = "http://schemas.
                                    microsoft.com/ado/2007/08/dataservices/scheme"/>
    <id>http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
                                    Customer('Rational')</id>
    <title type = "text"/>
    <updated>2009-12-16T23:25:50.875Z</updated>
    <author>
        <name/>
    </author>
    <link rel = "edit" title = "Customer" href = "Customer('Rational')"/>
    <link rel = "http://schemas.microsoft.com/ado/2007/08/dataservices/related/
                                orders" type = "application/atom+xml;type=feed"
                                title = "orders" href = "Customer('Rational')/orders"/>
    <content type = "application/xml">
        <m:properties>
            <d:customerId>Rational</d:customerId>
            <d:city>Rochester</d:city>
```

```
            <d:companyName>Rational</d:companyName>
            <d:contactName>John Doe</d:contactName>
            <d:country>USA</d:country>
            <d:version m:type = "Edm.Int32">0</d:version>
        </m:properties>
    </content>
</entry>
```

- Response Code: 201 Created

JSON

- Method: POST
- Request URI: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer
- Request Header: Accept: application/json Content-Type: application/json
- Request Payload:

```
{"customerId":"Rational",
"city":null,
"companyName":"Rational",
"contactName":"John Doe",
"country": "USA",}
```

- Response Header: Content-Type: application/json
- Response Payload:

```
{"d":{"__metadata":{"uri":"http://localhost:8080/wxsrestservice/restservice/
                          NorthwindGrid/Customer('Rational')",
"type":"NorthwindGridModel.Customer"},
"customerId":"Rational",
"city":null,
"companyName":"Rational",
"contactName":"John Doe",
"country":"USA",
"version":0,
"orders":{"__deferred":{"uri":"http://localhost:8080/wxsrestservice/restservice/
                          NorthwindGrid/Customer('Rational')/orders"}}}}
```

- Response Code: 201 Created

Insert link request

An InsertLink Request can be used to create a new Link between two eXtreme Scale entity instances. The URI of the request must resolve to an eXtreme Scale one-to-many association. The payload of the request contains a single link which points to the one-to-many association target entity.

If the URI of the InsertLink request represents a to-one association, the REST data service returns a 400 (Bad request) response.

If the URI of the InsertLink request points to an association which does not exist, the REST data service returns a 404 (Not Found) response to indicate the link cannot be found.

If the payload contains a link with a key which does not exist, the REST data service returns a 404 (Not Found) response to indicate the linked entity cannot be found.

If the payload contains more than one link, the eXtreme Scale Rest Data Service will parse the first link. The remaining links are ignored.

For more details on InsertLink request, refer to: MSDN Library: InsertLink Request.

The following InsertLink request example creates a link from Customer('IBM') to Order(orderId=5000,customer_customerId='IBM').

AtomPub

- Method: POST
- Request URI: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/$link/orders
- Request Header: Content-Type: application/xml
- Request Payload:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<uri>http://host:1000/wxsrestservice/restservice/NorthwindGrid/Order(orderId=
                          5000,customer_customerId='IBM')</uri>
```

- Response Payload: None
- Response Code: 204 No Content

JSON

- Method: POST
- Request URI: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/$links/orders
- Request Header: Content-Type: application/json
- Request Payload:

```
{"uri": "http://host:1000/wxsrestservice/restservice/NorthwindGrid/Order(orderId
                                =5000,customer_customerId='IBM')"}
```

- Response Payload: None
- Response Code: 204 No Content

Related concepts:
Operations with the REST data service
REST data services overview
Related tasks:
Accessing data with the REST data service

Update requests with REST data services

The WebSphere® eXtreme Scale REST data service supports update requests for entities, entity primitive properties, and so on.

Update an entity

An UpdateEntity Request can be used to update an existing eXtreme Scale entity. The client can use an HTTP PUT method to replace an existing eXtreme Scale entity, or use an HTTP MERGE method to merge the changes into an existing eXtreme Scale entity.
Note: 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.
When updating the entity, the client can specify if the entity, in addition to being updated, must be automatically linked to other existing entities in the data service that are related through single valued to-one associations.

The property of the entity to be updated is in the request payload. The property is parsed by the REST data service and then set to the correspondent property on the entity. For the AtomPub format, the property is specified as a `<d:PROPERTY_NAME>` XML element. For JSON, the property is specified as a property of a JSON object.

If a property is missing in the request payload, the REST data service sets the entity property value to the Java default value for HTTP PUT method. However, the database backend might reject such a default value if, for example, the column is not nullable in the database. Then a 500 (Internal Server Error) response code is returned to indicate an Internal Server Error. If a property is missing in the HTTP MERGE request payload, the REST data service does not change the existing property value.

If there are duplicate properties specified in the payload, the last property is used. All the previous values with the same property name are ignored by the REST data service.

If the payload contains a non-existent property, the REST data service returns a 400 (Bad Request) response code to indicate the request sent by the client was syntactically incorrect.

As part of the serialization of a resource, if the payload of an Update request contains any of the key properties for the entity, the REST data service ignores those key values since entity keys are immutable.

For details on UpdateEntity request, refer to: MSDN Library: UpdateEntity Request.

An UpdateEntity request updates the city name of Customer('IBM') to 'Raleigh'.

AtomPub

- Method: PUT
- Request URI: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')
- Request Header: Content-Type: application/atom+xml
- Request Payload:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<entry xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
 xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
 xmlns="http://www.w3.org/2005/Atom">
 <category term="NorthwindGridModel.Customer"
  scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
 <title />
 <updated>2009-07-28T21:17:50.609Z</updated>
 <author>
  <name />
 </author>
 <id />
 <content type="application/xml">
  <m:properties>
   <d:customerId>IBM</d:customerId>
   <d:city>Raleigh</d:city>
   <d:companyName>IBM Corporation</d:companyName>
   <d:contactName>Big Blue</d:contactName>
   <d:country>USA</d:country>
  </m:properties>
 </content>
</entry>
```

- Response Payload: None
- Response Code: 204 No Content

JSON

- Method: PUT

- Request URI: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')
- Request Header: Content-Type: application/json
- Request Payload:

```
{"customerId":"IBM",
"city":"Raleigh",
"companyName":"IBM Corporation",
"contactName":"Big Blue",
"country":"USA",}
```

- Response Payload: None
- Response Code: 204 No Content

Update an entity primitive property

The UpdatePrimitiveProperty Request can update a property value of an eXtreme Scale entity. The property and value to be updated are in the request payload. The property cannot be a key property since eXtreme Scale does not allow clients to change entity keys.

For more details on the UpdatePrimitiveProperty request, refer to: MSDN Library: UpdatePrimitiveProperty Request.

Here is an UpdatePrimitiveProperty request example. In this example, we update the city name of Customer('IBM') to 'Raleigh'.

AtomPub

- Method: PUT
- Request URI: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/city
- Request Header: Content-Type: application/xml
- Request Payload:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<city xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices">
        Raleigh
</city>
```

- Response Payload: None
- Response Code: 204 No Content

JSON

- Method: PUT
- Request URI: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/city
- Request Header: Content-Type: application/json
- Request Payload: {"city":"Raleigh"}
- Response Payload: None
- Response Code: 204 No Content

Update an entity primitive property value

The UpdateValue Request can update a raw property value of an eXtreme Scale entity. The value to be updated is represented as a raw value in the request payload. The property cannot be a key property since eXtreme Scale does not allow clients to change entity keys.

The content type of the request can be “text/plain” or “application/octet-stream” depending on the property type. For more information, see Retrieving non-entities with REST data services.

For more details on the UpdateValue request, refer to: MSDN Library: UpdateValue Request

Here is an UpdateValue request example. In this example, update the city name of Customer('IBM') to 'Raleigh'.

- Method: PUT
- Request URI: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/city/$value
- Request Header: Content-Type: text/plain
- Request Payload: Raleigh
- Response Payload: None
- Response Code: 204 No Content

Update a link

The UpdateLink request can be used to establish an association between two eXtreme Scale entity instances. The association can be a single valued (to-one) relation or a multi-valued (to-many) relation.

Updating a link between two eXtreme Scale entity instances can establish associations or remove associations. For example, if the client establishes a to-one association between an `Order(orderId=5000,customer_customerId='IBM')` entity and `Customer('ALFKI')` instance, it has to dissociate the `Order(orderId=5000,customer_customerId='IBM')` entity and entity from its currently associated Customer instance.

If either of the entity instances specified in the UpdateLink request cannot be found, the REST data service returns a 404 (Not Found) response.

If the URI of the UpdateLink request specifies a non-existent association, the REST data service returns a 404 (Not Found) response to indicate the link cannot be found.

If the URI specified in the UpdateLink request payload does not resolve to the same entity or the same key as specified in the URI, if exists, then the eXtreme Scale Rest Data Service returns a 400 (Bad Request) response.

If the UpdateLink request payload contains multiple links, then the REST data service parses the first link only. The rest of the links are ignored.

For more details on the UpdateLink request, refer to: MSDN Library: UpdateLink Request.

Here is an UpdateLink request example. In this example, we update the customer relation of `Order(orderId=5000,customer_customerId='IBM')` entity and from Customer('IBM') to Customer('IBM').

Remember: The previous example is for illustration only. Because all associations are typically key-associations for a partitioned grid, the link cannot be changed.
AtomPub

- Method: PUT
- Request URI: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(101)/$links/customer
- Request Header: Content-Type: application/xml
- Request Payload:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<uri>
        http://host:1000/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')
</uri>
```

- Response Payload: None
- Response Code: 204 No Content

JSON

- Method: PUT
- Request URI: `http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=5000,customer_customerId='IBM')/$links/customer`
- Request Header: Content-Type: application/xml
- Request Payload: `{"uri": "http://host:1000/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')"}`
- Response Payload: None
- Response Code: 204 No Content

Related concepts:
Operations with the REST data service
REST data services overview
Related tasks:
Accessing data with the REST data service

Delete requests with REST data services

The WebSphere® eXtreme Scale REST data service can delete entities, property values and links.

Delete an entity

The DeleteEntity Request can delete an eXtreme Scale entity from the REST data service.

If any relation to the to-be-deleted entity has cascade-delete set, then the eXtreme Scale Rest data service will delete the related entity or entities. For more details on the DeleteEntity request, refer to MSDN Library: DeleteEntity Request.

The following DeleteEntity request deletes the customer with key 'IBM'.

- Method: DELETE
- Request URI: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')
- Request Payload: None
- Response Payload: None
- Response Code: 204 No Content

Delete a property value

The DeleteValue Request sets an eXtreme Scale entity property to null.

Any property of an eXtreme Scale entity can be set to null with a DeleteValue request. To set a property to null, ensure all of the following:

- For any primitive number type and its wrapper, BigInteger, or BigDecimal, the property value is set to 0.
- For Boolean or boolean type, the property value is set to false.
- For char or Character type, the property value is set to character #X1 (NIL).
- For enum type, the property value is set to the enum value with ordinal 0.
- For all other types, the property value is set to null.

However, such a delete request could be rejected by the database backend if, for example, the property is not nullable in the database. In this case, the REST data service returns a 500 (Internal Server Error) response. For more details on the DeleteValue request, refer to: MSDN Library: DeleteValue Request.

Here is a DeleteValue request example. In this example, we set the contact name of Customer('IBM') to null.

- Method: DELETE
- Request URI: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/contactName
- Request Payload: None
- Response Payload: None
- Response Code: 204 No Content

Delete a link

The DeleteLink request can removes an association between two eXtreme Scale entity instances. The association can be a to-one relation or a to-many relation. However, such a delete request could be rejected by the database backend if, for example, the foreign key constraint is set. In this case, the REST data service returns a 500 (Internal Server Error) response. For more details on the DeleteLink request, refer to: MSDN Library: DeleteLink Request.

The following DeleteLink request removes the association between Order(101) and its associated Customer.

- Method: DELETE
- Request URI: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(101)/$links/customer
- Request Payload: None
- Response Payload: None
- Response Code: 204 No Content

Related concepts:
Operations with the REST data service
REST data services overview
Related tasks:
Accessing data with the REST data service

System APIs and plug-ins

A plug-in is a component that provides a function to the pluggable components, which include ObjectGrid and BackingMap. To most effectively use eXtreme Scale as an in-memory data grid or database processing space, you should carefully determine how best you can maximize performance with available plug-ins.

- Managing plug-in life cycles
  You can manage plug-in life cycles with specialized methods from each plug-in, which are available to be invoked at designated functional points. Both initialize and destroy methods define the life cycle of plug-ins, which are controlled by their *owner* objects. An owner object is the object that actually uses the given plug-in. An owner can be a grid client, server, or a backing map.
- Plug-ins for multimaster replication
  Consider transforming cached objects to increase the performance of your cache. You can use the ObjectTransformer plug-in when your processor usage is high. Up to 60-70 percent of the total processor time is spent serializing and copying entries. By implementing the ObjectTransformer plug-in, you can serialize and deserialize objects with your own implementation. You can use a CollisionArbiter plug-in to define how change collisions are handled in your domains.
- Plug-ins for versioning and comparing cache objects
  Use the OptimisticCallback plug-in to customize versioning and comparison operations of cache objects when you are using the optimistic locking strategy.
- Plug-ins for serializing cached objects
  WebSphere® eXtreme Scale uses multiple Java processes to serialize the data, by converting the Java object instances to bytes and back to objects again, as needed, to move the data between client and server processes.
- Plug-ins for providing event listeners
  You can use the ObjectGridEventListener, MapEventListener, ObjectGridLifecycleListener and BackingMapLifecycleListener plug-ins to configure notifications for various events in the eXtreme Scale cache. Listener plug-ins are registered with an ObjectGrid or BackingMap instance like other eXtreme Scale plug-ins and add integration and customization points for applications and cache providers.
- Plug-ins for evicting cache objects
  WebSphere eXtreme Scale provides a default mechanism for evicting cache entries and a plug-in for creating custom evictors. An evictor controls the membership of entries in each BackingMap. The default evictor uses a time to live (TTL) eviction policy for each BackingMap. If you provide a pluggable evictor mechanism, it typically uses an eviction policy that is based on the number of entries instead of on time.
- Plug-ins for indexing data
  Depending on the type of indexes you want to build, WebSphere eXtreme Scale provides built-in plug-ins that you can add to the BackingMap to build an index.
- Plug-ins for communicating with databases
  With a Loader plug-in, an ObjectGrid map can behave as a memory cache for data that is typically kept in a persistent store on either the same system or some other 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 ObjectGrid. A loader has the logic for reading and writing data to and from a persistent store.
- Plug-ins for managing transaction life cycle events
  Use the TransactionCallback plug-in to customize versioning and comparison operations of cache objects when you are using the optimistic locking strategy.

Related concepts:
Java plug-ins overview
Related tasks:
Configuring eXtreme Scale plug-ins with OSGi Blueprint
Configuring OSGi-enabled plug-ins using the ObjectGrid descriptor XML file
Building eXtreme Scale dynamic plug-ins
Scenario: Using an OSGi environment to develop and run eXtreme Scale plug-ins

Related information:
Building OSGi applications with the Blueprint Container specification
OSGi Bundle Activator API documentation
Spring namespace schema

Managing plug-in life cycles

You can manage plug-in life cycles with specialized methods from each plug-in, which are available to be invoked at designated functional points. Both initialize and destroy methods define the life cycle of plug-ins, which are controlled by their *owner* objects. An owner object is the object that actually uses the given plug-in. An owner can be a grid client, server, or a backing map.

About this task

Similarly all plug-ins can implement the optional mix-in interfaces appropriate for their owner object. Any ObjectGrid plug-in can implement the optional mix-in interface ObjectGridPlugin. Any BackingMap plug-in can implement the optional mix-in interface BackingMapPlugin. The optional mix-in interfaces require implementation of several additional methods beyond the initialize() and destroy() methods for the basic plug-ins. For more information about these interfaces, see the API documentation.

When owner objects are initializing, those objects set attributes on the plug-in, then invoke the initialize method of their owned plug-ins. During the destroy cycle of owner objects, the destroy method of plug-ins are consequently invoked also. For details on the specifics of initialize and destroy methods, along with other methods capable with each plug-in, refer to the topics relevant to each plug-in.

As an example, consider a distributed environment. Both the client-side ObjectGrids and the server-side ObjectGrids can have their own plug-ins. The life cycle of a client-side ObjectGrid, and therefore, its plug-in instances are independent from all server-side ObjectGrid and plug-in instances.

In such a distributed topology, assume that you have an ObjectGrid named `myGrid` defined in the objectGrid.xml file and configured with a customized ObjectGridEventListener named myObjectGridEventListener. The objectGridDeployment.xml file defines the deployment policy for the `myGrid` ObjectGrid. Both the objectGrid.xml and `objectGridDeployment.xml` files are used to start container servers. During the startup of the container server, the server-side `myGrid` ObjectGrid instance is initialized. Meanwhile, the initialize method of the myObjectGridEventListener instance that is owned by the myObjectGrid instance is invoked. After the container server is started, your application can connect to the server-side `myGrid` ObjectGrid instance and obtain a client-side instance.

When obtaining the client-side myGrid ObjectGrid instance, the client-side `myGrid` instance goes through its own initialization cycle and invokes the initialize method of its own client-side myObjectGridEventListener instance. This client-side myObjectGridEventListener instance is independent from the server-side myObjectGridEventListener instance. Its life cycle is controlled by its owner, which is the client-side `myGrid` ObjectGrid instance.

If your application disconnects or destroys the client-side `myGrid` ObjectGrid instance, then the destroy method that belongs to the client-side myObjectGridEventListener instance is invoked automatically. However, this process has no impact on server-side myObjectGridEventListener instance. The destroy method of the server-side myObjectGridEventListener instance can only be invoked during the destroy life cycle of the server-side `myGrid` ObjectGrid instance, when stopping a container server. Specifically, when stopping a container server, the contained ObjectGrid instances are destroyed and the destroy method of all their owned plug-ins is invoked.

Although the previous example applies specifically to the case of a client and a server instance of an ObjectGrid, the owner of a plug-in can also be a BackingMap interface. In addition, carefully to determine your configurations for plug-ins that you might write, based on these life cycle considerations. Use the following topics to write plug-ins that provide extended life cycle management events that you can use to set up or remove resources in your environment:

- Writing an ObjectGridPlugin plug-in
  An ObjectGridPlugin is an optional mix-in interface that you can use to provide extended life cycle management events to all other ObjectGrid plug-ins.
- Writing a BackingMapPlugin plug-in
  A BackingMap plug-in implements the BackingMapPlugin mix-in interface, which you can use to receive extended capabilities for managing its life cycle.

Related concepts:
OSGi framework overview
Related information:
API documentation

Writing an ObjectGridPlugin plug-in

An ObjectGridPlugin is an optional mix-in interface that you can use to provide extended life cycle management events to all other ObjectGrid plug-ins.

About this task

Any ObjectGrid plug-in that implements the ObjectGridPlugin receives the extended set of life cycle events, and can provide more control, which you can use to set up or remove resources. In a container for a partitioned data grid, there will be one ObjectGrid instance (the plugin owner) for each partition managed by the container.When individual partitions are removed, the resources that are used by that ObjectGrid instance must also be removed. Therefore, you might need to close or end a resource, such as an open configuration file or a running thread that is managed by a plug-in, when the owning partition for that resource is removed.

The ObjectGridPlugin interface provides methods to set or modify the state of the plug-in, as well as methods to introspect the current state of the plug-in. All methods must be implemented correctly, and the WebSphere® eXtreme Scale runtime environment verifies the method behavior under certain circumstances.

For example, after calling the initialize() method, the eXtreme Scale runtime environment calls the isInitialized() method to ensure that the method successfully completed the appropriate initialization.

Procedure

1. Implement the ObjectGridPlugin interface so that the ObjectGridPlugin plug-in receives notifications about significant eXtreme Scale events. Three main categories of methods exist:

Properties methods	Purpose
setObjectGrid()	Called to set the ObjectGrid instance the plug-in is used for.
getObjectGrid()	Called to get or confirm the ObjectGrid instance the plug-in is used for.

Initialization methods	Purpose
initialize()	Called to initialize the ObjectGridPlugin.
isInitialized()	Called to get or confirm the initialization status of the plug-in.

Destruction methods	Purpose
destroy()	Called to destroy the ObjectGridPlugin.
isDestroyed()	Called to get or confirm the destroyed status of the plug-in.

   See the API documentation for more information about these interfaces.

2. Configure an ObjectGridPlugin plug-in with XML. Use the com.company.org.MyObjectGridPluginTxCallback class, which implements the TransactionCallback interface and the ObjectGridPlugin interface.
   In the following code example, the custom transaction callback, which will ultimately receive extended life cycle events, is generated and added to an ObjectGrid.
   Important: The TransactionCallback interface already has an initialize method, a new initialize method is added as well as the destroy method and other ObjectGridPlugin methods. Each method is used, and the initialize methods only perform initialization one time. The following XML creates a configuration that uses the enhanced TransactionCallback interface.
   The following text must be in the myGrid.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">
            <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsobjgridplug_TransactionCallba
ck"
                  className="com.company.org.MyObjectGridPluginTxCallback" />
            <backingMap name="Book"/>
        </objectGrid>
    </objectGrids>
</objectGridConfig>
```

   Notice the bean declarations come before the `backingMap` declarations.

3. Provide the myGrid.xml file to the ObjectGridManager plug-in to facilitate the creation of this configuration.

Related tasks:
Writing a BackingMapPlugin plug-in
Related information:
../com.ibm.websphere.extremescale.javadoc.doc/topics/com/ibm/websphere/objectgrid/management/package-summary.html

Writing a BackingMapPlugin plug-in

A BackingMap plug-in implements the BackingMapPlugin mix-in interface, which you can use to receive extended capabilities for managing its life cycle.

About this task

Any existing BackingMap plug-in that also implements the BackingMapPlugin interface will automatically receive the extended set of lifecycle events during its construction and use.

The BackingMapPlugin interface provides methods to set or modify the state of the plug-in, as well as methods to introspect the current state of the plug-in.

All methods must be implemented correctly, and the WebSphere® eXtreme Scale runtime environment verifies the method behavior under certain circumstances. For example, after calling the initialize() method, the eXtreme Scale runtime environment calls the isInitialized() method to ensure that the method successfully completed the appropriate initialization.

Procedure

1. Implement the BackingMapPlugin interface so that the BackingMapPlugin plug-in receives notifications about significant eXtreme Scale events. Three main categories of methods exist:

Properties methods	Purpose
setBackingMap()	Called to set the BackingMap instance the plug-in is used for.
getBackingMap()	Called to get or confirm the BackingMap instance the plug-in is used for.

Initialization methods	Purpose
initialize()	Called to initialize the BackingMapPlugin plug-in.
isInitialized()	Called to get or confirm the initialization status of the plug-in.

Destruction methods	Purpose
destroy()	Called to destroy the BackingMapPlugin plug-in.
isDestroyed()	Called to get or confirm the destroyed status of the plug-in.

See the API documentation for more information about these interfaces.

2. Configure a BackingMapPlugin plug-in with XML. Assume that the class name of an eXtreme Scale Loader plug-in is the com.company.org.MyBackingMapPluginLoader class, which implements the Loader interface and the BackingMapPlugin interface.
In the following code example, the custom transaction callback, which will ultimately receive extended life cycle events, is generated and added to a BackingMap.

   You can also configure a BackingMapPlugin plug-in using XML. The following text must be in the myGrid.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">
            <backingMap name="Book" pluginCollectionRef="myPlugins" />
        </objectGrid>
    </objectGrids>
    <backingMapPluginCollections>
        <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsbackmapplug_myPlugins">
            <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsbackmapplug_Loader"
                  className="com.company.org.MyBackingMapPluginLoader" />
        </backingMapPluginCollection>
    </backingMapPluginCollections>
</objectGridConfig>
```

3. Provide the myGrid.xml file to the ObjectGridManager plug-in to facilitate the creation of this configuration.

Results

The BackingMap instance that is created has a Loader that receives BackingMapPlugin life cycle events.
Related tasks:
Writing an ObjectGridPlugin plug-in
Related information:
../com.ibm.websphere.extremescale.javadoc.doc/topics/com/ibm/websphere/objectgrid/management/package-summary.html

Plug-ins for multimaster replication

Consider transforming cached objects to increase the performance of your cache. You can use the ObjectTransformer plug-in when your processor usage is high. Up to 60-70 percent of the total processor time is spent serializing and copying entries. By implementing the ObjectTransformer plug-in, you can serialize and deserialize objects with your own implementation. You can use a CollisionArbiter plug-in to define how change collisions are handled in your domains.

- Developing custom arbiters for multi-master replication
  Change collisions might occur if the same records can be changed simultaneously in two places. In a multi-master replication topology, catalog service domains detect collisions automatically. When a catalog service domain detects a collision, it invokes an arbiter. Typically, collisions are resolved with the default collision arbiter. However, an application can provide a custom collision arbiter.

Developing custom arbiters for multi-master replication

Change collisions might occur if the same records can be changed simultaneously in two places. In a multi-master replication topology, catalog service domains detect collisions automatically. When a catalog service domain detects a collision, it invokes an arbiter. Typically, collisions are resolved with the default collision arbiter. However, an application can provide a custom collision arbiter.

Before you begin

- See Planning multiple data center topologies for more information about planning and designing the multi-master replication topology.
- See Configuring multiple data center topologies for more information about setting up links between catalog service domains.

About this task

If a catalog service domain receives a replicated entry that collides with a collision record, the default arbiter uses the changes from the lexically lowest named catalog service domain. For example, if domain A and B generate a conflict for a record, then the change from domain B is ignored. Domain A keeps its version and the record in domain B is changed to match the record from domain A. Domain names are converted to uppercase for comparison.

An alternative option is for the multi-master replication topology to call on a custom collision plug-in to decide the outcome. These instructions outline how to develop a custom collision arbiter and configure a multi-master replication topology to use it.

Procedure

1. Develop a custom collision arbiter and integrate it into your application.
   The class must implement the interface:

   ```
   com.ibm.websphere.objectgrid.revision.CollisionArbiter
   ```

   A collision plug-in has three choices for deciding the outcome of a collision. It can choose the local copy or the remote copy or it can provide a revised version of the entry. A catalog service domain provides the following information to a custom collision arbiter:
   - The existing version of the record
   - The collision version of the record
   - A Session object that must be used to create the revised version of the collided entry

   The plug-in method returns an object that indicates its decision. The method invoked by the domain to call the plug-in must return true or false, where false means to ignore the collision. When the collision is ignored, the local version remains unchanged and the arbiter forgets that it ever saw the existing version. The method returns a true value if the method used the provided session to create a new, merged version of the record, reconciling the change.
2. In the objectgrid.xml file, specify the custom arbiter plug-in.
   The ID must be `CollisionArbiter`.

   ```
   <dgc:objectGrid name="revisionGrid" txTimeout="10">
          <dgc:bean className="com.you.your_application.
                                                 CustomArbiter"
   id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsmultimasterprog_CollisionArbi
   ter">
                  <dgc:property name="property" type="java.lang.String"
                                                 value="propertyValue"/>
          </dgc:bean>
   </dgc:objectGrid>
   ```

Related concepts:
Planning multiple data center topologies
Topologies for multi-master replication
Configuration considerations for multi-master topologies
Design considerations for multi-master replication
Loader considerations in a multi-master topology
Related tasks:
Configuring multiple data center topologies

Plug-ins for versioning and comparing cache objects

Use the OptimisticCallback plug-in to customize versioning and comparison operations of cache objects when you are using the optimistic locking strategy.

You can provide a pluggable optimistic callback object that implements the com.ibm.websphere.objectgrid.plugins.OptimisticCallback interface. For entity maps, a high performance OptimisticCallback plug-in is automatically configured.

Purpose

Use the OptimisticCallback interface to provide optimistic comparison operations for the values of a map. An OptimisticCallback plug-in is required when you use the optimistic locking strategy. The product provides a default OptimisticCallback implementation. However, typically your application must plug in its own implementation of the OptimisticCallback interface.

Default implementation

The eXtreme Scale framework provides a default implementation of the OptimisticCallback interface that is used if the application does not plug in an application-provided OptimisticCallback object. The default implementation always returns the special value of NULL_OPTIMISTIC_VERSION as the version object for the value and never updates the version object. This action makes optimistic comparison a "no operation" function. In most cases, you do not want the "no operation" function to occur when you are using the optimistic locking strategy. Your applications must implement the OptimisticCallback interface and plug in their own OptimisticCallback implementations so that the default implementation is not used. However, at least one scenario exists where the default provided OptimisticCallback implementation is useful. Consider the following situation:

- A loader is plugged in for the backing map.
- The loader knows how to perform the optimistic comparison without assistance from an OptimisticCallback plug-in.

How can the loader perform optimistic versioning without assistance from an OptimisticCallback object? The loader has knowledge of the value class object and knows which field of the value object is used as an optimistic versioning value. For example, suppose the following interface is used for the value object for the employees map:

```
public interface Employee
{
    // Sequential sequence number used for optimistic versioning.
    public long getSequenceNumber();
    public void setSequenceNumber(long newSequenceNumber);
    // Other get/set methods for other fields of Employee object.
}
```

In this example, the loader knows that it can use the getSequenceNumber method to get the current version information for an Employee value object. The loader increments the returned value to generate a new version number before it updates the persistent storage with the new Employee value. For a Java™ database connectivity (JDBC) loader, the current sequence number in the WHERE clause of an overqualified SQL UPDATE statement is used, and it uses the new generated sequence number to set the sequence number column to the new sequence number value. Another possibility is that the loader makes use of some backend-provided function that automatically updates a hidden column that can be used for optimistic versioning.

In some situations, a stored procedure or trigger can possibly be used to help maintain a column that holds versioning information. If the loader is using one of these techniques for maintaining optimistic versioning information, then the application does not need to provide an OptimisticCallback implementation. The default OptimisticCallback implementation is usable in this scenario because the loader can handle optimistic versioning without any assistance from an OptimisticCallback object.

Default implementation for entities

Entities are stored in the ObjectGrid using tuple objects. The default OptimisticCallback implementation behavior is similar to the behavior for non-entity maps. However, the version field in the entity is identified using the @Version annotation or the version attribute in the entity descriptor XML file.

The version attribute can be of the following types: int, Integer, short, Short, long, Long or java.sql.Timestamp. An entity must only have one version attribute defined. Only set the version attribute during construction. After the entity is persisted, the value of the version attribute should not be modified.

If a version attribute is not configured and the optimistic locking strategy is used, then the entire tuple is implicitly versioned using the entire state of the tuple, which is much more expensive

In the following example, the Employee entity has a long version attribute named SequenceNumber:

```
@Entity
public class Employee
{
private long sequence;
    // Sequential sequence number used for optimistic versioning.
    @Version
    public long getSequenceNumber() {
        return sequence;
    }
    public void setSequenceNumber(long newSequenceNumber) {
        this.sequence = newSequenceNumber;
    }
    // Other get/set methods for other fields of Employee object.
}
```

Writing an OptimisticCallback plug-in

An OptimisticCallback plug-in must to implement the OptimisticCallback interface and follow the common ObjectGrid plug-in conventions. See OptimisticCallback interface for more information.

The following list provides a description or consideration for each of the methods in the OptimisticCallback interface:

NULL_OPTIMISTIC_VERSION

This special value is returned by the getVersionedObjectForValue method if the OptimisticCallback implementation does not require version checking. The built-in plugin implementation of the com.ibm.websphere.objectgrid.plugins.builtins.NoVersioningOptimisticCallback class uses this value because versioning is disabled when you are specifying this plug-in implementation.

getVersionedObjectForValue method

The getVersionedObjectForValue method might return a copy of the value or an attribute of the value that can be used for versioning purposes. This method is called whenever an object is associated with a transaction. When no Loader is plugged into a backing map, the backing map uses this value at commit time to perform an optimistic version comparison. The optimistic version comparison is used by the backing map to ensure that the version has not changed after this transaction first accessed the map entry that was modified by this transaction. If another transaction had already modified the version for this map entry, the version comparison fails and the backing map displays an OptimisticCollisionException exception to force the transaction to roll back. If a Loader is plugged in, the backing map does not use the optimistic versioning information. Instead, the Loader is responsible for performing the optimistic versioning comparison and updating the versioning information when necessary. The Loader typically gets the initial versioning object from the LogElement passed to the batchUpdate method on the loader, which is called when a flush operation occurs or a transaction is committed.

The following code shows the implementation used by the EmployeeOptimisticCallbackImpl object:

```
public Object getVersionedObjectForValue(Object value)
{
    if (value == null)
    {
        return null;
    }
    else
```

```
    {
        Employee emp = (Employee) value;
        return new Long( emp.getSequenceNumber() );
    }
}
```

As demonstrated in the previous example, the sequenceNumber attribute is returned in a java.lang.Long object as expected by the Loader, which implies that the same person that wrote the Loader either wrote the EmployeeOptimisticCallbackImpl implementation or worked closely with the person that implemented the EmployeeOptimisticCallbackImpl - for example, agreed on the value returned by the getVersionedObjectForValue method. The default OptimisticCallback plug-in returns the special value NULL_OPTIMISTIC_VERSION as the version object.

updateVersionedObjectForValue method

This method is called whenever a transaction has updated a value and a new versioned object is needed. If the getVersionedObjectForValue method returns an attribute of the value, this method typically updates the attribute value with a new version object. If getVersionedObjectForValue method returns a copy of the value, this method typically does not complete any actions. The default OptimisticCallback plug-in does not complete any actions with this method because the default implementation of getVersionedObjectForValue always returns the special value NULL_OPTIMISTIC_VERSION as the version object. The following example shows the implementation used by the EmployeeOptimisticCallbackImpl object that is used in the OptimisticCallback section:

```
public void updateVersionedObjectForValue(Object value)
{
    if ( value != null )
    {
        Employee emp = (Employee) value;
        long next = emp.getSequenceNumber() + 1;
        emp.updateSequenceNumber( next );
    }
}
```

As demonstrated in the previous example, the sequenceNumber attribute increments by one so that the next time the getVersionedObjectForValue method is called, the java.lang.Long value that is returned has a long value that is the original sequence number value plus one, for example, is the next version value for this employee instance. This example implies that the same person that wrote the Loader either wrote EmployeeOptimisticCallbackImpl or worked closely with the person that implemented the EmployeeOptimisticCallbackImpl.

serializeVersionedValue method

This method writes the versioned value to the specified stream. Depending on the implementation, the versioned value can be used to identify optimistic update collisions. In some implementations, the versioned value is a copy of the original value. Other implementations might have a sequence number or some other object to indicate the version of the value. Because the actual implementation is unknown, this method is provided to perform the appropriate serialization. The default implementation calls the writeObject method.

inflateVersionedValue method

This method takes the serialized version of the versioned value and returns the actual versioned value object. Depending on the implementation, the versioned value can be used to identify optimistic update collisions. In some implementations, the versioned value is a copy of the original value. Other implementations might have a sequence number or some other object to indicate the version of the value. Because the actual implementation is unknown, this method is provided to perform the appropriate deserialization. The default implementation calls the readObject method.

Using application-provided OptimisticCallback object

You have two approaches to add an application-provided OptimisticCallback object into the BackingMap configuration: XML configuration and programmatic configuration.

Programmatically plug in an OptimisticCallback object

The following example demonstrates how an application can programmatically plug in an OptimisticCallback object for the employee backing map in the local grid1 ObjectGrid instance:

```
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogManager.createObjectGrid( "grid1" );
BackingMap bm = dg.defineMap("employees");
EmployeeOptimisticCallbackImpl cb = new EmployeeOptimisticCallbackImpl();
bm.setOptimisticCallback( cb );
```

XML configuration approach to plug in an OptimisticCallback object

The application can use an XML file to plug in its OptimisticCallback object as shown in the following example:

```
<?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="grid1">
```

```
        <backingMap name="employees" pluginCollectionRef="employees" lockStrategy="OPTIMISTIC" />
    </objectGrid>
</objectGrids>

<backingMapPluginCollections>
    <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxsplugcall_employees">
        <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxsplugcall_OptimisticCallback"
className="com.xyz.EmployeeOptimisticCallbackImpl" />
    </backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

Plug-ins for serializing cached objects

WebSphere® eXtreme Scale uses multiple Java processes to serialize the data, by converting the Java object instances to bytes and back to objects again, as needed, to move the data between client and server processes.

To serialize data in eXtreme Scale, you can use Java serialization, the ObjectTransformer plug-in, or the DataSerializer plug-ins.

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.

- Serializer programming overview
  You can use the DataSerializer plug-ins to write optimized serializers for storing Java objects and other data in binary form in the grid. The plug-in also provides methods that you can use to query attributes within the binary data without requiring the entire data object to be inflated.
- Avoiding object inflation when updating and retrieving cache data
  You can use the DataSerializer plug-ins to bypass automatic object inflation and manually retrieve attributes from data that has already been serialized. You can also use the DataSerializer to insert and update data in its serialized form. This usage can be useful when only part of the data needs to be accessed or when the data needs to be passed between systems.
- ObjectTransformer plug-in
  With the ObjectTransformer plug-in, you can serialize, deserialize, and copy objects in the cache for increased performance.

Related concepts:
Serialization using Java
Serialization using the DataSerializer plug-ins
ObjectTransformer plug-in
Samples
Java plug-ins overview
Serializer programming overview
IBM eXtremeMemory
Serialization overview
Related tasks:
Avoiding object inflation when updating and retrieving cache data
Planning to use IBM eXtremeMemory
Related information:
Oracle Java Serialization API

Serializer programming overview

You can use the DataSerializer plug-ins to write optimized serializers for storing Java™ objects and other data in binary form in the grid. The plug-in also provides methods that you can use to query attributes within the binary data without requiring the entire data object to be inflated.

The DataSerializer plug-ins include three main plug-ins and several optional mix-in interfaces. The MapSerializerPlugin plug-in includes metadata about the relationship between a map and other maps. It also includes a reference to a KeySerializerPlugin and ValueSerializerPlugin. The key and value serializer plug-ins include metadata and serialization code responsible for interacting with the respective key and value data for a map. A MapSerializerPlugin plug-in must include one or both key and value serializers.

The KeySerializerPlugin plug-in provides methods and metadata for serializing, inflating and introspecting keys. The ValueSerializer plug-in provides methods and metadata for serializing, inflating and introspecting values. Both interfaces have different requirements. For details on what methods are available on the DataSerializer plug-ins, see the API documentation for the com.ibm.websphere.objectgrid.plugins.io package.

MapSerializerPlugin plug-in
: The MapSerializerPlugin is the main plug-in point to the BackingMap interface, and it includes two nested plug-ins: the KeySerializerPlugin and ValueSerializerPlugin plug-ins. Since eXtreme Scale does not support nested or wired plug-ins, the BasicMapSerializerPlugin plug-in accesses these nested plug-ins artificially. When you use these plug-ins with the OSGi framework, the only proxy is the MapSerializerPlugin plug-in. All nested plug-ins must not be cached within other dependent plug-ins, such as loaders, unless those plug-ins also listen for BackingMap life cycle events. This is important when running in an OSGi framework, because references to those plug-ins can continue to be refreshed.

KeySerializerPlugin plug-in
: The KeySerializerPlugin plug-in extends the DataSerializer interface and includes other mix-in interfaces and metadata that describes the key. Use this plug-in to serialize and inflate key data objects and attributes.

ValueSerializerPlugin plug-in
: The ValueSerializerPlugin plug-in extends the DataSerializer interface, but exposes no additional methods. Use this plug-in to serialize and inflate value data objects and attributes.

Optional and mix-in interfaces

Optional and mix-in interfaces provide additional capabilities, such as:

Optimistic versioning
: The Versionable interface allows the ValueSerializerPlugin plug-in to handle version checking and version updates when using optimistic locking. If the Versioning is not implemented and optimistic locking is enabled, then the version is the entire serialized form of the data object value.

Non-hashCode-based routing
: The Partitionable interface allows KeySerializerPlugin implementations to route requests to explicit partitions. This is equivalent to the PartitionableKey interface, when used with the ObjectMap API without a KeySerializerPlugin. Without this feature, the key is routed to the partition based on the resulting hashCode.

UserReadable (toString) interface
: The UserReadable (toString) interface allows all DataSerializer implementations to provide an alternative method to display data in log files and debuggers. With this capability, you can hide sensitive data such as passwords. If DataSerializer implementations do not implement this interface, then the runtime environment might call toString() directly on the object or include alternative representations, if appropriate.

Evolution support
: The Mergeable interface can be implemented on ValueSerializerPlugin plug-in implementations to allow interoperability between multiple versions of objects when there are different DataSerializer versions updating data in the grid through it's lifetime. The Mergeable methods allow the DataSerializer plug-in to retain any data that it might not otherwise understand.

Related concepts:
Serialization using Java
Serialization overview
Serialization using the DataSerializer plug-ins
ObjectTransformer plug-in
Samples
Java plug-ins overview
Plug-ins for serializing cached objects
IBM eXtremeMemory
Serialization overview
Samples
Related tasks:
Avoiding object inflation when updating and retrieving cache data
Planning to use IBM eXtremeMemory
Avoiding object inflation when updating and retrieving cache data
Programming to use the OSGi framework
Related information:
Oracle Java Serialization API
DataSerializer API documentation

Avoiding object inflation when updating and retrieving cache data

You can use the DataSerializer plug-ins to bypass automatic object inflation and manually retrieve attributes from data that has already been serialized. You can also use the DataSerializer to insert and update data in its serialized form. This usage can be useful when only part of the data needs to be accessed or when the data needs to be passed between systems.

About this task

This task uses the COPY_TO_BYTES_RAW copy mode with the MapSerializerPlugin and ValueSerializerPlugin plug-ins. The MapSerializer is the main plug-in point to the BackingMap interface. It includes two nested plug-ins, the KeyDataSerializer and ValueDataSerializer. Since the product does not support nested plug-ins, the BaseMapSerializer supports nested or wired plug-ins artificially. Therefore, when you use these APIs in the OSGi container, the MapSerializer is the only proxy. All nested plug-ins must not be cached within other dependent plug-ins, such as a loader, unless it also listens for BackingMap life cycle events, so that it can refresh its supporting references.

When COPY_TO_BYTES_RAW is set, all ObjectMap methods return SerializedValue objects, allowing the user to retrieve the serialized form or the Java object form of the value.

When using a KeySerializerPlugin plug-in, all methods that return keys, such as the MapIndexPlugin or Loader plug-ins return SerializedKey objects.

When the data is already in serialized form, the data is inserted using the same SerializedKey and SerializedValue objects. When the data is in byte[] format, the DataObjectKeyFactory and DataObjectValueFactory factories are used to create the appropriate key or value wrapper. The factories are available on the DataObjectContext, which can be accessed from the SerializerAccessor for the BackingMap, or from within the DataSerializer implementation.

The example in this topic demonstrates how to complete the following actions:

Procedure

1. Use the DataSerializer plug-ins to serialize and inflate data objects.
2. Retrieve serialized values.
3. Retrieve individual attributes from a serialized value.
4. Insert pre-serialized keys and values.

Example

Use this example to update and retrieve cache data:

```
import java.io.IOException;
import com.ibm.websphere.objectgrid.CopyMode;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridException;
import com.ibm.websphere.objectgrid.ObjectMap;
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.io.XsDataOutputStream;
import com.ibm.websphere.objectgrid.plugins.io.SerializerAccessor;
import com.ibm.websphere.objectgrid.plugins.io.ValueSerializerPlugin;
import com.ibm.websphere.objectgrid.plugins.io.dataobject.DataObjectContext;
import com.ibm.websphere.objectgrid.plugins.io.dataobject.SerializedKey;
import com.ibm.websphere.objectgrid.plugins.io.dataobject.SerializedValue;

    /**
     * Use the DataSerializer to serialize an Order key.
     */
    public byte[] serializeOrderKey(ObjectGrid grid, String key)
            throws IOException {
        SerializerAccessor sa = grid.getMap("Order").getSerializerAccessor();
        DataObjectContext dftObjCtx = sa.getDefaultContext();
        XsDataOutputStream out = dftObjCtx.getDataStreamManager()
                .createOutputStream();
        sa.getMapSerializerPlugin().getKeySerializerPlugin()
                .serializeDataObject(sa.getDefaultContext(), key, out);
        return out.toByteArray();
    }

    /**
     * Use the DataSerializer to serialize an Order value.
     */
    public byte[] serializeOrderValue(ObjectGrid grid, Order value)
            throws IOException {
        SerializerAccessor sa = grid.getMap("Order").getSerializerAccessor();
        DataObjectContext dftObjCtx = sa.getDefaultContext();
        XsDataOutputStream out = dftObjCtx.getDataStreamManager()
                .createOutputStream();
        sa.getMapSerializerPlugin().getValueSerializerPlugin()
                .serializeDataObject(sa.getDefaultContext(), value, out);
        return out.toByteArray();
    }

    /**
     * Retrieve a single Order in serialized form.
     */
    public byte[] fetchOrderRAWBytes(Session session, String key)
            throws ObjectGridException {
        ObjectMap map = session.getMap("Order");

        // Override the CopyMode to retrieve the serialized form of the value.
        // This process affects all API methods from this point on for the life
        // of the Session.
        map.setCopyMode(CopyMode.COPY_TO_BYTES_RAW, null);
        SerializedValue serValue = (SerializedValue) map.get(key);

        if (serValue == null)
            return null;

        // Retrieve the byte array and return it to the caller.
        return serValue.getInputStream().toByteArray();
    }

    /**
     * Retrieve one or more attributes from the Order without inflating the
     * Order object.
     */
    public Object[] fetchOrderAttribute(Session session, String key,
            String... attributes) throws ObjectGridException, IOException {
        ObjectMap map = session.getMap("Order");

        // Override the CopyMode to retrieve the serialized form of the value.
        // This process affects all API methods from this point on for the life
        // of the Session.
        map.setCopyMode(CopyMode.COPY_TO_BYTES_RAW, null);
        SerializedValue serValue = (SerializedValue) map.get(key);

        if (serValue == null)
            return null;

        // Retrieve a single attribute from the byte buffer.
```

```
        ValueSerializerPlugin valSer = session.getObjectGrid()
                .getMap(map.getName()).getSerializerAccessor()
                .getMapSerializerPlugin().getValueSerializerPlugin();
        Object attrCtx = valSer.getAttributeContexts(attributes);
        return valSer.inflateDataObjectAttributes(serValue.getContext(),
                serValue.getInputStream(), attrCtx);
    }

    /**
     * Inserts a pre-serialized key and value into the Order map.
     */
    public void insertRAWOrder(Session session, byte[] key, byte[] value)
            throws ObjectGridException {
        ObjectMap map = session.getMap("Order");

        // Get a referece to the default DataObjectContext for the map.
        DataObjectContext dftDtaObjCtx = session.getObjectGrid()
                .getMap(map.getName()).getSerializerAccessor()
                .getDefaultContext();

        // Wrap the key and value in a SerializedKey and SerializedValue
        // wrapper.
        SerializedKey serKey = dftDtaObjCtx.getKeyFactory().createKey(key);
        SerializedValue serValue = dftDtaObjCtx.getValueFactory().createValue(
                value);

        // Insert the serialized form of the key and value.
        map.insert(serKey, serValue);
    }
}
```

Related concepts:
Serialization using Java
Serialization overview
Serialization using the DataSerializer plug-ins
ObjectTransformer plug-in
Samples
Java plug-ins overview
Plug-ins for serializing cached objects
Serializer programming overview
IBM eXtremeMemory
Serializer programming overview
Serialization overview
Samples
Related tasks:
Programming to use the OSGi framework
Related information:
Oracle Java Serialization API
DataSerializer API documentation

ObjectTransformer plug-in

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="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxsplugobjtrans_2_myMap">
            <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxsplugobjtrans_2_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);
```

Related concepts:
Serialization using Java
Serialization overview
Serialization using the DataSerializer plug-ins
Samples
Java plug-ins overview
Plug-ins for serializing cached objects
Serializer programming overview
IBM eXtremeMemory
Tuning serialization performance
Tuning serialization
Using a loader with entity maps and tuples
Tuning copy operations with the ObjectTransformer interface
Related tasks:

Avoiding object inflation when updating and retrieving cache data
Planning to use IBM eXtremeMemory
Related information:
Oracle Java Serialization API

Plug-ins for providing event listeners

You can use the ObjectGridEventListener, MapEventListener, ObjectGridLifecycleListener and BackingMapLifecycleListener plug-ins to configure notifications for various events in the eXtreme Scale cache. Listener plug-ins are registered with an ObjectGrid or BackingMap instance like other eXtreme Scale plug-ins and add integration and customization points for applications and cache providers.

ObjectGridEventListener plug-in

An ObjectGridEventListener plug-in provides eXtreme Scale life cycle events for the ObjectGrid instance, shards, and transactions. Use the ObjectGridEventListener plug-in to receive notifications when significant events occur on an ObjectGrid. These events include ObjectGrid initialization, the beginning of a transaction, the ending a transaction, and destroying an ObjectGrid. To listen for these events, create a class that implements the ObjectGridEventListener interface and add it to the eXtreme Scale.
For more information about writing an ObjectGridEventListener plug-in, see ObjectGridEventListener plug-in. You can also refer to the API documentation for more information.

Adding and removing ObjectGridEventListener instances

An ObjectGrid can have multiple ObjectGridEventListener listeners. Add and remove the listeners using the addEventListener, and removeEventListener methods on the ObjectGrid interface. You can also declaratively registerObjectGridEventListener plug-ins with the ObjectGrid descriptor file. For examples, see ObjectGridEventListener plug-in.

MapEventListener plug-in

A MapEventListener plug-in provides callback notifications and significant cache state changes that occur for a BackingMap instance. For details on writing a MapEventListener plug-in, see MapEventListener plug-in. You can also refer to the API documentation for more information.
Adding and removing MapEventListener instances

An eXtreme Scale can have multiple MapEventListener listeners. Add and remove listeners with the addMapEventListener, and removeMapEventListener methods on the BackingMap interface. You can also declaratively register MapEventListener listeners with the ObjectGrid descriptor file. For examples, see MapEventListener plug-in.

BackingMapLifecycleListener plug-in

A BackingMapLifecycleListener plug-in provides callback notifications for life cycle state changes that occur for a BackingMap instance. The BackingMap instance proceeds through a predefined set of states during its life time.
Adding and removing BackingMapLifecycleListener instances

An eXtreme Scale server can have multiple BackingMapLifecycleListener listeners. Add and remove listeners with the addMapEventListener and removeMapEventListener methods on the BackingMap interface. Any BackingMap plug-ins that implement the BackingMapLifecycleListener interface are also automatically added as a BackingMapLifecycleListener for the ObjectGrid instance they are registered with. You can also declaratively register BackingMapLifecycleListener listeners with the ObjectGrid descriptor file. For examples, see BackingMapLifecycleListener plug-in.

ObjectGridLifecycleListener plug-in

An ObjectGridLifecycleListener plug-in provides callback notifications for life cycle state changes that occur for an ObjectGrid instance. The ObjectGrid instance proceeds through a predefined set of states during its life time.
Adding and removing ObjectGridLifecycleListener instances

An eXtreme Scale can have multiple ObjectGridLifecycleListener listeners. Add and remove listeners with the addEventListener and removeEventListener methods on the ObjectGrid interface. Any ObjectGrid plug-ins that implement the ObjectGridLifecycleListener interface are automatically added as an ObjectGridLifecycleListener for the ObjectGrid instance that they are registered with. You can also declaratively register ObjectGridLifecycleListener listeners with the ObjectGrid deployment descriptor file. For examples, see ObjectGridLifecycleListener plug-in.

- MapEventListener plug-in
  A MapEventListener plug-in provides callback notifications and significant cache state changes that occur for a BackingMap object: when a map has finished pre-loading or when an entry is evicted from the map. A particular MapEventListener plug-in is a custom class you write implementing the MapEventListener interface.
- ObjectGridEventListener plug-in
  An ObjectGridEventListener plug-in provides WebSphere eXtreme Scale life cycle events for the ObjectGrid, shards and transactions. An ObjectGridEventListener plug-in provides notifications when an ObjectGrid is initialized or destroyed, and when a transaction is started or ended. ObjectGridEventListener plug-ins are custom classes you write implementing the ObjectGridEventListener interface. Optionally, the implementation includes ObjectGridEventGroup sub-interfaces and follow the common eXtreme Scale plug-in conventions.
- BackingMapLifecycleListener plug-in
  A BackingMapLifecycleListener plug-in receives notification of WebSphere eXtreme Scale life cycle state change events for the backing map.
- ObjectGridLifecycleListener plug-in
  An ObjectGridLifecycleListener plug-in receives notification of WebSphere eXtreme Scale life cycle, state change events for the data grid.

Related tasks:
Administering OSGi-enabled services using the xscmd utility
Managing ObjectGrid availability
Updating OSGi services for eXtreme Scale plug-ins with xscmd

MapEventListener plug-in

A MapEventListener plug-in provides callback notifications and significant cache state changes that occur for a BackingMap object: when a map has finished pre-loading or when an entry is evicted from the map. A particular MapEventListener plug-in is a custom class you write implementing the MapEventListener interface.

MapEventListener plug-in conventions

When you develop a MapEventListener plug-in, you must follow common plug-in conventions. For more information about plug-in conventions, see Java plug-ins overview. For other types of listener plug-ins, see Plug-ins for providing event listeners.
After you write a MapEventListener implementation, you can plug it in to the BackingMap configuration programmatically or with an XML configuration.

Write a MapEventListener implementation

Your application can include an implementation of the MapEventListener plug-in. The plug-in must implement the MapEventListener interface to receive significant events about a map. Events are sent to the MapEventListener plug-in when an entry is evicted from the map and when the preload of a map completes.

Programmatically plug in a MapEventListener implementation

The class name for the custom MapEventListener is the com.company.org.MyMapEventListener class. This class implements the MapEventListener interface. The following code snippet creates the custom MapEventListener object and adds it to a BackingMap object:

```
ObjectGridManager objectGridManager =
        ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid", false);
BackingMap myMap = myGrid.defineMap("myMap");
MyMapEventListener myListener = new MyMapEventListener();
myMap.addMapEventListener(myListener);
```

Plug in a MapEventListener implementation using XML

A MapEventListner implementation can be configured using XML. The following XML must be in the myGrid.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">
            <backingMap name="myMap" pluginCollectionRef="myPlugins" />
        </objectGrid>
    </objectGrids>
    <backingMapPluginCollections>
        <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsmapeventlistplugin_myPlugins">
            <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsmapeventlistplugin_MapEventListene
r" className=
                                                    "com.company.org.MyMapEventListener" />
        </backingMapPluginCollection>
    </backingMapPluginCollections>
</objectGridConfig>
```

Providing this file to the ObjectGridManager instance facilitates the creation of this configuration. The following code snippet shows how to create an ObjectGrid instance using this XML file. The newly created ObjectGrid instance has a MapEventListener set on the myMap BackingMap.

```
ObjectGridManager objectGridManager =
        ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid =
        objectGridManager.createObjectGrid("myGrid", new URL("file:etc/test/myGrid.xml"),
                true, false);
```

ObjectGridEventListener plug-in

An ObjectGridEventListener plug-in provides WebSphere® eXtreme Scale life cycle events for the ObjectGrid, shards and transactions. An ObjectGridEventListener plug-in provides notifications when an ObjectGrid is initialized or destroyed, and when a transaction is started or ended. ObjectGridEventListener plug-ins are custom classes you write implementing the ObjectGridEventListener interface. Optionally, the implementation includes ObjectGridEventGroup sub-interfaces and follow the common eXtreme Scale plug-in conventions.

Overview

An ObjectGridEventListener plug-in is useful when a Loader plug-in is available, and you must initialize Java™ Database Connectivity (JDBC) connections or connections to a back end when transactions start and end. Typically, an ObjectGridEventListener plug-in and a Loader plug-in are written together.

Writing an ObjectGridEventListener plug-in

An ObjectGridEventListener plug-in must implement the ObjectGridEventListener interface to receive notifications about significant eXtreme Scale events. To receive additional event notifications, you can implement the following interfaces. These sub-interfaces are included in the ObjectGridEventGroup interface:

- ShardEvents interface
- ShardLifecycle interface
- TransactionEvents interface

For more information about these interfaces, see the API documentation.

Shard events

When the catalog service places partition primary or replica shards in a Java virtual machine (JVM), a new ObjectGrid instance is created in that JVM to host that shard. Some applications that need to start threads on the JVM host the primary need notification of these events. The ObjectGridEventGroup.ShardEvents interface declares the shardActivate and shardDeactivate methods. These methods are called only when a shard is activated as a primary and when the shard is deactivated from a primary. These two events allow the application to start additional threads when the shard is a primary and stop the threads when the shard returns to being a replica or is taken out of service.

An application can determine which partition has been activated by looking up a specific BackingMap in the ObjectGrid reference that is provided to the shardActivate method using the ObjectGrid#getMap method. The application can then see the partition number using the BackingMap#getPartitionId() method. The partitions are numbered from 0 to the number of partitions in the deployment descriptor minus one.

Shard life-cycle events

ObjectGridEventListener.initialize and ObjectGridEventListener.destroy method events are delivered using the ObjectGridEventGroup.ShardLifecycle interface.

Transaction events

ObjectGridEventListener.transactionBegin and ObjectGridEventListener.transactionEnd methods are delivered through the ObjectGridEventGroup.TransactionEvents interface.
If an ObjectGridEventListener plug-in implements the ObjectGridEventListener and ShardLifecycle interfaces, then shard life-cycle events are the only events that are delivered to the listener. After you implement any of the new ObjectGridEventGroup inner interfaces, eXtreme Scale only delivers those specific events by the new interfaces. With this implementation, code can be backwards compatible. If you are using the new inner interfaces, it can now receive just the specific events that are needed.

Using the ObjectGridEventListener plug-in

To use a custom ObjectGridEventListener plug-in, first create a class that implements the ObjectGridEventListener interface and any optional ObjectGridEventGroup sub-interfaces. Add the custom listener to an ObjectGrid to receive notification of significant events. You have two approaches to add an ObjectGridEventListener plug-in into the eXtreme Scale configuration: programmatic configuration and XML configuration.

Configure an ObjectGridEventListener plug-in programmatically

Assume that the class name of the eXtreme Scale event listener is the com.company.org.MyObjectGridEventListener class. This class implements the ObjectGridEventListener interface. The following code snippet creates the custom ObjectGridEventListener and adds it to an ObjectGrid.

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid", false);
MyObjectGridEventListener myListener = new MyObjectGridEventListener();
myGrid.addEventListener(myListener);
```

Configure an ObjectGridEventListener plug-in with XML

You can also configure an ObjectGridEventListener plug-in using XML. The following XML creates a configuration that is equivalent to the described programmatically created ObjectGrid event listener. The following text must be in the myGrid.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">
            <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsobjectgrideventlistplugin_ObjectGr
idEventListener"
                                                    className="com.company.org.MyObjectGridEventListener" />
            <backingMap name="Book"/>
        </objectGrid>
```

```
    </objectGrids>
</objectGridConfig>
```

Notice the bean declarations come before the backingMap declarations. Provide this file to the ObjectGridManager plug-in to facilitate the creation of this configuration. The following code snippet demonstrates how to create an ObjectGrid instance using this XML file. The ObjectGrid instance that is created has an ObjectGridEventListener listener set on the myGrid ObjectGrid.

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid",
        new URL("file:etc/test/myGrid.xml"), true, false);
```

Related concepts:
Data invalidation
JMS event listener
Related tasks:
8.5+ Querying and invalidating data
Related information:
ObjectMap.invalidate method
EntityManager.invalidate method
ObjectGridEventListener interface

BackingMapLifecycleListener plug-in

A BackingMapLifecycleListener plug-in receives notification of WebSphere® eXtreme Scale life cycle state change events for the backing map.

The BackingMapLifecycleListener plug-in receives an event containing a BackingMapLifecycleListener.State object for each state change of the backing map. Any BackingMap plug-in that also implements the BackingMapLifecycleListener interface will automatically be added as a listener for the BackingMap instance where the plug-in is registered.

Overview

A BackingMapLifecycleListener plug-in is useful when an existing BackingMap plug-in needs to perform activities relative to activities in a related plugin. As an example, a loader plug-in might need to retrieve configuration from a cooperating MapIndexPlugin or DataSerializer plug-in.

By implementing the BackingMapLifecycleListener interface, and detecting the BackingMapLifecycleListener.State.INITIALIZED event, the loader can know about the state of other plug-ins in the BackingMap instance. The loader can safely retrieve information from the cooperating MapIndexPlugin or DataSerializer plug-in, since the BackingMap is in the INITIALIZED state, which means that the other plug-in has had its initialize() method called.

A BackingMapLifecycleListener can be added or removed at any time, either before or after the ObjectGrid and its BackingMaps are initialized.

Write a BackingMapLifecycleListener plug-in

A BackingMapLifecycleListener plug-in must implement the BackingMapLifecycleListener interface to receive notifications about significant eXtreme Scale events. Any BackingMap plug-in can implement the BackingMapLifecycleListener interface and be automatically added as a listener when it is also added to the backing map.

For more information about these interfaces, see the API documentation.

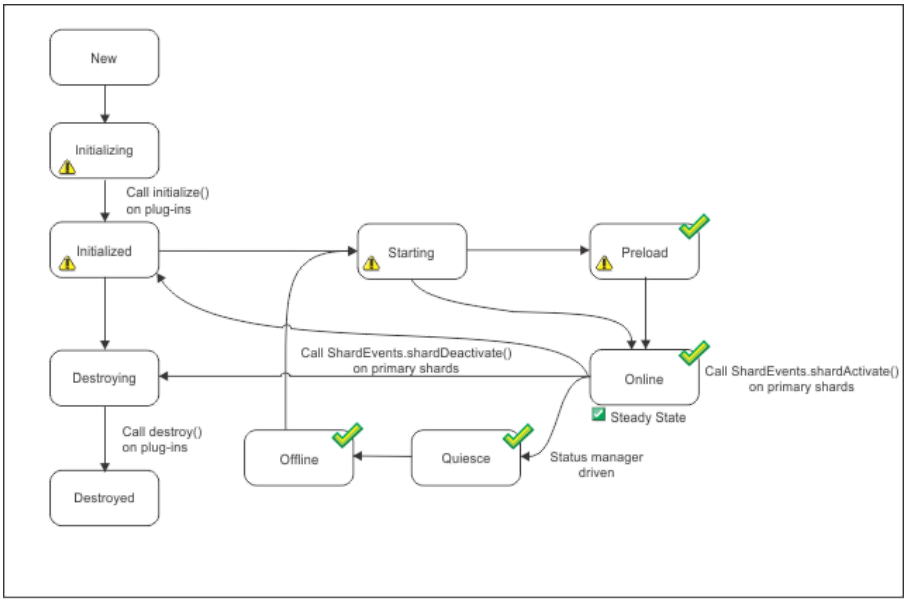
Life cycle event and plug-in relationships

The BackingMapLifecycleListener retrieves the life cycle state from the event in the backingMapStateChanged method; for example:

```
public void backingMapStateChanged(BackingMap map,
                                   LifecycleEvent event)
throws LifecycleFailedException {
  switch(event.getState()) {
    case INITIALIZED: // All other plug-ins are initialized.
      // Retrieve reference to plug-in X for use from map.
      break;
    case DESTROYING:  // Destroy phase is starting
      // Eliminate reference to plug-in X it may be destroyed before this plug-in
      break;
  }
}
```

The following illustration summarizes the states of the BackingMap objects as life cycle events occur and are sent to a BackingMapLifecycleListener plug-in.

Figure 1. BackingMap state summary



The following table describes the relationship between life cycle events sent to a BackingMapLifecycleListener plug-in and the states of the BackingMap and other plug-in objects.

BackingMapLifecycleListener.State value	**Description**
INITIALIZING	The BackingMap initialization phase is starting. The BackingMap and BackingMap plug-ins are about to be initialized.
INITIALIZED	The BackingMap initialization phase is complete. All BackingMap plug-ins are initialized. The INITIALIZED state might recur when shard placement activities (promotion or demotion) occur.
STARTING	The BackingMap instance is being activated for use as a local instance, client instance or as an instance in a primary or replica shard on the server. All ObjectGrid plug-ins in the ObjectGrid instance owning this BackingMap instance have been initialized. The STARTING state might recur when shard placement activities (promotion or demotion) occur.
PRELOAD	The BackingMap instance is set to the PRELOAD state by the StateManager API for preloading, or the configured loader is preloading data into the backing map.
ONLINE	The BackingMap instance is ready for work as a local instance, client instance, or as an instance in a primary or replica shard on the server. All ObjectGrid plug-ins in the ObjectGrid instance owning this BackingMap instance have been initialized. This steady state is typical of the BackingMap. The ONLINE state might recur when shard placement activities (promotion or demotion) occur.
QUIESCE	Work is stopping on the BackingMap as a result of the StateManager API or other event. No new work is allowed. Your plug-in ends any existing work as soon as possible.
OFFLINE	All work is stopped on the BackingMap as a result of the StateManager API or another event. No new work is allowed.
DESTROYING	The BackingMap instance is starting the destroy phase. BackingMap plug-ins for the instance are about to be destroyed.
DESTROYED	The BackingMap instance and all BackingMap plug-ins have been destroyed.

Configure a BackingMapLifecycleListener plug-in with XML

Assume that the class name of the eXtreme Scale event listener is the com.company.org.MyBackingMapLifecycleListener class. This class implements the BackingMapLifecycleListener interface.

You can configure a BackingMapLifecycleListener plug-in using XML. The following text must be in the object grid 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">
            <backingMap name="myMap" pluginCollectionRef="myPlugins" />
        </objectGrid>
    </objectGrids>
    <backingMapPluginCollections>
        <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsconfigbacklife_myPlugins">
            <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsconfigbacklife_BackingMapLifecycle
Listener"
                  className="com.company.org.MyBackingMapLifecycleListener" />
        </backingMapPluginCollection>
```

```
    </backingMapPluginCollections>
</objectGridConfig>
```

Provide this file to the ObjectGridManager plug-in to facilitate the creation of this configuration. The BackingMap instance that is created has a BackingMapLifecycleListener listener set on the myGrid ObjectGrid.

Like the BackingMapLifecycleListener, other BackingMap plug-ins, such as Loader or MapIndexPlugin, that you specify using XML that also implement the BackingMapLifecycleListener interface, will automatically be added as life cycle listeners.

Related reference:
ObjectGridLifecycleListener plug-in

ObjectGridLifecycleListener plug-in

An ObjectGridLifecycleListener plug-in receives notification of WebSphere® eXtreme Scale life cycle, state change events for the data grid.

The ObjectGridLifecycleListener plug-in receives an event containing an ObjectGridLifecycleListener.State object for each state change of the ObjectGrid. Any ObjectGrid plug-in that also implements the ObjectGridLifecycleListener interface will automatically be added as a listener for the ObjectGrid instance where the plug-in is registered.

Overview

An ObjectGridLifecycleListener plug-in is useful when an existing ObjectGrid plug-in needs to perform activities relative to activities in a related plug-in. As an example, a TransactionCallback plug-in might need to retrieve the configuration from a cooperating ObjectGridEventListener or ShardListener plug-in.

By implementing the ObjectGridLifecycleListener interface, and detecting the ObjectGridLifecycleListener.State.INITIALIZED event, the TransactionCallback plug-in can detect the state of other plug-ins in the ObjectGrid instance. The TransactionCallback plug-in can safely retrieve information from the cooperating ObjectGridEventListener plug-in or ShardListener plug-in, since the ObjectGrid is in the INITIALIZED state, which means that the other plug-in has had its initialize() method called.

You can add an ObjectGridLifecycleListener plug-in at any time, either before or after the ObjectGrid is initialized.

Write an ObjectGridLifecycleListener plug-in

An ObjectGridLifecycleListener plug-in must implement the ObjectGridLifecycleListener interface to receive notifications about significant eXtreme Scale events. Any ObjectGrid plug-in can implement the ObjectGridLifecycleListener interface and be automatically added as a listener when it is also added to the ObjectGrid.

For more information about these interfaces, see the API documentation.

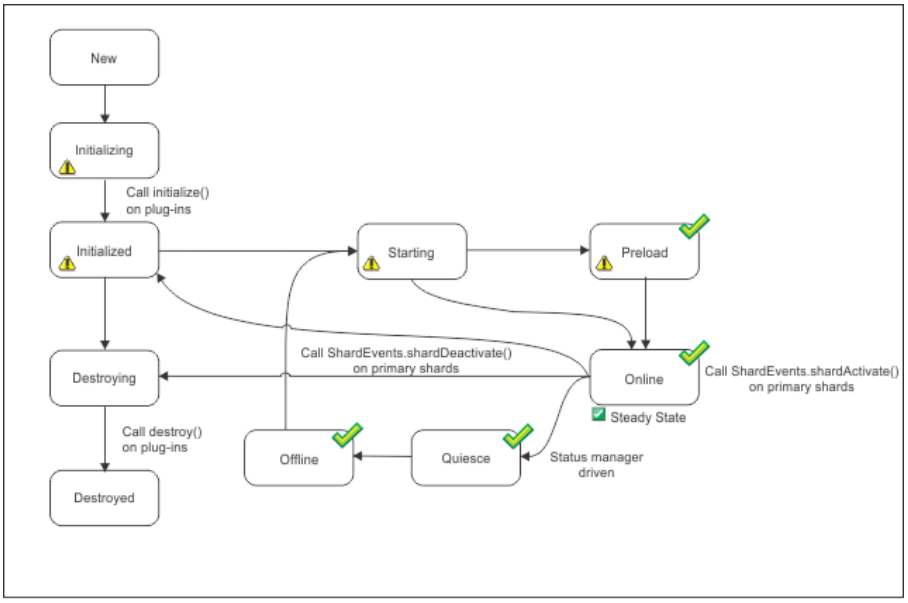
Life cycle event and plug-in relationships

The ObjectGridLifecycleListener retrieves the life cycle state from the event in the objectgridStateChanged method; for example:

```
public void objectGridStateChanged(ObjectGrid grid,
                                   LifecycleEvent event)
throws LifecycleFailedException {
  switch(event.getState()) {
    case INITIALIZED: // All other plug-ins are initialized.
      // Retrieve reference to plug-in X for use from grid.
      break;
    case DESTROYING:  // Destroy phase is starting
      // Eliminate reference to plug-in X it may be destroyed before this plug-in
      break;
  }
```

The following illustration summarizes the states of the ObjectGrid objects as life cycle events occur and are sent to a ObjectGridLifecycleListener plug-in.

Figure 1. ObjectGrid state summary



The following table further describes the relationship between life cycle events sent to a ObjectGridLifecycleListener and the states of the ObjectGrid and other plug-in objects.

ObjectGridLifecycleListener.State value	**Description**
INITIALIZING	The ObjectGrid initialization phase is starting. The ObjectGrid and ObjectGrid plug-ins are about to be initialized.
INITIALIZED	The ObjectGrid initialization phase is complete. All ObjectGrid plug-ins are initialized. The INITIALIZED state might recur when shard placement activities (promotion or demotion) occur. All BackingMap plug-ins in the BackingMap instances owned by this ObjectGrid instance have been initialized.
STARTING	The ObjectGrid instance is being activated for use as a local instance, client instance, or as an instance in a primary or replica shard on the server. The STARTING state might recur when shard placement activities (promotion or demotion) occur.
PRELOAD	The ObjectGrid instance is set to the PRELOAD state by the StateManager API or other configuration.
ONLINE	The ObjectGrid instance is ready for work as a local instance, client instance, or as an instance in a primary or replica shard on the server. This steady state is typical of the ObjectGrid. The ONLINE state might recur when shard placement activities (promotion or demotion) occur.
QUIESCE	Work is stopping on the ObjectGrid as a result of the StateManager API or other event. No new work is allowed. End any existing work as soon as possible.
OFFLINE	All work is stopped on the ObjectGrid as a result of the StateManager API or other event. No new work is allowed.
DESTROYING	The ObjectGrid instance is starting the destroy phase. ObjectGrid plug-ins for the instance are about to be destroyed. During the destroy phase, all BackingMap instances owned by this ObjectGrid instance are also destroyed.
DESTROYED	The ObjectGrid instance, its BackingMap instances, and all ObjectGrid plug-ins have been destroyed.

Configure an ObjectGridLifecycleListener plug-in with XML

Assume that the class name of the eXtreme Scale event listener is the com.company.org.MyObjectGridLifecycleListener class. This class implements the ObjectGridLifecycleListener interface.

You can configure an ObjectGridLifecycleListener plug-in using XML. The following XML creates a configuration using the ObjectGridLifecycleListener. The following text must be in the object grid 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">
            <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsobjgridlifelist_ObjectGridLifecycl
eListener"
                  className="com.company.org.MyObjectGridLifecycleListener" />
            <backingMap name="Book"/>
        </objectGrid>
    </objectGrids>
</objectGridConfig>
```

Notice the bean declarations come before the backingMap declarations. Provide this file to the ObjectGridManager plug-in to facilitate the creation of this configuration.

Like the registered ObjectGridLifecycleListener in the previous example, other ObjectGrid plug-ins, CollisionArbiter or TransactionCallback for example, that you specify using XML that also implement the ObjectGridLifecycleListener interface, will automatically be added as life cycle listeners.

Related reference:
BackingMapLifecycleListener plug-in

Plug-ins for indexing data

Depending on the type of indexes you want to build, WebSphere® eXtreme Scale provides built-in plug-ins that you can add to the BackingMap to build an index.

HashIndex

The built-in HashIndex, the com.ibm.websphere.objectgrid.plugins.index.HashIndex class, is a MapIndexPlugin plug-in that you can add into BackingMap to build static or dynamic indexes. This class supports both the MapIndex and MapRangeIndex interfaces. Defining and implementing indexes can significantly improve query performance.

- Configuring the HashIndex plug-in
  You can configure the built-in HashIndex, the com.ibm.websphere.objectgrid.plugins.index.HashIndex class, with an XML file, programmatically or with an entity annotation on an entity map.

Related concepts:
Plug-ins for custom indexing of cache objects
Using a composite index
Indexing
Related tasks:
Configuring the HashIndex plug-in
Accessing data with indexes (Index API)
Related reference:
HashIndex plug-in attributes

Configuring the HashIndex plug-in

You can configure the built-in HashIndex, the com.ibm.websphere.objectgrid.plugins.index.HashIndex class, with an XML file, programmatically or with an entity annotation on an entity map.

About this task

Configuring a composite index is the same as configuring a regular index with XML, except for the attributeName property value. In a composite index, the value of attributeName property is a comma-delimited list of attributes. For example, the value class `Address` has three attributes: `city`, `state`, and `zipcode`. A composite index can be defined with the attributeName property value as `"city,state,zipcode"` indicating that `city`, `state`, and `zipcode` are included in the composite index.

The composite HashIndexes do not support range lookups and therefore cannot have the RangeIndex property set to true.

Procedure

- Configure a composite index in the ObjectGrid descriptor XML file.
  Use the backingMapPluginCollections element to define the plug-in:

```
<bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txshashconf_MapIndexPlugin"
className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
        <property name="Name" type="java.lang.String" value="Address.CityStateZip"/>
        <property name="AttributeName" type="java.lang.String" value="city,state,zipcode"/>
</bean>
```

- Configure a composite index programmatically.
  The following example code creates the same composite index:

```
          HashIndex mapIndex = new HashIndex();
    mapIndex.setName("Address.CityStateZip");
    mapIndex.setAttributeName(("city,state,zipcode"));
    mapIndex.setRangeIndex(false);

    BackingMap bm = objectGrid.defineMap("mymap");
    bm.addMapIndexPlugin(mapIndex);
```

- Configure a composite index with entity notations.

If you are using entity maps, you can use an annotation approach to define a composite index. You can define a list of CompositeIndex within the CompositeIndexes annotation on the entity class level. The CompositeIndex has a name and attributeNames property. Each CompositeIndex is associated with a HashIndex instance applied to the backing map that is associated with the entity. The HashIndex is configured as a non-range index.

```
@Entity
@CompositeIndexes({
    @CompositeIndex(name="CityStateZip", attributeNames="city,state,zipcode"),
    @CompositeIndex(name="lastnameBirthday", attributeNames="lastname,birthday")
 })
public class Address {
    @Id int id;
    String street;
    String city;
    String state;
    String zipcode;
    String lastname;
    Date birthday;
}
```

The name property for each composite index must be unique within the entity and backing map. If the name is not specified, a generated name is used. The attributeName property is used to populate the HashIndex attributeName with the comma-delimited list of attributes. The attribute names coincide with the persistent field names when the entities are configured to use field-access, or the property name as defined for the JavaBeans naming conventions for property-access entities. For example: If the attribute name is `street`, the property getter method is named getStreet.

Example: Adding a HashIndex class into a BackingMap instance

In the following example, you configure the HashIndex plug-in by adding static index plug-ins to the XML file:

```
<backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txshashconf_person">
   <bean id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txshashconf_MapIndexPlugin"
      className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
         <property name="Name" type="java.lang.String" value="CODE"
            description="index name" />
         <property name="RangeIndex" type="boolean" value="true"
            description="true for MapRangeIndex" />
         <property name="AttributeName" type="java.lang.String" value="employeeCode"
            description="attribute name" />
   </bean>
</backingMapPluginCollection>
```

In this XML configuration example, the built-in HashIndex class is used as the index plug-in. The HashIndex supports properties that users can configure, such as Name, RangeIndex, and AttributeName.

- The Name property is configured as `CODE`, a string that identifies this index plug-in. The Name property value must be unique within the scope of the backing map. The name can be used to retrieve the index object by name from the ObjectMap instance for the BackingMap.
- The RangeIndex property is configured as `true`, which means the application can cast the retrieved index object to the MapRangeIndex interface. If the RangeIndex property is configured as `false`, the application can only cast the retrieved index object to the MapIndex interface. A MapRangeIndex supports functions to find data using range functions such as greater than, less than, or both, while a MapIndex supports equals functions only. If the index is to be used by query, the RangeIndex property must be configured to `true` on single-attribute indexes or `false` on relationship or composite indexes. For a relationship index and composite index, the RangeIndex property must be configured to `false`.
- The AttributeName property is configured as `employeeCode`, which means the employeeCode attribute of the cached object is used to build a single-attribute index. If an application must search for cached objects with multiple attributes, the AttributeName property can be set to a comma-delimited list of attributes, yielding a composite index.

In summary, the previous example defines a single-attribute range HashIndex. It is a single-attribute HashIndex because the AttributeName property value is `employeeCode` that includes only one attribute name. It also is a range HashIndex.

- HashIndex plug-in attributes
  You can use the following attributes to configure the HashIndex plug-in.
- Plug-ins for custom indexing of cache objects
  With a MapIndexPlugin plug-in, or index, you can write custom indexing strategies that are beyond the built-in indexes that eXtreme Scale provides.
- Using a composite index
  The composite HashIndex improves query performance and avoids expensive map scanning. The feature also provides a convenient way for the HashIndex API to find cached objects when search criteria involve many attributes.

Related concepts:
Plug-ins for indexing data
Plug-ins for custom indexing of cache objects
Using a composite index
Indexing
Tuning query performance
Related tasks:
Accessing data with indexes (Index API)
Related reference:
HashIndex plug-in attributes

HashIndex plug-in attributes

You can use the following attributes to configure the HashIndex plug-in.

Attributes

Name
: Specifies the name of the index. The name must be unique for each map. The name is used to retrieve the index object from the object map instance for the backing map.

AttributeName
: Specifies the comma-delimited names of the attributes to index. For field-access indexes, the attribute names are equivalent to the field names. For property-access indexes, the attribute names are the JavaBean compatible property names. If only one attribute name exists, the HashIndex is a single attribute index. If this attribute is a relationship, it is also a relationship index. If multiple attribute names are included in the attribute names, the HashIndex is a composite index.

FieldAccessAttribute
: Used for non-entity maps. If `true`, the object is accessed using the fields directly. If not specified or `false`, the getter method for the attribute is used to access the data.

POJOKeyIndex
: Used for non-entity maps. If `true`, the index introspects the object in the key part of the map. This setting is useful when the key is a composite key and the value does not have the key embedded within it. If not specified or `false`, then the index introspects the object in the value part of the map.

RangeIndex
: If `true`, range indexing is enabled and the application can cast the retrieved index object to the MapRangeIndex interface. If the RangeIndex property is configured as `false`, the application can cast the retrieved index object to the MapIndex interface only.

Single-attribute HashIndex versus composite HashIndex

When the AttributeName property of HashIndex includes multiple attribute names, the HashIndex is a composite index. Otherwise, if it includes only one attribute name, it is a single-attribute index. For example, the AttributeName property value of a composite HashIndex might be `city,state,zipcode`. It includes three attributes delimited by commas. If the AttributeName property value is only `zipcode` that only has one attribute, it is a single-attribute HashIndex.

Composite HashIndex provides an efficient way to look up cached objects when search criteria involve many attributes. However, it does not support range index and its RangeIndex property must set to `false`.
Restriction: A composite index cannot be created if GlobalIndexEnabled is set to true.
For more information, see Using a composite index.

Relationship HashIndex

If the indexed attribute of single-attribute HashIndex is a relationship, either single- or multi-valued, the HashIndex is a relationship HashIndex. For relationship HashIndex, the RangeIndex property of HashIndex must set to “false”.

Relationship HashIndex can speed up queries that use cyclical references or use the IS NULL, IS EMPTY, SIZE, and MEMBER OF query filters. For more information, see Query optimization using indexes.

Key HashIndex

For non-entity maps, when the POJOKeyIndex property of HashIndex is set to `true`, the HashIndex is a key HashIndex and the key part of entry are used for indexing. When the AttributeName property of HashIndex is not specified, the whole key is indexed; otherwise, the key HashIndex can only be a single-attribute HashIndex.

For example, adding the following property into the preceding sample causes the HashIndex to become key HashIndex because the POJOKeyIndex property value is `true`.

```
<property name="POJOKeyIndex" type="boolean" value="true"
description="indicates if POJO key HashIndex" />
```

In the preceding key index example, because the AttributeName property value is specified as `employeeCode`, the indexed attribute is the employeeCode field of the key part of map entry. If you want to build key index on the whole key part of map entry, remove the AttributeName property.

Range HashIndex

When the RangeIndex property of HashIndex is set to `true`, the HashIndex is a range index and can support the MapRangeIndex interface. A MapRangeIndex implementation supports functions to find data using range functions, such as greater than, less than, or both, while a MapIndex supports equals functions only. For a single-attribute index, the RangeIndex property can be set to `true` only if the indexed attribute is of type Comparable. If the single-attribute index will be used by query, the RangeIndex property must set to `true` and the indexed attribute must be of type Comparable. For relationship HashIndex and composite HashIndex, the RangeIndex property must set to `false`.

The preceding sample is a range HashIndex because the RangeIndex property value is `true`.

The following table provides a summary for using range index.

Table 1. Support for range index. States whether HashIndex types support range index.

HashIndex type	Supports range index
Single-attribute HashIndex: indexed key or attribute is of type Comparable	Yes
Single-attribute HashIndex: indexed key or attribute is not of type Comparable	No

HashIndex type	Supports range index
Composite HashIndex	No
Relationship HashIndex	No

Query optimization with HashIndex plug-ins

Defining indexes can significantly improve query performance.WebSphere® eXtreme Scale queries can use built-in HashIndex plug-ins to improve performance of queries. Although using indexes can significantly improve query performance, it might have a performance impact on transactional map operations.

Related concepts:
Plug-ins for indexing data
Plug-ins for custom indexing of cache objects
Using a composite index
Indexing
Tuning query performance
Related tasks:
Configuring the HashIndex plug-in
Accessing data with indexes (Index API)

Plug-ins for custom indexing of cache objects

With a MapIndexPlugin plug-in, or index, you can write custom indexing strategies that are beyond the built-in indexes that eXtreme Scale provides.

MapIndexPlugin implementations must use the MapIndexPlugin interface and follow the common eXtreme Scale plug-in conventions.

The following sections include some of the important methods of the index interface.

setProperties method

Use the setProperties method to initialize the index plug-in programmatically. The Properties object parameter that is passed into the method should contain required configuration information to initialize the index plug-in properly. The setProperties method implementation, along with the getProperties method implementation, are required in a distributed environment because the index plug-in configuration moves between client and server processes. An implementation example of this method follows.

```
setProperties(Properties properties)

// setProperties method sample code
    public void setProperties(Properties properties) {
        ivIndexProperties = properties;

        String ivRangeIndexString = properties.getProperty("rangeIndex");
        if (ivRangeIndexString != null && ivRangeIndexString.equals("true")) {
            setRangeIndex(true);
        }
        setName(properties.getProperty("indexName"));
        setAttributeName(properties.getProperty("attributeName"));

        String ivFieldAccessAttributeString = properties.getProperty("fieldAccessAttribute");
        if (ivFieldAccessAttributeString != null && ivFieldAccessAttributeString.equals("true")) {
            setFieldAccessAttribute(true);
        }

        String ivPOJOKeyIndexString = properties.getProperty("POJOKeyIndex");
        if (ivPOJOKeyIndexString != null && ivPOJOKeyIndexString.equals("true")) {
            setPOJOKeyIndex(true);
        }
    }
```

getProperties method

The getProperties method extracts the index plug-in configuration from a MapIndexPlugin instance. You can use the extracted properties to initialize another MapIndexPlugin instance to have the same internal states. The getProperties method and setProperties method implementations are required in a distributed environment. An implementation example of the getProperties method follows.

```
getProperties()

// getProperties method sample code
    public Properties getProperties() {
        Properties p = new Properties();
        p.put("indexName", indexName);
        p.put("attributeName", attributeName);
        p.put("rangeIndex", ivRangeIndex ? "true" : "false");
        p.put("fieldAccessAttribute", ivFieldAccessAttribute ? "true" : "false");
        p.put("POJOKeyIndex", ivPOJOKeyIndex ? "true" : "false");
        return p;
    }
```

setEntityMetadata method

The setEntityMetadata method is called by the WebSphere® eXtreme Scale run time during initialization to set the EntityMetadata of the associated BackingMap on the MapIndexPlugin instance. The EntityMetadata is required for supporting indexing of tuple objects. A tuple is a data set that represents an entity object or its key. If the BackingMap is for an entity, then you must implement this method.

The following code sample implements the setEntityMetadata method.

```
setEntityMetadata(EntityMetadata entityMetadata)
```

```
// setEntityMetadata method sample code
    public void setEntityMetadata(EntityMetadata entityMetadata) {
        ivEntityMetadata = entityMetadata;
        if (ivEntityMetadata != null) {
            // this is a tuple map
            TupleMetadata valueMetadata = ivEntityMetadata.getValueMetadata();
            int numAttributes = valueMetadata.getNumAttributes();
            for (int i = 0; i < numAttributes; i++) {
                String tupleAttributeName = valueMetadata.getAttribute(i).getName();
                if (attributeName.equals(tupleAttributeName)) {
                    ivTupleValueIndex = i;
                    break;
                }
            }

            if (ivTupleValueIndex == -1) {
                // did not find the attribute in value tuple, try to find it on key tuple.
                // if found on key tuple, implies key indexing on one of tuple key attributes.
                TupleMetadata keyMetadata = ivEntityMetadata.getKeyMetadata();
                numAttributes = keyMetadata.getNumAttributes();
                for (int i = 0; i < numAttributes; i++) {
                    String tupleAttributeName = keyMetadata.getAttribute(i).getName();
                    if (attributeName.equals(tupleAttributeName)) {
                        ivTupleValueIndex = i;
                        ivKeyTupleAttributeIndex = true;
                        break;
                    }
                }
            }

            if (ivTupleValueIndex == -1) {
                // if entityMetadata is not null and we could not find the
                                                    // attributeName in entityMetadata, this is an
                // error
                throw new ObjectGridRuntimeException("Invalid attributeName.  Entity: " +
                                                        ivEntityMetadata.getName());
            }
        }
    }
```

Attribute name methods

The setAttributeName method sets the name of the attribute to be indexed. The cached object class must provide the get method for the indexed attribute. For example, if the object has an employeeName or EmployeeName attribute, the index calls the getEmployeeName method on the object to extract the attribute value. The attribute name must be the same as the name in the get method, and the attribute must implement the Comparable interface. If the attribute is boolean type, you can also use the isAttributeName method pattern.

The getAttributeName method returns the name of the indexed attribute.

getAttribute method

The getAttribute method returns the indexed attribute value from the specified object. For example, if an Employee object has an attribute called employeeName that is indexed, you can use the getAttribute method to extract the employeeName attribute value from a specified Employee object. This method is required in a distributed WebSphere eXtreme Scale environment.

```
getAttribute(Object value)
```

```
// getAttribute method sample code
    public Object getAttribute(Object value) throws ObjectGridRuntimeException {
        if (ivPOJOKeyIndex) {
            // In the POJO key indexing case, no need to get attribute from value object.
            // The key itself is the attribute value used to build the index.
            return null;
        }

        try {
            Object attribute = null;
            if (value != null) {
                // handle Tuple value if ivTupleValueIndex != -1
                if (ivTupleValueIndex == -1) {
                    // regular value
                    if (ivFieldAccessAttribute) {
                        attribute = this.getAttributeField(value).get(value);
                    } else {
                        attribute = getAttributeMethod(value).invoke(value, emptyArray);
```

```
                    }
                } else {
                    // Tuple value
                    attribute = extractValueFromTuple(value);
                }
            }
            return attribute;
        } catch (InvocationTargetException e) {
             throw new ObjectGridRuntimeException(
                    "Caught unexpected Throwable during index update processing,
                                                                                index name = " + indexName + ":
" + t,
                    t);
        } catch (Throwable t) {
             throw new ObjectGridRuntimeException(
                    "Caught unexpected Throwable during index update processing,
                                                                            index name = " + indexName + ": " + t,
                    t);
        }
    }
```

Related concepts:
Plug-ins for indexing data
Using a composite index
Indexing
Related tasks:
Configuring the HashIndex plug-in
Accessing data with indexes (Index API)
Related reference:
HashIndex plug-in attributes

Using a composite index

The composite HashIndex improves query performance and avoids expensive map scanning. The feature also provides a convenient way for the HashIndex API to find cached objects when search criteria involve many attributes.

Improved performance

A composite HashIndex provides a fast and convenient way to search for cached objects with multiple attributes in match-searching criteria. The composite index supports full attribute-match searches, but does not support range searches.

Note: Composite indexes do not support the BETWEEN operator in the ObjectGrid query language because BETWEEN would require range support. The greater than (>) and less than (<) conditionals also do not work because they require range indexes.
A composite index can improve performance of queries if the appropriate composite index is available for the WHERE condition. This means that the composite index has exactly the same attributes as involved in the WHERE condition with full attributes matched.

A query might have many attributes involved in a condition as in the following example.

```
SELECT a FROM Address a WHERE a.city='Rochester' AND a.state='MN' AND a.zipcode='55901'
```

Composite index can improve query performance by avoiding scanning map or joining multiple single-attribute index results. In the example, if a composite index is defined with attributes (city,state,zipcode), the query engine can use the composite index to find the entry with `city='Rochester'`, `state='MN'`, and `zipcode='55901'`. Without composite index and attribute index on city, state, and zipcode attributes, the query engine must scan the map or join multiple single-attribute searches, which usually have expensive overhead. Also, querying for the composite index supports a full-matched pattern only.

Configuring a composite index

You can configure composite indexing in three ways: using XML, programmatically, and with entity annotations only for entity maps.

Programmatic configuration

The following example creates the a composite index.

```
        HashIndex mapIndex = new HashIndex();
    mapIndex.setName("Address.CityStateZip");
    mapIndex.setAttributeName(("city,state,zipcode"));
    mapIndex.setRangeIndex(false);
    BackingMap bm = objectGrid.defineMap("mymap");
    bm.addMapIndexPlugin(mapIndex);
```

Note that configuring a composite index is the same as configuring a regular index with XML except for the attributeName property value. In a composite index case, the value of attributeName is a comma-delimited list of attributes. For example, the value class Address has 3 attributes: city, state, and zipcode. A composite index can be defined with the attributeName property value as `"city,state,zipcode"` indicating that city, state, and zipcode are included in the composite index.

Composite HashIndexes do not support range lookups and therefore cannot have the RangeIndex property set to true.

Using XML

To configure a composite index with XML, include the following configuration in the backingMapPluginCollections element in the ObjectGrid descriptor XML file.

```
Composite index - XML configuration approach
<bean id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxscompindx_MapIndexPlugin"
className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
<property name="Name" type="java.lang.String" value="Address.CityStateZip"/>
<property name="AttributeName" type="java.lang.String" value="city,state,zipcode"/>
</bean>
```

With entity annotations

In the entity map case, annotation approach can be used to define a composite index. You can define a list of CompositeIndex within CompositeIndexes annotation on the entity class level. The CompositeIndex has a name and attributeNames property. Each CompositeIndex is associated with a HashIndex instance applied to the backing map that is associated with the entity. The HashIndex is configured as a non-range index.

```
@Entity
@CompositeIndexes({
    @CompositeIndex(name="CityStateZip", attributeNames="city,state,zipcode"),
    @CompositeIndex(name="lastnameBirthday", attributeNames="lastname,birthday")
 })
public class Address {
    @Id int id;
    String street;
    String city;
    String state;
    String zipcode;
    String lastname;
    Date birthday;
}
```

The name property for each composite index must be unique within the entity and BackingMap. If the name is not specified, a generated name is used. The attributeNames property is used to populate the HashIndex attributeName with the comma-delimited list of attributes. The attribute names coincide with the persistent field names when the entities are configured to use field-access, or the property name as defined for the JavaBeans naming conventions for property-access entities. For example: If the attribute name is "street", the property getter method is named getStreet.

Performing composite index lookups

After a composite index is configured, an application can use the findAll(Object) method of the MapIndex interface to perform lookups.

```
Session sess = objectgrid.getSession();
ObjectMap map = sess.getMap("MAP_NAME");
MapIndex codeIndex = (MapIndex) map.getIndex("INDEX_NAME");
Object[] compositeValue = new Object[]{ MapIndex.EMPTY_VALUE,
                        "MN", "55901"};
Iterator iter = mapIndex.findAll(compositeValue);
// Close the session (optional in Version 7.1.1 and later) for improved performance
sess.close();
```

The MapIndex.EMPTY_VALUE is assigned to the compositeValue[0] which indicates that the city attribute is excluded from evaluation. Only objects with state attribute equal to "MN" and zipcode attribute equal to "55901" are included in the result.

The following queries benefit from the previous composite index configuration:

```
SELECT a FROM Address a WHERE a.city='Rochester' AND a.state='MN' AND a.zipcode='55901'

SELECT a FROM Address a WHERE a.state='MN' AND a.zipcode='55901'
```

The query engine finds the appropriate composite index and use it to improve query performance in full attribute-match cases.

In some scenarios, the application might need to define multiple composite indexes with overlapped attributes in order to satisfy all queries with full attributes matched. A disadvantage of increasing the number of indexes is the possible performance overhead on map operations.

Configuring a global composite index

A global composite index is a type of global index used to index multiple attributes of a cached object which is in the form of a composite index. You can configure a global composite index in two ways, either programmatically or using XML.

Programmatic configuration

The following example creates the a composite index.

```
          HashIndex mapIndex = new HashIndex();
    mapIndex.setName("Address.CityStateZip");
    mapIndex.setAttributeName(("city,state,zipcode"));
    mapIndex.setRangeIndex(false);
    mapIndex.setGlobalIndexEnabled(true);
    BackingMap bm = objectGrid.defineMap("mymap");
    bm.addMapIndexPlugin(mapIndex);
```

Using XML

To configure a global composite index with XML, include the following configuration in the backingMapPluginCollections element in the ObjectGrid descriptor XML file.

```
Global composite index - XML configuration approach
<bean id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxscompindx_MapIndexPlugin"
className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
<property name="Name" type="java.lang.String" value="Address.CityStateZip"/>
<property name="AttributeName" type="java.lang.String" value="city,state,zipcode"/>
<property name="GlobalIndexEnabled" type="boolean" value="true" description="required"/>
</bean>
```

After a global composite index is configured, an application can use the findAll(Object) method of the MapIndex interface to perform lookups.

```
Session sess = objectgrid.getSession();
ObjectMap map = sess.getMap("MAP_NAME");
globalIndex = (MapGlobalIndex)map.getIndex("IndexName");
Object values = new Object[]{ "MN" , "55901" } ;
//  OR :
//  Object[] values =  new Object[] { new Object[] {  "MN" , "55901"  } };
Set keys = mapIndex.findKeys (values);
sess.close();
```

The MapGlobalIndex.EMPTY_VALUE is assigned to the compositeValue[0] which indicates that the city attribute is excluded from evaluation. Only objects with state attribute equal to "MN" and zipcode attribute equal to "55901" are included in the result.

The following queries benefit from the previous composite index configuration:

```
SELECT a FROM Address a WHERE a.city='Rochester' AND a.state='MN' AND a.zipcode='55901'
```

```
SELECT a FROM Address a WHERE a.state='MN' AND a.zipcode='55901'
```

The query engine finds the appropriate composite index and use it to improve query performance in full attribute-match cases.

In some scenarios, the application might need to define multiple composite indexes with overlapped attributes in order to satisfy all queries with full attributes matched. A disadvantage of increasing the number of indexes is the possible performance overhead on map operations.

Migration and interoperability

The only constraint for the use of a composite index is that an application cannot configure it in a distributed environment with heterogeneous containers. Old and new container servers cannot be mixed, since older container servers do not recognize a composite index configuration. The composite index is just like the existing regular attribute index, except that the former allows indexing over multiple attributes. When using only the regular attribute index, a mixed-container environment is still viable.

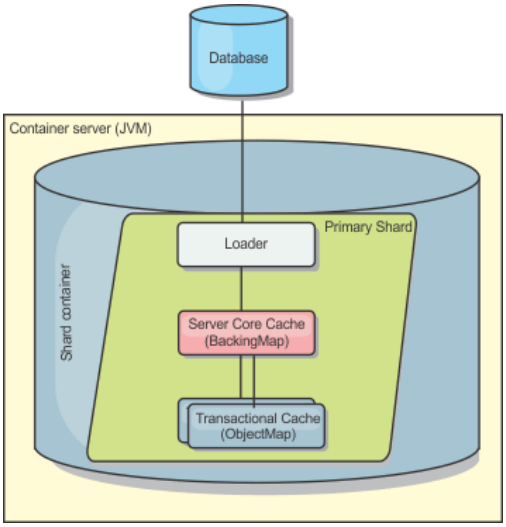
Related concepts:
Plug-ins for indexing data
Plug-ins for custom indexing of cache objects
Indexing
Related tasks:
Configuring the HashIndex plug-in
Accessing data with indexes (Index API)
Related reference:
HashIndex plug-in attributes

Plug-ins for communicating with databases

With a Loader plug-in, an ObjectGrid map can behave as a memory cache for data that is typically kept in a persistent store on either the same system or some other 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 ObjectGrid. A loader has the logic for reading and writing data to and from a persistent store.

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).

Figure 1. Loader



WebSphere® eXtreme Scale includes two built-in loaders to integrate with relational database back ends. The Java Persistence API (JPA) loaders use the Object-Relational Mapping (ORM) capabilities of both the OpenJPA and Hibernate implementations of the JPA specification.

Using a loader

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.

Restriction: BackMaps that are configured with a Loader plug-in can read but cannot write to the map in a multi-partition transaction.

Loaders in multi-master configurations

For considerations about using loaders in multi-master configurations, see Loader considerations in a multi-master topology.

Programmatically plug in a Loader

The following snippet of code demonstrates how to plug an application-provided Loader into the backing map for map1 using the ObjectGrid API:

```
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogManager.createObjectGrid( "grid" );
BackingMap bm = og.defineMap( "map1" );
MyLoader loader = new MyLoader();
loader.setDataBaseName("testdb");
loader.setIsolationLevel("read committed");
bm.setLoader( loader );
```

This snippet assumes that the MyLoader class is the application-provided class that implements the com.ibm.websphere.objectgrid.plugins.Loader interface. Because the association of a Loader with a backing map cannot be changed after ObjectGrid is initialized, the code must be run before invoking the initialize method of the ObjectGrid interface that is being called. An IllegalStateException exception occurs on a setLoader method call if it is called after initialization has occurred.

The application-provided Loader can have set properties. In the example, the MyLoader loader is used to read and write data from a table in a relational database. The loader must specify the name of the database and the SQL isolation level. The MyLoader loader has the setDataBaseName and setIsolationLevel methods that allow the application to set these two Loader properties.

XML configuration approach to plug in a Loader

An application-provided Loader can also be plugged in by using an XML file. The following example demonstrates how to plug the MyLoader loader into the map1 backing map with the same database name and isolation level Loader properties. You must specify the className for your loader, the database name and connection details, and the isolation level properties. You can use the same XML structure if you are only using a preloader by specifying the preloader classname instead of a complete loader classname.:

```
<?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="grid">
        <backingMap name="map1" pluginCollectionRef="map1" lockStrategy="OPTIMISTIC" />
```

```
    </objectGrid>
</objectGrids>
<backingMapPluginCollections>
    <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxsuseloaders_map1">
        <bean id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxsuseloaders_Loader"
className="com.myapplication.MyLoader">
            <property name="dataBaseName"
                      type="java.lang.String"
                      value="testdb"
                      description="database name" />
            <property name="isolationLevel"
                      type="java.lang.String"
                      value="read committed"
                      description="iso level" />
        </bean>
    </backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

- Configuring database loaders
  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).
- Writing a loader
  You can write your own loader plug-in implementation in your applications, which must follow the common WebSphere eXtreme Scale plug-in conventions.
- Map pre-loading
  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 pre-loading 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.
- Configuring write-behind loader support
  You can enable write-behind support with the ObjectGrid descriptor XML file or programmatically with the BackingMap interface.
- JPA loader programming considerations
  A Java Persistence API (JPA) Loader is a loader plug-in implementation that uses JPA to interact with the database. Use the following considerations when you develop an application that uses a JPA loader.
- Using a loader with entity maps and tuples
  The entity manager converts all entity objects into tuple objects before they are stored in an WebSphere eXtreme Scale map. Every entity has a key tuple and a value tuple. This key-value pair is stored in the associated eXtreme Scale map for the entity. When you are using an eXtreme Scale map with a loader, the loader must interact with the tuple objects.
- Writing a loader with a replica preload controller
  A Loader with a replica preload controller is a Loader that implements the ReplicaPreloadController interface in addition to the Loader interface.

Related concepts:
Loader considerations in a multi-master topology
Programming for JPA integration
Database integration: Write-behind, in-line, and side caching
Writing a loader
JPAEntityLoader plug-in
Using a loader with entity maps and tuples
Writing a loader with a replica preload controller
Loaders
Related tasks:
Monitoring eXtreme Scale information in DB2
Related reference:
JPA loader programming considerations

Configuring database loaders

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).

Preload considerations

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). For an overview of how eXtreme Scale interacts with a loader, see In-line cache.

Each backing map has a boolean preloadMode attribute that is set to indicate if preload of a map runs asynchronously. By default, the preloadMode attribute is set to false, which indicates that the backing map initialization does not complete until the preload of the map is complete. For example, backing map initialization is not complete until the preloadMap method returns. If the preloadMap method reads a large amount of data from its back end and loads it into the map, it might take a relatively long time to complete. In this case, you can configure a backing map to use asynchronous preload of the map by setting the preloadMode attribute to true. This setting causes the backing map initialization code to start a thread that invokes the preloadMap method, allowing initialization of a backing map to complete while the preload of the map is still in progress.

In a distributed eXtreme Scale scenario, one of the preload patterns is client preload. In the client preload pattern, an eXtreme Scale client is responsible for retrieving data from the backend and then inserting the data into the distributed container server using DataGrid agents. Furthermore, client preload could be executed in the Loader.preloadMap method in one and only one specific partition. In this case, asynchronously loading the data to the grid becomes very important. If the client preload were executed in the same thread, the backing map would never be initialized, so the partition it resides in would never become ONLINE. Therefore, the eXtreme Scale client could not send the request to the partition, and eventually it would cause an exception.

If an eXtreme Scale client is used in the preloadMap method, you should set the preloadMode attribute to true. The alternative is to start a thread in the client preload code.

The following snippet of code illustrates how the preloadMode attribute is set to enable asynchronous preload:

```
BackingMap bm = og.defineMap( "map1" );
bm.setPreloadMode( true );
```

The preloadMode attribute can also be set by using a XML file as illustrated in the following example:

```
<backingMap name="map1" preloadMode="true" pluginCollectionRef="map1"
        lockStrategy="OPTIMISTIC" />
```

TxID and use of the TransactionCallback interface

Both the get method and batchUpdate methods on the Loader interface are passed a TxID object that represents the Session transaction that requires the get or batchUpdate operation to be performed. The get and batchUpdate methods can be called more than once per transaction. Therefore, transaction-scoped objects that are needed by the Loader are typically kept in a slot of the TxID object. A Java™ database connectivity (JDBC) Loader is used to illustrate how a Loader uses the TxID and TransactionCallback interfaces.

Several ObjectGrid maps can be stored in the same database. Each map has its own loader, and each loader might need to connect to the same database. When the loaders connect to the database, they should use the same JDBC connection. Using the same connection commits the changes to each table as part of the same database transaction. Typically, the same person who writes the Loader implementation also writes the TransactionCallback implementation.

The Loader plug-in can fail when it is unable to communicate to the database back end. This failure can happen if the database server or the network connection is down. The write-behind loader queues the updates and tries to push the data changes to the loader periodically. The loader must notify the ObjectGrid runtime that there is a database connectivity problem by throwing a LoaderNotAvailableException exception. It is not necessary to connect to the database in write-behind mode. However, your TransactionCallback implementation, (*MyTransactionCallback*), must be aware that the database is down and consequently, the database is not run failing database operations. Use the Loader interface and the LoaderNotAvailableException exception to tell WebSphere® eXtreme Scale that the database is down. Do not use your TransactionCallback implementation to communication outages to the server.

A correctly implemented combination of the Loader interface and your TransactionCallback implementation typically does *lazy* instantiation of the SQL connection at the time of the Loader.batchUpdate() call. So instead of programming *MyTransactionCallback*.begin() to create a connection, the Loader.batchUpdate() method calls a private method on the *MyTransactionCallback* object to instantiate the database connection. If this operation fails, then the Loader can immediately throw the LoaderNotAvailableException exception because the database is down. The operations will be intermittently retried until a connection is reestablished.

The best method is when the TransactionCallback interface is extended to add methods that the Loader needs for getting a database connection and for caching prepared statements. The reason for this methodology becomes apparent as you see how the TransactionCallback and TxID interfaces are used by the loader.

As an example, the loader might need the TransactionCallback interface to be extended as follows:

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import com.ibm.websphere.objectgrid.TxID;
public interface MyTransactionCallback extends TransactionCallback
{
    Connection getAutoCommitConnection(TxID tx, String databaseName) throws SQLException;
    Connection getConnection(TxID tx, String databaseName, int isolationLevel ) throws SQLException;
    PreparedStatement getPreparedStatement(TxID tx, Connection conn, String tableName, String sql)
                      throws SQLException;
    Collection getPreparedStatementCollection( TxID tx, Connection conn,        String tableName );
}
```

Using these new methods, the Loader get and batchUpdate methods can get a connection as follows:

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import com.ibm.websphere.objectgrid.TxID;
private Connection getConnection(TxID tx, int isolationLevel)
{
    Connection conn = ivTcb.getConnection(tx, databaseName, isolationLevel );
    return conn;
}
```

In the previous example and in the examples that follow, ivTcb and ivOcb are Loader instance variables that were initialized as described in the Preload considerations section. The ivTcb variable is a reference to the MyTransactionCallback instance and the ivOcb is a reference to the MyOptimisticCallback instance. The databaseName variable is an instance variable of the Loader that was set as a Loader property during the initialization of the backing map. The isolationLevel argument is one of the JDBC Connection constants that are defined for the various isolation levels that JDBC supports. If the Loader is using an optimistic implementation, the get method typically uses a JDBC auto–commit connection to fetch the data from the database. In that case, the Loader might have a getAutoCommitConnection method that is implemented as follows:

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
```

```
import com.ibm.websphere.objectgrid.TxID;
private Connection getAutoCommitConnection(TxID tx)
{
    Connection conn = ivTcb.getAutoCommitConnection(tx, databaseName);
    return conn;
}
```

Recall that the batchUpdate method has the following switch statement:

```
switch ( logElement.getType().getCode() )
{
    case LogElement.CODE_INSERT:
        buildBatchSQLInsert( tx, key, value, conn );
        break;
    case LogElement.CODE_UPDATE:
        buildBatchSQLUpdate( tx, key, value, conn );
        break;
    case LogElement.CODE_DELETE:
        buildBatchSQLDelete( tx, key, conn );
        break;
}
```

Each of the buildBatchSQL methods uses the MyTransactionCallback interface to get a prepared statement. Following is a snippet of code that shows the buildBatchSQLUpdate method building an SQL update statement for updating an EmployeeRecord entry and adding it for the batch update:

```
private void buildBatchSQLUpdate( TxID tx, Object key, Object value,
        Connection conn )
throws SQLException, LoaderException
{
    String sql = "update EMPLOYEE set LASTNAME = ?, FIRSTNAME = ?, DEPTNO = ?,
    SEQNO = ?, MGRNO = ? where EMPNO = ?";
    PreparedStatement sqlUpdate = ivTcb.getPreparedStatement( tx, conn,
                        "employee", sql );
    EmployeeRecord emp = (EmployeeRecord) value;
    sqlUpdate.setString(1, emp.getLastName());
    sqlUpdate.setString(2, emp.getFirstName());
    sqlUpdate.setString(3, emp.getDepartmentName());
    sqlUpdate.setLong(4, emp.getSequenceNumber());
    sqlUpdate.setInt(5, emp.getManagerNumber());
    sqlUpdate.setInt(6, key);
    sqlUpdate.addBatch();
}
```

After the batchUpdate loop has built all of the prepared statements, it calls the getPreparedStatementCollection method. This method is implemented as follows:

```
private Collection getPreparedStatementCollection( TxID tx, Connection conn )
{
    return ( ivTcb.getPreparedStatementCollection( tx, conn, "employee" ) );
}
```

When the application invokes the commit method on the Session, the Session code calls the commit method on the TransactionCallback method after it has pushed all the changes made by the transaction out to the Loader for each map that was changed by the transaction. Because all of the Loaders used the MyTransactionCallback method to get any connection and prepared statements they needed, the TransactionCallback method knows which connection to use to request that the back end commits the changes. So, extending the TransactionCallback interface with methods that are needed by each of the Loaders has the following advantages:

- The TransactionCallback object encapsulates the use of TxID slots for transaction-scoped data, and the Loader does not require information about the TxID slots. The Loader only needs to know about the methods that are added to TransactionCallback using the MyTransactionCallback interface for the supporting functions needed by the Loader.
- The TransactionCallback object can ensure that connection sharing occurs between each Loader that connects to the same backend so that a two phase commit protocol can be avoided.
- The TransactionCallback object can ensure that connecting to the backend is driven to completion through a commit or rollback invoked on the connection when appropriate.
- TransactionCallback ensures that the cleanup of database resources occurs when a transaction completes.
- TransactionCallback hides if it is obtaining a managed connection from a managed environment such as WebSphere Application Server or some other Java 2 Platform, Enterprise Edition (J2EE) compliant application server. This advantage allows the same Loader code to be used in both a managed and unmanaged environments. Only the TransactionCallback plug-in must be changed.
- For detailed information about how the TransactionCallback implementation uses the TxID slots for transaction-scoped data, see TransactionCallback plug-in.

OptimisticCallback

As mentioned earlier, the Loader might use an optimistic approach for concurrency control. In this case, the buildBatchSQLUpdate method example must be modified slightly for implementing an optimistic approach. Several possible ways exist for using an optimistic approach. A typical way is to have either a timestamp column or sequence number counter column for versioning each update of the row. Assume that the employee table has a sequence number column that increments each time the row is updated. You then modify the signature of the buildBatchSQLUpdate method so that it is passed the LogElement object instead of the key and value pair. It also needs to use the OptimisticCallback object that is plugged into the backing map for getting both the initial version object and for updating the version object. The following is an example of a modified buildBatchSQLUpdate method that uses the ivOcb instance variable that was initialized as described in the preloadMap section:

```
modified batch-update method code example
private void buildBatchSQLUpdate( TxID tx, LogElement le, Connection conn )
        throws SQLException, LoaderException
```

```
{
    // Get the initial version object when this map entry was last read
    // or updated in the database.
    Employee emp = (Employee) le.getCurrentValue();
    long initialVersion = ((Long) le.getVersionedValue()).longValue();
    // Get the version object from the updated Employee for the SQL update
    //operation.
    Long currentVersion = (Long)ivOcb.getVersionedObjectForValue( emp );
    long nextVersion = currentVersion.longValue();
    // Now build SQL update that includes the version object in where clause
    // for optimistic checking.
    String sql = "update EMPLOYEE set LASTNAME = ?, FIRSTNAME = ?,
    DEPTNO = ?,SEQNO = ?, MGRNO = ? where EMPNO = ? and SEQNO = ?";
    PreparedStatement sqlUpdate = ivTcb.getPreparedStatement( tx, conn,
                        "employee", sql );
    sqlUpdate.setString(1, emp.getLastName());
    sqlUpdate.setString(2, emp.getFirstName());
    sqlUpdate.setString(3, emp.getDepartmentName());
    sqlUpdate.setLong(4, nextVersion );
    sqlUpdate.setInt(5, emp.getManagerNumber());
    sqlUpdate.setInt(6, key);
    sqlUpdate.setLong(7, initialVersion);
    sqlUpdate.addBatch();
}
```

The example shows that the LogElement is used to obtain the initial version value. When the transaction first accesses the map entry, a LogElement is created with the initial Employee object that is obtained from the map. The initial Employee object is also passed to the getVersionedObjectForValue method on the OptimisticCallback interface and the result is saved in the LogElement. This processing occurs before an application is given a reference to the initial Employee object and has a chance to call some method that changes the state of the initial Employee object.

The example shows that the Loader uses the getVersiondObjectForValue method to obtain the version object for the current updated Employee object. Before calling the batchUpdate method on the Loader interface, eXtreme Scale calls the updateVersionedObjectForValue method on the OptimisticCallback interface to cause a new version object to be generated for the updated Employee object. After the batchUpdate method returns to the ObjectGrid, the LogElement is updated with the current version object and becomes the new initial version object. This step is necessary because the application might have called the flush method on the map instead of the commit method on the Session. It is possible for the Loader to be called multiple times by a single transaction for the same key. For that reason, eXtreme Scale ensures that the LogElement is updated with the new version object each time the row is updated in the employee table.

Now that the Loader has both the initial version object and the next version object, it can run an SQL update statement that sets the SEQNO column to the next version object value and uses the initial version object value in the where clause. This approach is sometimes referred to as an overqualified update statement. The use of the overqualified update statement allows the relational database to verify that the row was not changed by some other transaction between the time that this transaction read the data from the database and the time that this transaction updates the database. If another transaction modified the row, then the count array that is returned by the batch update indicates that zero rows were updated for this key. The Loader is responsible for verifying that the SQL update operation did update the row. If it does not, the Loader displays a com.ibm.websphere.objectgrid.plugins.OptimisticCollisionException exception to inform the Session that the batchUpdate method failed due to more than one concurrent transaction trying to update the same row in the database table. This exception causes the Session to roll back and the application must retry the entire transaction. The rationale is that the retry will be successful, which is why this approach is called optimistic. The optimistic approach performs better if data is infrequently changed or concurrent transactions rarely try to update the same row.

It is important for the Loader to use the key parameter of the OptimisticCollisionException constructor to identify which key or set of keys caused the optimistic batchUpdate method to fail. The key parameter can either be the key object itself or an array of key objects if more than one key resulted in optimistic update failure. And eXtreme Scale uses the getKey method of the OptimisticCollisionException constructor to determine which map entries contain stale data and caused the exception to result. Part of the rollback processing is to evict each stale map entry from the map. Evicting stale entries is necessary so that any subsequent transaction that accesses the same key or keys results in the get method of the Loader interface being called to refresh the map entries with the current data from the database.

Other ways for a Loader to implement an optimistic approach include:

- No timestamp or sequence number column exists. In this case, the getVersionObjectForValue method on the OptimisticCallback interface simply returns the value object itself as the version. With this approach, the Loader needs to build a where clause that includes each of the fields of the initial version object. This approach is not efficient, and not all column types are eligible to be used in the where clause of an overqualified SQL update statement. This approach is typically not used.
- No timestamp or sequence number column exists. However, unlike the prior approach, the where clause only contains the value fields that were modified by the transaction. One method to detect which fields are modified is to set the copy mode on the backing map to be CopyMode.COPY_ON_WRITE mode. This copy mode requires that a value interface be passed to the setCopyMode method on the BackingMap interface. The BackingMap creates dynamic proxy objects that implement the provided value interface. With this copy mode, the Loader can cast each value to a com.ibm.websphere.objectgrid.plugins.ValueProxyInfo object. The ValueProxyInfo interface has a method that allows the Loader to obtain the List of attribute names that were changed by the transaction. This method enables the Loader to call the get methods on the value interface for the attribute names to obtain the changed data and to build an SQL update statement that only sets the changed attributes. The where clause can now be built to have the primary key column plus each of the changed attribute columns. This approach is more efficient than the prior approach, but it requires more code to be written in the Loader and leads to the possibility that the prepared statement cache needs to be larger to handle the different permutations. However, if transactions typically only modify a few of the attributes, this limitation might not be a problem.
- Some relational databases might have an API to assist in automatically maintaining column data that is useful for optimistic versioning. Consult your database documentation to determine if this possibility exists.

Writing a loader

You can write your own loader plug-in implementation in your applications, which must follow the common WebSphere® eXtreme Scale plug-in conventions.

Including a loader plug-in

The Loader interface has the following definition:

```
public interface Loader
{
    static final SpecialValue KEY_NOT_FOUND;
    List get(TxID txid, List keyList, boolean forUpdate) throws LoaderException;
    void batchUpdate(TxID txid, LogSequence sequence) throws
                              LoaderException, OptimisticCollisionException;
    void preloadMap(Session session, BackingMap backingMap) throws LoaderException;
}
```

See Loaders for more information.

get method

The backing map calls the Loader get method to get the values that are associated with a key list that is passed as the keyList argument. The get method is required to return a java.lang.util.List list of values, one value for each key that is in the key list. The first value that is returned in the value list corresponds to the first key in the key list, the second value returned in the value list corresponds to the second key in the key list, and so on. If the loader does not find the value for a key in the key list, the Loader is required to return the special KEY_NOT_FOUND value object that is defined in the Loader interface. Because a backing map can be configured to allow `null` as a valid value, it is very important for the Loader to return the special KEY_NOT_FOUND object when the Loader cannot find the key. This special value allows the backing map to distinguish between a null value and a value that does not exist because the key was not found. If a backing map does not support null values, a Loader that returns a null value instead of the KEY_NOT_FOUND object for a key that does not exist results in an exception.

The forUpdate argument tells the Loader if the application called a get method on the map or a getForUpdate method on the map. See the ObjectMap interface for more information. The Loader is responsible for implementing a concurrency control policy that controls concurrent access to the persistent store. For example, many relational database management systems support the for update syntax on the SQL select statement that is used to read data from a relational table. The Loader can choose to use the for update syntax on the SQL select statement based on whether `boolean true` is passed as the argument value for the forUpdate parameter of this method. Typically, the Loader uses the for update syntax only when the pessimistic concurrency control policy is used. For an optimistic concurrency control, the Loader never uses `for update` syntax on the SQL select statement. The Loader is responsible to decide to use the forUpdate argument based on the concurrency control policy that is being used by the Loader.

For an explanation of the txid parameter, see Plug-ins for managing transaction life cycle events.

batchUpdate method

The batchUpdate method is important on the Loader interface. This method is called whenever the eXtreme Scale needs to apply all the current changes to the Loader. The Loader is given a list of changes for the selected Map. The changes are iterated and applied to the backend. The method receives the current TxID value and the changes to apply. The following sample iterates over the set of changes and batches three Java database connectivity (JDBC) statements, one with insert, another with update, and one with delete.

```
import java.util.Collection;
import java.util.Map;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import com.ibm.websphere.objectgrid.TxID;
import com.ibm.websphere.objectgrid.plugins.Loader;
import com.ibm.websphere.objectgrid.plugins.LoaderException;
import com.ibm.websphere.objectgrid.plugins.LogElement;
import com.ibm.websphere.objectgrid.plugins.LogSequence;

    public void batchUpdate(TxID tx, LogSequence sequence) throws LoaderException {
        // Get a SQL connection to use.
        Connection conn = getConnection(tx);
        try {
            // Process the list of changes and build a set of prepared
            // statements for executing a batch update, insert, or delete
            // SQL operation.
            Iterator iter = sequence.getPendingChanges();
            while (iter.hasNext()) {
                LogElement logElement = (LogElement) iter.next();
                Object key = logElement.getKey();
                Object value = logElement.getCurrentValue();
                switch (logElement.getType().getCode()) {
                case LogElement.CODE_INSERT:
                    buildBatchSQLInsert(tx, key, value, conn);
                    break;
                case LogElement.CODE_UPDATE:
                    buildBatchSQLUpdate(tx, key, value, conn);
                    break;
                case LogElement.CODE_DELETE:
                    buildBatchSQLDelete(tx, key, conn);
                    break;
                }
            }
            // Execute the batch statements that were built by above loop.
            Collection statements = getPreparedStatementCollection(tx, conn);
            iter = statements.iterator();
            while (iter.hasNext()) {
                PreparedStatement pstmt = (PreparedStatement) iter.next();
                pstmt.executeBatch();
            }
```

```
        } catch (SQLException e) {
            LoaderException ex = new LoaderException(e);
            throw ex;
        }
    }
```

The preceding sample illustrates the high level logic of processing the LogSequence argument, but the details of how a SQL insert, update, or delete statement is built are not illustrated. Some of the key points that are illustrated include:

- The getPendingChanges method is called on the LogSequence argument to obtain an iterator over the list of LogElements that the Loader needs to process.
- The LogElement.getType().getCode() method is used to determine if the LogElement is for a SQL insert, update, or delete operation.
- An SQLException exception is caught and is chained to a LoaderException exception that prints to report that an exception occurred during the batch update.
- JDBC batch update support is used to minimize the number of queries to the backend that must be made.

preloadMap method

During the eXtreme Scale initialization, each backing map that is defined is initialized. If a Loader is plugged into a backing map, the backing map invokes the preloadMap method on the Loader interface to allow the loader to prefetch data from its back end and load the data into the map. The following sample assumes the first 100 rows of an Employee table is read from the database and is loaded into the map. The EmployeeRecord class is an application-provided class that holds the employee data that is read from the employee table.

Note: This sample fetches all the data from database and then insert them into the base map of one partition. In a real-world distributed eXtreme Scale deployment scenario, data should be distributed into all the partitions. Refer to Developing client-based JPA loaders for more information.

```
import java.sql.PreparedStatement;
import java.sql.SQLException;
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.TxID;
import com.ibm.websphere.objectgrid.plugins.Loader;
import com.ibm.websphere.objectgrid.plugins.LoaderException
```

```
    public void preloadMap(Session session, BackingMap backingMap) throws LoaderException {
        boolean tranActive = false;
        ResultSet results = null;
        Statement stmt = null;
        Connection conn = null;
        try {
            session.beginNoWriteThrough();
            tranActive = true;
            ObjectMap map = session.getMap(backingMap.getName());
            TxID tx = session.getTxID();
            // Get a auto-commit connection to use that is set to
            // a read committed isolation level.
            conn = getAutoCommitConnection(tx);
            // Preload the Employee Map with EmployeeRecord
            // objects. Read all Employees from table, but
            // limit preload to first 100 rows.
            stmt = conn.createStatement();
            results = stmt.executeQuery(SELECT_ALL);
            int rows = 0;
            while (results.next() && rows < 100) {
                int key = results.getInt(EMPNO_INDEX);
                EmployeeRecord emp = new EmployeeRecord(key);
                emp.setLastName(results.getString(LASTNAME_INDEX));
                emp.setFirstName(results.getString(FIRSTNAME_INDEX));
                emp.setDepartmentName(results.getString(DEPTNAME_INDEX));
                emp.updateSequenceNumber(results.getLong(SEQNO_INDEX));
                emp.setManagerNumber(results.getInt(MGRNO_INDEX));
                map.put(new Integer(key), emp);
                ++rows;
            }
            // Commit the transaction.
            session.commit();
            tranActive = false;
        } catch (Throwable t) {
            throw new LoaderException("preload failure: " + t, t);
        } finally {
            if (tranActive) {
                try {
                    session.rollback();
                } catch (Throwable t2) {
                    // Tolerate any rollback failures and
                    // allow original Throwable to be thrown.
                }
            }
            // Be sure to clean up other databases resources here
            // as well such a closing statements, result sets, etc.
        }
    }
```

This sample illustrates the following key points:

- The preloadMap backing map uses the Session object that is passed to it as the session argument.
- The Session.beginNoWriteThrough method is used to begin the transaction instead of the begin method.

- The Loader cannot be called for each put operation that occurs in this method for loading the map.
- The Loader can map columns of the employee table to a field in the EmployeeRecord Java object. The Loader catches all throwable exceptions that occur and throws a LoaderException exception with the caught throwable exception chained to it.
- The finally block ensures that any throwable exception that occurs between the time the beginNoWriteThrough method is called and the commit method is called cause the finally block to roll back the active transaction. This action is critical to ensure that any transaction that has been started by the preloadMap method is completed before returning to the caller. The finally block is a good place to perform other cleanup actions that might be needed, like closing the JDBC connection and other JDBC objects.

The preloadMap sample is using a SQL select statement that selects all rows of the table. In your application-provided Loader, you might need to set one or more Loader properties to control how much of the table needs to be preloaded into the map.

Because the preloadMap method is only called one time during the BackingMap initialization, it is also a good place to run the one time Loader initialization code. Even if a Loader chooses not to prefetch data from the backend and load the data into the map, it probably needs to perform some other one time initialization to make other methods of the Loader more efficient. The following example illustrates caching the TransactionCallback object and OptimisticCallback object as instance variables of the Loader so that the other methods of the Loader do not have to make method calls to get access to these objects. This caching of the ObjectGrid plug-in values can be performed because after the BackingMap is initialized, the TransactionCallback and the OptimisticCallback objects cannot be changed or replaced. It is acceptable to cache these object references as instance variables of the Loader.

```
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.plugins.OptimisticCallback;
import com.ibm.websphere.objectgrid.plugins.TransactionCallback;

    // Loader instance variables.
    MyTransactionCallback ivTcb; // MyTransactionCallback

    // extends TransactionCallback
    MyOptimisticCallback ivOcb; // MyOptimisticCallback

    // implements OptimisticCallback
    // ...
    public void preloadMap(Session session, BackingMap backingMap) throws LoaderException
                  [Replication programming]
        // Cache TransactionCallback and OptimisticCallback objects
        // in instance variables of this Loader.
        ivTcb = (MyTransactionCallback) session.getObjectGrid().getTransactionCallback();
        ivOcb = (MyOptimisticCallback) backingMap.getOptimisticCallback();
        // The remainder of preloadMap code (such as shown in prior example).
    }
```

For information about preloading and recoverable preloading as it pertains to replication failover, see Replication for availabilitythe information about replication in the *Product Overview*.

Loaders with entity maps

If the loader is plugged into an entity map, the loader must handle tuple objects. Tuple objects are a special entity data format. The loader must conversion data between tuple and other data formats. For example, the get method returns a list of values that correspond to the set of keys that are passed in to the method. The passed-in keys are in the type of Tuple, says key tuples. Assuming that the loader persists the map with a database using JDBC, the get method must convert each key tuple into a list of attribute values that correspond to the primary key columns of the table that is mapped to the entity map, run the SELECT statement with the WHERE clause that uses converted attribute values as criteria to fetch data from database, and then convert the returned data into value tuples. The get method gets data from the database and converts the data into value tuples for passed-in key tuples, and then returns a list of value tuples correspond to the set of tuple keys that are passed in to the caller. The get method can perform one SELECT statement to fetch all data at one time, or run a SELECT statement for each key tuple. For programming details that show how to use the Loader when the data is store using an entity manager, see Using a loader with entity maps and tuples.

Related concepts:
Plug-ins for communicating with databases
JPAEntityLoader plug-in
Using a loader with entity maps and tuples
Writing a loader with a replica preload controller
Loaders
Related reference:
JPA loader programming considerations

Map pre-loading

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 pre-loading 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;
```

Pre-loading 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 a data grid because the application is no longer limited by the heap size of a single Java™ virtual machine (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 how 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.

See Writing a loader with a replica preload controller for an example on how to implement a Loader for a replicated data grid.

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.

Pre-loading map set

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 data grid. It might be more efficient to preload all the maps by designating a single map as the pre-loading 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 pre-loading

Some customers have very large data sets that need caching. Pre-loading this data can be very time consuming. Sometimes, the pre-loading must complete before the application can go online. You can benefit from making pre-loading recoverable. Suppose there are a million records to preload. The primary is pre-loading 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 1. 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 pre-loading from that point onwards.

Clearly, while a primary is pre-loading the map, it must leave some state in a map in the MapSet that is being replicated so that the replica determines what status to return. You can use an extra map named, for example, RecoveryMap map. This RecoveryMap map must be part of the same MapSet 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 map as part of that transaction. The preloaded data and the RecoveryMap map data are replicated atomically to the replicas. When the replica is promoted to primary, it can now check the RecoveryMap map to see what has happened.

The RecoveryMap map 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 map 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 receives 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 2. Commit sequence on the primary

Step with loader	Step without loader

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 acknowledgement	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.

```
CommitedTx = {}, RolledBackTx = {}
```

During synchronous commit processing, use the following sequence:

Table 3. 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 acknowledgement	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 change data. 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 time 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.

Configuring write-behind loader support

You can enable write-behind support with the ObjectGrid descriptor XML file or programmatically with the BackingMap interface.

Use either the ObjectGrid descriptor XML file to enable write-behind support, or programmatically by using the BackingMap interface.

ObjectGrid descriptor XML file

When you configure an ObjectGrid with an ObjectGrid descriptor XML file, the write-behind loader is enabled by setting the writeBehind attribute on the backingMap tag. An example follows:

```
<objectGrid name="library" >
     <backingMap name="book" writeBehind="T300;C900" pluginCollectionRef="bookPlugins"/>
```

In the previous example, write-behind support of the book backing map is enabled with parameter T300;C900. The write-behind attribute specifies the maximum update time, maximum key update count, or both. The format of the write-behind parameter is:

```
write-behind attribute ::=<defaults> | <update time> | <update key count> | <update time> ";" <update key count>

update time ::= "T" <positive integer>
update key count ::= "C" <positive integer>
defaults ::= "" {table}
```

Updates to the loader occur when one of the following events occurs:

1. The maximum update time in seconds has elapsed since the last update.
2. The number of updated keys in the queue map has reached the update key count.

These parameters are only hints. The real update count and update time will be within close range of the parameters. However, we do not guarantee that the actual update count or update time are the same as defined in the parameters. Also, to prevent all partitions from accessing the database simultaneously, the ObjectGrid randomizes the update starting time. For example, the first update can occur after up to twice as long as the real update time.

In the previous example T300;C900, the loader writes the data to the back-end when 300 seconds have passed since the last update or when 900 keys are pending to be updated. The default update time is 300 seconds and the default update key count is 1000.

Table 1. Some write-behind options

Attribute value	Time
T100	The update time is 100 seconds, and the update key count is 1000 (the default value)
C2000	The update time is 300 seconds (the default value), and the update key count is 2000.
T300;C900	The update time is 300 seconds and the update key count is 900.
""	The update time is 300 second (the default value), and the update key count is 1000 (the default value). Note: If you configure the write-behind loader as an empty string: `writeBehind=""`, the write-behind loader is enabled with the default values. Therefore, do not specify the writeBehind attribute if you do not want write-behind support enabled.

Programmatically enabling write-behind support

When you are creating a backing map programmatically for a local, in-memory eXtreme Scale, you can use the following method on the BackingMap interface to enable and disable write-behind support.

```
public void setWriteBehind(String writeBehindParam);
```

For more details about how to use the setWriteBehind method, see BackingMap interface.

- Write-behind caching
  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 loader application design considerations
  When you implement a write-behind loaders, you must consider several issues such as integrity constraints, locking behavior, and performance.
- Handling failed write-behind updates
  Because the WebSphere® eXtreme Scale transaction finishes before the back-end transaction starts, the transaction could have false success.
- Example: Writing a write-behind dumper class
  This sample source code shows how to write a watcher (dumper) to handle failed write-behind updates.

Related reference:
Example: Writing a write-behind dumper class

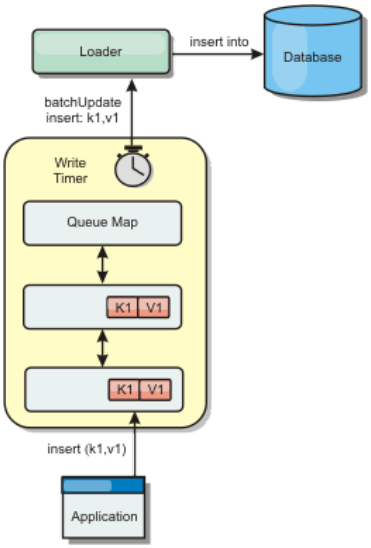
Write-behind caching

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 1. 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.

Related concepts:
Write-behind loader application design considerations
Handling failed write-behind updates
Related reference:
Example: Writing a write-behind dumper class
Example: Writing a write-behind dumper class

Write-behind loader application design considerations

When you implement a write-behind loaders, you must consider several issues such as integrity constraints, locking behavior, and performance.

Application design considerations

Enabling write-behind support is simple, but designing an application to work with write-behind support needs careful consideration. Without write-behind support, the ObjectGrid transaction encloses the back-end transaction. The ObjectGrid transaction starts before the back-end transaction starts, and it ends after the back-end transaction ends.

With write-behind support enabled, the ObjectGrid transaction finishes before the back-end transaction starts. The ObjectGrid transaction and back-end transaction are de-coupled.

Referential integrity constraints

Each backing map that is configured with write-behind support has its own write-behind thread to push the data to the back-end. Therefore, the data that updated to different maps in one ObjectGrid transaction are updated to the back-end in different back-end transactions. For example, transaction T1 updates key key1 in map Map1 and key key2 in map Map2. The key1 update to map Map1 is updated to the back-end in one back-end transaction, and the key2 updated to map Map2 is updated to the back-end in another back-end transaction by different write-behind threads. If data stored in Map1 and Map2 have relations, such as foreign key constraints in the back-end, the updates might fail.

When designing the referential integrity constraints in your back-end database, ensure that out-of-order updates are allowed.

Queue map locking behavior

Another major transaction behavior difference is the locking behavior. ObjectGrid supports three different locking strategies: PESSIMISTIC, OPTIMISITIC, and NONE. The write-behind queue maps uses pessimistic locking strategy no matter which lock strategy is configured for its backing map. Two different types of operations exist that acquire a lock on the queue map:

- When an ObjectGrid transaction commits, or a flush (map flush or session flush) happens, the transaction reads the key in the queue map and places an S lock on the key.
- When an ObjectGrid transaction commits, the transaction tries to upgrade the S lock to X lock on the key.

Because of this extra queue map behavior, you can see some locking behavior differences.

- If the user map is configured as PESSIMISTIC locking strategy, there isn't much locking behavior difference. Every time a flush or commit is called, an S lock is placed on the same key in the queue map. During the commit time, not only is an X lock acquired for key in the user map, it is also acquired for the key in the queue map.
- If the user map is configured as OPTIMISTIC or NONE locking strategy, the user transaction will follow the PESSIMISTIC locking strategy pattern. Every time a flush or commit is called, an S lock is acquired for the same key in the queue map. During the commit time, an X lock is acquired for the key in the queue map using the same transaction.

Loader transaction retries

ObjectGrid does not support 2-phase or XA transactions. The write-behind thread removes records from the queue map and updates the records to the back-end. If the server fails in the middle of the transaction, some back-end updates can be lost.

The write-behind loader will automatically retry to write failed transactions and will send an in-doubt LogSequence to the back-end to avoid data loss. This action requires the loader to be idempotent, which means when the Loader.batchUpdate(TxId, LogSequence) is called twice with the same value, it gives the same result as if it were applied one time. Loader implementations must implement the RetryableLoader interface to enable this feature. See the API documentation for more details.

Write-behind caching performance considerations

Write-behind caching support improves response time by removing the loader update from the transaction. It also increases database throughput because database updates are combined. It is important to understand the overhead introduced by write-behind thread, which pulls the data out of the queue map and pushed to the loader.

The maximum update count or the maximum update time need to be adjusted based on the expected usage patterns and environment. If the value of the maximum update count or the maximum update time is too small, the overhead of the write-behind thread may exceed the benefits. Setting a large value for these two parameters could also increase the memory usage for queuing the data and increase the stale time of the database records.

For best performance, tune the write-behind parameters based on the following factors:

- Ratio of read and write transactions
- Same record update frequency
- Database update latency.

Related concepts:
Write-behind caching
Handling failed write-behind updates
Related reference:
Example: Writing a write-behind dumper class

Handling failed write-behind updates

Because the WebSphere® eXtreme Scale transaction finishes before the back-end transaction starts, the transaction could have false success.

If you try to insert an entry in an eXtreme Scale transaction that does not exist in the backing map but exists in the backend database, causing a duplicate key, the eXtreme Scale transaction succeeds. However, the transaction in which the write-behind thread inserts the object into the backend database fails with a duplicate key exception.

Handling failed write-behind updates: client side

Such an update, or any other failed back-end update, is a failed write-behind update. Failed write-behind updates are stored in a failed write-behind update map. This map serves as an event queue for failed updates. The key of the update is a unique Integer object, and the value is an instance of FailedUpdateElement. The failed write-behind update map is configured with an evictor, which evicts the records one hour after it has been inserted. So the failed-update records are lost if they are not retrieved within one hour.

The ObjectMap API can be used to retrieve the failed write-behind update map entries. The failed write-behind update map name is: `IBM_WB_FAILED_UPDATES_<map name>`. See the WriteBehindLoaderConstants API documentation for the prefix names of each of the write-behind system maps. The following is an example.

```
process failed - example code
ObjectMap failedMap = session.getMap(
    WriteBehindLoaderConstants.WRITE_BEHIND_FAILED_UPDATES_MAP_PREFIX + "Employee");
Object key = null;

session.begin();
while(key = failedMap.getNextKey(ObjectMap.QUEUE_TIMEOUT_NONE)) {
    FailedUpdateElement element = (FailedUpdateElement) failedMap.get(key);
    Throwable throwable = element.getThrowable();
    Object failedKey = element.getKey();
    Object failedValue = element.getAfterImage();
    failedMap.remove(key);
    // Do something interesting with the key, value, or exception.

}
session.commit();
```

A getNextKey method call works with a specific partition for each eXtreme Scale transaction. In a distributed environment, to get keys from all partitions, you must start multiple transactions, as shown in the following example:

```
getting keys from all partitions - example code
ObjectMap failedMap = session.getMap(
    WriteBehindLoaderConstants.WRITE_BEHIND_FAILED_UPDATES_MAP_PREFIX + "Employee");
while (true) {
    session.begin();
    Object key = null;
    while(( key = failedMap.getNextKey(5000) )!= null ) {
        FailedUpdateElement element = (FailedUpdateElement) failedMap.get(key);
    Throwable throwable = element.getThrowable();
        Object failedKey = element.getKey();
        Object failedValue = element.getAfterImage();
        failedMap.remove(key);
        // Do something interesting with the key, value, or exception.

    }
    Session.commit();
}
```

Note: The failed update map provides a way to monitor the application health. If a system produces many records in the failed update map, it is a sign that you need to revise the application or architecture to use the write-behind support. You can use the **xscmd -showMapSizes** command to see the failed update map entry size.

Handling failed write-behind updates: shard listener

It is important to detect and log when a write-behind transaction fails. Every application using write-behind needs to implement a watcher to handle failed write-behind updates. This avoids potentially running out of memory as records in the bad update Map are not evicted because the application is expected to handle them.

The following code shows how to plug in such a watcher, or "dumper," which must be added to the ObjectGrid descriptor XML as in the snippet.

```
        <objectGrid name="Grid">
                <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxsfailupdate_ObjectGridEventListener
" className="utils.WriteBehindDumper"/>
```

You can see the ObjectGridEventListener bean that has been added, which is the write-behind watcher referred to above. The watcher interacts over the Maps for all primary shards in a JVM looking for ones with write-behind enabled. If it finds one then it tries to log up to 100 bad updates. It keeps watching a primary shard until the shard is moved to a different JVM. All applications using write-behind must use a watcher similar to this one. Otherwise, the Java™ virtual machines run out of memory because this error map is never evicted

See Example: Writing a write-behind dumper class for more information.

Related concepts:
Write-behind caching
Write-behind loader application design considerations
Related reference:
Example: Writing a write-behind dumper class
Example: Writing a write-behind dumper class

Example: Writing a write-behind dumper class

This sample source code shows how to write a watcher (dumper) to handle failed write-behind updates.

```
//
//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. "
//
//5724-J34 (C) COPYRIGHT International Business Machines Corp. 2009
//All Rights Reserved * Licensed Materials - Property of IBM
```

```
//
package utils;

import java.util.Collection;
import java.util.Iterator;
import java.util.concurrent.Callable;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.ScheduledFuture;
import java.util.concurrent.ScheduledThreadPoolExecutor;
import java.util.concurrent.TimeUnit;
import java.util.logging.Logger;

import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridException;
import com.ibm.websphere.objectgrid.ObjectGridRuntimeException;
import com.ibm.websphere.objectgrid.ObjectMap;
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.UndefinedMapException;
import com.ibm.websphere.objectgrid.plugins.ObjectGridEventGroup;
import com.ibm.websphere.objectgrid.plugins.ObjectGridEventListener;
import com.ibm.websphere.objectgrid.writebehind.FailedUpdateElement;
import com.ibm.websphere.objectgrid.writebehind.WriteBehindLoaderConstants;

/**
 * Write behind expects transactions to the Loader to succeed. If a transaction for a key fails then
 * it inserts an entry in a Map called PREFIX + mapName. The application should be checking this
 * map for entries to dump out write behind transaction failures. The application is responsible for
 * analyzing and then removing these entries. These entries can be large as they include the key, before
 * and after images of the value and the exception itself. Exceptions can easily be 20k on their own.
 *
 * The class is registered with the grid and an instance is created per primary shard in a JVM. It creates
 * a single thread
 * and that thread then checks each write behind error map for the shard, prints out the problem and
 * then removes the entry.
 *
 * This means there will be one thread per shard. If the shard is moved to another JVM then the deactivate
 * method stops the thread.
 * @author bnewport
 *
 */
public class WriteBehindDumper implements ObjectGridEventListener, ObjectGridEventGroup.ShardEvents,
 Callable<Boolean>
{
        static Logger logger = Logger.getLogger(WriteBehindDumper.class.getName());

        ObjectGrid grid;

        /**
         * Thread pool to handle table checkers. If the application has it's own pool
         * then change this to reuse the existing pool
         */
        static ScheduledExecutorService pool = new ScheduledThreadPoolExecutor(2); // two threads to dump records

        // the future for this shard
        ScheduledFuture<Boolean> future;

        // true if this shard is active
        volatile boolean isShardActive;

        /**
         * Normal time between checking Maps for write behind errors
         */
        final long BLOCKTIME_SECS = 20L;

        /**
         * An allocated session for this shard. No point in allocating them again and again
         */
        Session session;
        /**
         * When a primary shard is activated then schedule the checks to periodically check
         * the write behind error maps and print out any problems
         */
        public void shardActivated(ObjectGrid grid)
        {
                try
                {
                        this.grid = grid;
                        session = grid.getSession();

                        isShardActive = true;
                        future = pool.schedule(this, BLOCKTIME_SECS, TimeUnit.SECONDS); // check every BLOCKTIME_SECS
seconds initially
                }
                catch(ObjectGridException e)
                {
                        throw new ObjectGridRuntimeException("Exception activating write dumper", e);
                }
        }

        /**
```

```
         * Mark shard as inactive and then cancel the checker
         */
        public void shardDeactivate(ObjectGrid arg0)
        {
                isShardActive = false;
                // if it's cancelled then cancel returns true
                if(future.cancel(false) == false)
                {
                        // otherwise just block until the checker completes
                        while(future.isDone() == false) // wait for the task to finish one way or the other
                        {
                                try
                                {
                                        Thread.sleep(1000L); // check every second
                                }
                                catch(InterruptedException e)
                                {
                                }
                        }
                }
        }

        /**
         * Simple test to see if the map has write behind enabled and if so then return
         * the name of the error map for it.
         * @param mapName The map to test
         * @return The name of the write behind error map if it exists otherwise null
         */
        static public String getWriteBehindNameIfPossible(ObjectGrid grid, String mapName)
        {
                BackingMap map = grid.getMap(mapName);
                if(map != null && map.getWriteBehind() != null)
                {
                        return WriteBehindLoaderConstants.WRITE_BEHIND_FAILED_UPDATES_MAP_PREFIX + mapName;
                }
                else
                        return null;
        }

        /**
         * This runs for each shard. It checks if each map has write behind enabled and if it does
   * then it prints out any write behind
         * transaction errors and then removes the record.
         */
        public Boolean call()
        {
                logger.fine("Called for " + grid.toString());
                try
                {
                        // while the primary shard is present in this JVM
                        // only user defined maps are returned here, no system maps like write behind maps are in
                        // this list.
                        Iterator<String> iter = grid.getListOfMapNames().iterator();
                        boolean foundErrors = false;
                        // iterate over all the current Maps
                        while(iter.hasNext() && isShardActive)
                        {
                                String origName = iter.next();

                                // if it's a write behind error map
                                String name = getWriteBehindNameIfPossible(grid, origName);
                                if(name != null)
                                {
                                        // try to remove blocks of N errors at a time
                                        ObjectMap errorMap = null;
                                        try
                                        {
                                                errorMap = session.getMap(name);
                                        }
                                        catch(UndefinedMapException e)
                                        {
                                                // at startup, the error maps may not exist yet, patience...
                                                continue;
                                        }
                                        // try to dump out up to N records at once
                                        session.begin();
                                        for(int counter = 0; counter < 100; ++counter)
                                        {
                                                Integer seqKey = (Integer)errorMap.getNextKey(1L);
                                                if(seqKey != null)
                                                {
                                                        foundErrors = true;
                                                        FailedUpdateElement elem =
(FailedUpdateElement)errorMap.get(seqKey);
                                                        //
                                                        // Your application should log the problem here
                                                        logger.info("WriteBehindDumper ( " + origName + ") for key (" +
elem.getKey() + ") Exception: " +
                                                                elem.getThrowable().toString());
                                                        //
                                                        //
```

```
                                                errorMap.remove(seqKey);
                                        }
                                        else
                                                break;
                                }
                                session.commit();
                        }
                } // do next map
                // loop faster if there are errors
                if(isShardActive)
                {
                        // reschedule after one second if there were bad records
                        // otherwise, wait 20 seconds.
                        if(foundErrors)
                                future = pool.schedule(this, 1L, TimeUnit.SECONDS);
                        else
                                future = pool.schedule(this, BLOCKTIME_SECS, TimeUnit.SECONDS);
                }
        }
        catch(ObjectGridException e)
        {
                logger.fine("Exception in WriteBehindDumper" + e.toString());
                e.printStackTrace();

                //don't leave a transaction on the session.
                if(session.isTransactionActive())
                {
                        try { session.rollback(); } catch(Exception e2) {}
                }
        }
        return true;
    }

    public void destroy() {
            // TODO Auto-generated method stub

    }

    public void initialize(Session arg0) {
            // TODO Auto-generated method stub

    }

    public void transactionBegin(String arg0, boolean arg1) {
            // TODO Auto-generated method stub

    }

    public void transactionEnd(String arg0, boolean arg1, boolean arg2,
                    Collection arg3) {
            // TODO Auto-generated method stub

    }
}
```

Related concepts:
Write-behind caching
Write-behind loader application design considerations
Handling failed write-behind updates
Configuring write-behind loader support
Write-behind caching
Handling failed write-behind updates

JPA loader programming considerations

A Java™ Persistence API (JPA) Loader is a loader plug-in implementation that uses JPA to interact with the database. Use the following considerations when you develop an application that uses a JPA loader.

eXtreme Scale entity and JPA entity

You can designate any POJO class as an eXtreme Scale entity using eXtreme Scale entity annotations, XML configuration, or both. You can also designate the same POJO class as a JPA entity using JPA entity annotations, XML configuration, or both.

eXtreme Scale entity: An eXtreme Scale entity represents persistent data that is stored in ObjectGrid maps. An entity object is transformed into a key tuple and a value tuple, which are then stored as key-value pairs in the maps. A tuple is an array of primitive attributes.

JPA entity: A JPA entity represents persistent data that is stored in a relational database automatically using container-managed persistence. The data is persisted in some form of a data storage system in the appropriate form, such as database tuples in a database.

When an eXtreme Scale entity is persisted, its relations are stored in other entity maps. For example, when you are persisting a Consumer entity with a one-to-many relation to a ShippingAddress entity, if cascade-persist is enabled, the ShippingAddress entity is stored in the shippingAddress map in tuple format. If you

are persisting a JPA entity, the related JPA entities are also persisted to database tables if cascade-persist is enabled. When a POJO class is designated as both an eXtreme Scale entity and a JPA entity, the data can be persisted to both ObjectGrid entity maps and databases. Common uses follow:

- **Preload scenario**: An entity is loaded from a database using a JPA provider and persists it into ObjectGrid entity maps.
- **Loader scenario**: A Loader implementation is plugged in for the ObjectGrid entity maps so an entity stored in ObjectGrid entity maps can be persisted into or loaded from a database using JPA providers.

It is also common that a POJO class is designated as a JPA entity only. In that case, what is stored in the ObjectGrid maps are the POJO instances, versus the entity tuples in the ObjectGrid entity case.

Application design considerations for entity maps

When you are plugging in a JPALoader interface, the object instances are directly stored in the ObjectGrid maps.

However, you are when plugging in a JPAEntityLoader, the entity class is designated as both an eXtreme Scale entity and a JPA entity. In that case, treat this entity as if it has two persistent stores: the ObjectGrid entity maps and the JPA persistence store. The architecture becomes more complicated than the JPALoader case.

For more information about the JPAEntityLoader plug-in and application design considerations, see JPAEntityLoader plug-in. This information can also help if you plan to implement your own loader for the entity maps.

Performance considerations

Ensure that you set the proper eager or lazy fetch type for relationships. For example, a bidirectional one-to-many relationship Consumer to ShippingAddress, with OpenJPA to help explain the performance differences. In this example, a JPA query attempts `select o from Consumer o where . . .` to do a bulk load, and also load all related ShippingAddress objects. A one-to-many relationship defined in the Consumer class follows:

```
@Entity
public class Consumer implements Serializable {

    @OneToMany(mappedBy="consumer",cascade=CascadeType.ALL, fetch =FetchType.EAGER)
    ArrayList <ShippingAddress> addresses;
```

The many-to-one relation consumer defined in the ShippingAddress class follows:

```
@Entity
public class ShippingAddress implements Serializable{

    @ManyToOne(fetch=FetchType.EAGER)
    Consumer consumer;
}
```

If the fetch types of both relationships are configured as `eager`, OpenJPA uses N+1+1 queries to get all the Consumer objects and ShippingAddress objects, where N is the number of ShippingAddress objects. However if the ShippingAddress is changed to use lazy fetch type as follows, it only takes two queries to get all the data.

```
@Entity
public class ShippingAddress implements Serializable{

    @ManyToOne(fetch=FetchType.LAZY)
    Consumer consumer;
}
```

Although the query returns the same results, having a lower number of queries significantly decreases interaction with the database, which can increase application performance.

- JPAEntityLoader plug-in
  The JPAEntityLoader plug-in is a built-in Loader implementation that uses Java Persistence API (JPA) to communicate with the database when you are using the EntityManager API. When you are using the ObjectMap API, use the JPALoader loader.

Related concepts:
Plug-ins for communicating with databases
Writing a loader
JPAEntityLoader plug-in
Using a loader with entity maps and tuples
Writing a loader with a replica preload controller
Loaders

JPAEntityLoader plug-in

The JPAEntityLoader plug-in is a built-in Loader implementation that uses Java™ Persistence API (JPA) to communicate with the database when you are using the EntityManager API. When you are using the ObjectMap API, use the JPALoader loader.

Loader details

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.

Loaders provide two main functions:

1. **get**: In the get method, the JPAEntityLoader plug-in first calls the javax.persistence.EntityManager.find(Class entityClass, Object key) method to find the JPA entity. Then the plug-in projects this JPA entity into entity tuples. During the projection, both the tuple attributes and the association keys are stored in the value tuple. After processing each key, the get method returns a list of entity value tuples.
2. **batchUpdate**: The batchUpdate method takes a LogSequence object that contains a list of LogElement objects. Each LogElement object contains a key tuple and a value tuple. To interact with the JPA provider, you must first find the eXtreme Scale entity based on the key tuple. Based on the LogElement type, you run the following JPA calls:
   - **insert**: javax.persistence.EntityManager.persist(Object o)
   - **update**: javax.persistence.EntityManager.merge(Object o)
   - **remove**: javax.persistence.EntityManager.remove(Object o)

A LogElement with type **update** makes the JPAEntityLoader call the javax.persistence.EntityManager.merge(Object o) method to merge the entity. However, an **update** type LogElement might be the result of either a com.ibm.websphere.objectgrid.em.EntityManager.merge(object o) call or an attribute change of the eXtreme Scale EntityManager managed-instance. See the following example:

```
com.ibm.websphere.objectgrid.em.EntityManager em = og.getSession().getEntityManager();
em.getTransaction().begin();
Consumer c1 = (Consumer) em.find(Consumer.class, c.getConsumerId());
c1.setName("New Name");
em.getTransaction().commit();
```

In this example, an update type LogElement is sent to the JPAEntityLoader of the map consumer. The javax.persistence.EntityManager.merge(Object o) method is called to the JPA entity manager instead of an attribute update to the JPA-managed entity. Because of this changed behavior, some limitations exist with using this programming model.

Application design rules

Entities have relationships with other entities. Designing an application with relationships involved and with JPAEntityLoader plugged in requires additional considerations. The application should follow the following four rules, described in the following sections.

Limited relationship depth support

The JPAEntityLoader is only supported when using entities without any relationships or entities with single-level relationships. Relationships with more than one level, such as `Company > Department > Employee` are not supported.

One loader per map

Using the Consumer-ShippingAddress entity relationships as an example, when you load a consumer with eager fetch enabled, you could load all the related ShippingAddress objects. When you persist or merge a Consumer object, you could persist or merge related ShippingAddress objects if cascade-persist or cascade-merge is enabled.

You cannot plug in a loader for the root entity map which stores the Consumer entity tuples. You must configure a loader for each entity map.

Same cascade type for JPA and eXtreme Scale

Reconsider the scenario where the entity Consumer has a one-to-many relationship with ShippingAddress. You can look at the scenario where cascade-persist is enabled for this relationship. When a Consumer object is persisted into eXtreme Scale, the associated N number of ShippingAddress objects are also persisted into eXtreme Scale.

A persist call of the Consumer object with a cascade-persist relationship to ShippingAddress translates to one javax.persistence.EntityManager.persist(consumer) method call and *N* javax.persistence.EntityManager.persist(shippingAddress) method calls by the JPAEntityLoader layer. However, these N extra persist calls to ShippingAddress objects are unnecessary because of the cascade-persist setting from the JPA provider point of view. To solve this problem, eXtreme Scale provides a new method isCascaded on the LogElement interface. The isCascaded method indicates whether the LogElement is a result of an eXtreme Scale EntityManager cascade operation. In this example, the JPAEntityLoader of the ShippingAddress map receives *N* LogElement objects because of the cascade persist calls. The JPAEntityLoader finds out that the isCascaded method returns true and then ignores them without making any JPA calls. Therefore, from a JPA point of view, only one javax.persistence.EntityManager.persist(consumer) method call is received.

The same behavior is exhibited if you merge an entity or remove an entity with cascade enabled. The cascaded operations are ignored by the JPAEntityLoader plug-in.

The design of the cascade support is to replay the eXtreme Scale EntityManager operations to the JPA providers. These operations include persist, merge, and remove operations. To enable cascade support, verify that the cascade setting for the JPA and the eXtreme Scale EntityManager are the same.

Use entity update with caution

As previously described, the design of the cascade support is to replay eXtreme Scale EntityManager operations to the JPA providers. If your application calls the ogEM.persist(consumer) method to the eXtreme Scale EntityManager, even the associated ShippingAddress objects are persisted because of the cascade-persist setting, and the JPAEntityLoader only calls the jpAEM.persist(consumer) method to the JPA providers.

However, if your application updates a managed entity, this update translates to a JPA merge call by the JPAEntityLoader plug-in. In this scenario, support for multiple levels of relationships and key associations is not guaranteed. In this case, the best practice is to use the javax.persistence.EntityManager.merge(o) method instead of updating a managed entity.

Related concepts:
Plug-ins for communicating with databases

Writing a loader
Using a loader with entity maps and tuples
Writing a loader with a replica preload controller
Loaders
Related reference:
JPA loader programming considerations

Using a loader with entity maps and tuples

The entity manager converts all entity objects into tuple objects before they are stored in an WebSphere® eXtreme Scale map. Every entity has a key tuple and a value tuple. This key-value pair is stored in the associated eXtreme Scale map for the entity. When you are using an eXtreme Scale map with a loader, the loader must interact with the tuple objects.

eXtreme Scale includes loader plug-ins that simplify integration with relational databases. The Java™ Persistence API (JPA) Loaders use a Java Persistence API to interact with the database and create the entity objects. The JPA loaders are compatible with eXtreme Scale entities.

Tuples

A tuple contains information about the attributes and associations of an entity. Primitive values are stored using their primitive wrappers. Other supported object types are stored in their native format. Associations to other entities are stored as a collection of key tuple objects that represent the keys of the target entities.

Each attribute or association is stored using a zero-based index. You can retrieve the index of each attribute using the getAttributePosition or getAssociationPosition methods. After the position is retrieved, it remains unchanged for the duration of the eXtreme Scale life cycle. The position can change when the eXtreme Scale is restarted. The setAttribute, setAssociation and setAssociations methods are used to update the elements in the tuple.

Attention: When you are creating or updating tuple objects, update every primitive field with a non-null value. Primitive values such as `int` should not be null. If you do not change the value to a default, poor performance issues can result, also affecting fields marked with the @Version annotation or version attribute in the entity descriptor XML file.
The following example further explains how to process tuples. For more information about defining entities for this example, see Entity manager tutorial: Order Entity Schema. WebSphere eXtreme Scale is configured for using loaders with each of the entities. Additionally, only the Order entity is taken, and this specific entity has a many-to-one relationship with the Customer entity. The attribute name is `customer`, and it has a one-to-many relationship with the OrderLine entity.

Use the Projector to build Tuple objects automatically from entities. Using the Projector can simplify loaders when you are using an object-relational mapping utility such as Hibernate or JPA.

order.java

```
@Entity
public class Order
{
    @Id  String orderNumber;
    java.util.Date date;
    @OneToOne(cascade=CascadeType.PERSIST) Customer customer;
    @OneToMany(cascade=CascadeType.ALL, mappedBy="order") @OrderBy("lineNumber") List<OrderLine> lines;
}
```

customer.java

```
@Entity
public class Customer {
    @Id String id;
    String firstName;
    String surname;
    String address;
    String phoneNumber;
}
```

orderLine.java

```
@Entity
public class OrderLine
{
    @Id @ManyToOne(cascade=CascadeType.PERSIST) Order order;
    @Id int lineNumber;
    @OneToOne(cascade=CascadeType.PERSIST) Item item;
    int quantity;
    double price;
}
```

A OrderLoader class that implements the Loader interface is shown in the following code. The following example assumes that an associated TransactionCallback plug-in is defined.

orderLoader.java

```
public class OrderLoader implements com.ibm.websphere.objectgrid.plugins.Loader {

        private EntityMetadata entityMetaData;
        public void batchUpdate(TxID txid, LogSequence sequence)
            throws LoaderException, OptimisticCollisionException {
        ...
        }
        public List get(TxID txid, List keyList, boolean forUpdate)
            throws LoaderException {
        ...
        }
        public void preloadMap(Session session, BackingMap backingMap)
            throws LoaderException {
                this.entityMetaData=backingMap.getEntityMetadata();
        }

}
```

The instance variable entityMetaData is initialized during the preLoadMap method call from the eXtreme Scale. The *entityMetaData* variable is not null if the Map is configured to use entities. Otherwise, the value is null.

batchUpdate method

The batchUpdate method provides the ability to know what action the application intended to perform. Based on an insert, update or a delete operation, a connection can be opened to the database and the work performed. Because the key and values are of type Tuple, they must be transformed so the values make sense on the SQL statement.

The ORDER table was created with the following Data Definition Language (DDL) definition, as shown in the following code:

```
CREATE TABLE ORDER (ORDERNUMBER VARCHAR(250) NOT NULL, DATE TIMESTAMP, CUSTOMER_ID VARCHAR(250))
ALTER TABLE ORDER ADD CONSTRAINT PK_ORDER PRIMARY KEY (ORDERNUMBER)
```

The following code demonstrates how to convert a Tuple to an Object:

```
public void batchUpdate(TxID txid, LogSequence sequence)
            throws LoaderException, OptimisticCollisionException {
        Iterator iter = sequence.getPendingChanges();
        while (iter.hasNext()) {
            LogElement logElement = (LogElement) iter.next();
            Object key = logElement.getKey();
            Object value = logElement.getCurrentValue();

            switch (logElement.getType().getCode()) {
            case LogElement.CODE_INSERT:

1)              if (entityMetaData!=null) {

// The order has just one key orderNumber
2)                  String ORDERNUMBER=(String) getKeyAttribute("orderNumber", (Tuple) key);
// Get the value of date
3)                  java.util.Date unFormattedDate = (java.util.Date) getValueAttribute("date",(Tuple)value);
// The values are 2 associations. Lets process customer because
// the our table contains customer.id as primary key
4)                  Object[] keys= getForeignKeyForValueAssociation("customer","id",(Tuple) value);
                    //Order to Customer is M to 1. There can only be 1 key
5)                  String CUSTOMER_ID=(String)keys[0];
// parse variable unFormattedDate and format it for the database as formattedDate
6)                   String formattedDate = "2007-05-08-14.01.59.780272"; // formatted for DB2
// Finally, the following SQL statement to insert the record
7) //INSERT INTO ORDER (ORDERNUMBER, DATE, CUSTOMER_ID) VALUES(ORDERNUMBER,formattedDate, CUSTOMER_ID)
                }
                break;
            case LogElement.CODE_UPDATE:
                break;
            case LogElement.CODE_DELETE:
                break;
            }
        }

    }
// returns the value to attribute as stored in the key Tuple
 private Object getKeyAttribute(String attr, Tuple key) {
        //get key metadata
        TupleMetadata keyMD = entityMetaData.getKeyMetadata();
        //get position of the attribute
        int keyAt = keyMD.getAttributePosition(attr);
        if (keyAt > -1) {
            return key.getAttribute(keyAt);
        } else { // attribute undefined
            throw new IllegalArgumentException("Invalid position index for  "+attr);
        }

    }
// returns the value to attribute as stored in the value Tuple
    private Object getValueAttribute(String attr, Tuple value) {
        //similar to above, except we work with value metadata instead
        TupleMetadata valueMD = entityMetaData.getValueMetadata();
```

```
        int keyAt = valueMD.getAttributePosition(attr);
        if (keyAt > -1) {
            return value.getAttribute(keyAt);
        } else {
            throw new IllegalArgumentException("Invalid position index for  "+attr);
        }
    }
// returns an array of keys that refer to association.
    private Object[] getForeignKeyForValueAssociation(String attr, String fk_attr, Tuple value) {
        TupleMetadata valueMD = entityMetaData.getValueMetadata();
        Object[] ro;

        int customerAssociation = valueMD.getAssociationPosition(attr);
        TupleAssociation tupleAssociation = valueMD.getAssociation(customerAssociation);

        EntityMetadata targetEntityMetaData = tupleAssociation.getTargetEntityMetadata();

        Tuple[] customerKeyTuple = ((Tuple) value).getAssociations(customerAssociation);

        int numberOfKeys = customerKeyTuple.length;
        ro = new Object[numberOfKeys];

        TupleMetadata keyMD = targetEntityMetaData.getKeyMetadata();
        int keyAt = keyMD.getAttributePosition(fk_attr);
        if (keyAt < 0) {
            throw new IllegalArgumentException("Invalid position index for  " + attr);
        }
        for (int i = 0; i < numberOfKeys; ++i) {
            ro[i] = customerKeyTuple[i].getAttribute(keyAt);
        }

        return ro;

    }
```

1. Ensure that entityMetaData is not null, which implies that the key and value cache entries are of type Tuple. From the entityMetaData, Key TupleMetaData is retrieved, which really reflects only the key part of Order metadata.
2. Process the KeyTuple and get the value of Key Attribute orderNumber
3. Process the ValueTuple and get the value of attribute date
4. Process the ValueTuple and get the value of Keys from association customer
5. Extract CUSTOMER_ID. Based on relationship, an Order can only have one customer, we will only have one key. Hence the size of keys is 1. For simplicity, we skipped parsing of date to correct format.
6. Because this is an insert operation, the SQL statement is passed onto the data source connection to complete the insert operation.

Transaction demarcation and access to database is covered in Writing a loader.

get method

If the key is not found in the cache, call the get method in the Loader plug-in to find the key.

The key is a Tuple. The first step is to convert the Tuple to primitive values that can be passed onto the SELECT SQL statement. After all the attributes are retrieved from the database, you must convert into Tuples. The following code demonstrates the Order class.

```
public List get(TxID txid, List keyList, boolean forUpdate)     throws LoaderException {
                System.out.println("OrderLoader: Get called");
                ArrayList returnList = new ArrayList();

1)              if (entityMetaData != null) {
                        int index=0;
                        for (Iterator iter = keyList.iterator(); iter.hasNext();) {
2)                              Tuple orderKeyTuple=(Tuple) iter.next();

                                // The order has just one key orderNumber
3)                              String ORDERNUMBERKEY = (String) getKeyAttribute("orderNumber",orderKeyTuple);
                                //We need to run a query to get values of
4)                              // SELECT CUSTOMER_ID, date FROM ORDER WHERE ORDERNUMBER='ORDERNUMBERKEY'

5)                              //1) Foreign key: CUSTOMER_ID
6)                              //2) date
                                // Assuming those two are returned as
7)                              String  CUSTOMER_ID = "C001"; // Assuming Retrieved and initialized
8)                              java.util.Date retrievedDate = new java.util.Date();
                                // Assuming this date reflects the one in database

                                // We now need to convert this data into a tuple before returning

                                //create a value tuple
9)                              TupleMetadata valueMD = entityMetaData.getValueMetadata();
                                Tuple valueTuple=valueMD.createTuple();


                                //add retrievedDate object to Tuple
                                int datePosition = valueMD.getAttributePosition("date");
10)                             valueTuple.setAttribute(datePosition, retrievedDate);

                                //Next need to add the Association
11)                             int customerPosition=valueMD.getAssociationPosition("customer");
```

```
                                TupleAssociation customerTupleAssociation =
                                  valueMD.getAssociation(customerPosition);
                                EntityMetadata customerEMD = customerTupleAssociation.getTargetEntityMetadata();
                                TupleMetadata customerTupleMDForKEY=customerEMD.getKeyMetadata();
12)                             int customerKeyAt=customerTupleMDForKEY.getAttributePosition("id");


                                Tuple customerKeyTuple=customerTupleMDForKEY.createTuple();
                                customerKeyTuple.setAttribute(customerKeyAt, CUSTOMER_ID);
13)                             valueTuple.addAssociationKeys(customerPosition, new Tuple[] {customerKeyTuple});


14)                             int linesPosition = valueMD.getAssociationPosition("lines");
                                TupleAssociation linesTupleAssociation = valueMD.getAssociation(linesPosition);
                                EntityMetadata orderLineEMD = linesTupleAssociation.getTargetEntityMetadata();
                                TupleMetadata orderLineTupleMDForKEY = orderLineEMD.getKeyMetadata();
                                int lineNumberAt = orderLineTupleMDForKEY.getAttributePosition("lineNumber");
                                int orderAt = orderLineTupleMDForKEY.getAssociationPosition("order");

                                if (lineNumberAt < 0 || orderAt < 0) {
                                        throw new IllegalArgumentException(
                                        "Invalid position index for lineNumber or order "+
                                        lineNumberAt + " " + orderAt);
                                }
15) // SELECT LINENUMBER FROM ORDERLINE WHERE ORDERNUMBER='ORDERNUMBERKEY'
                                // Assuming two rows of line number are returned with values 1 and 2

                                Tuple orderLineKeyTuple1 = orderLineTupleMDForKEY.createTuple();
                                orderLineKeyTuple1.setAttribute(lineNumberAt, new Integer(1));// set Key
                                orderLineKeyTuple1.addAssociationKey(orderAt, orderKeyTuple);

                                Tuple orderLineKeyTuple2 = orderLineTupleMDForKEY.createTuple();
                                orderLineKeyTuple2.setAttribute(lineNumberAt, new Integer(2));// Init Key
                                orderLineKeyTuple2.addAssociationKey(orderAt, orderKeyTuple);

16)                             valueTuple.addAssociationKeys(linesPosition, new Tuple[]
                                    {orderLineKeyTuple1, orderLineKeyTuple2 });

                                returnList.add(index, valueTuple);

                                index++;

                        }
                }else {
                        // does not support tuples
                }
                return returnList;
        }
```

1. The get method is called when the ObjectGrid cache could not find the key and requests the loader to fetch. Test for entityMetaData value and proceed if not null.
2. The keyList contains Tuples.
3. Retrieve value of attribute orderNumber.
4. Run query to retrieve date (value) and customer ID (foreign key).
5. CUSTOMER_ID is a foreign key that must be set in the association tuple.
6. The date is a value and should already be set.
7. Since this example does not perform JDBC calls, CUSTOMER_ID is assumed.
8. Since this example does not perform JDBC calls, date is assumed.
9. Create value Tuple.
10. Set the value of date into the Tuple, based on its position.
11. Order has two associations. Start with the attribute customer which refers to the customer entity. You must have the value of ID to set in the Tuple.
12. Find the position of ID on the customer entity.
13. Set the values of the association keys only.
14. Also, lines is an association that must be set up as a group of association keys, in the same way as you did for customer association.
15. Since you must set up keys for the lineNumber associated with this order, run the SQL to retrieve lineNumber values.
16. Set up the association keys in the valueTuple. This completes the creation of the Tuple that is returned to the BackingMap.

This topic offers the steps create tuples, and a description of the Order entity only. Complete similar steps for the other entities, and the entire process that is tied together with the TransactionCallback plug-in. See Plug-ins for managing transaction life cycle events for details.

Related concepts:
Plug-ins for communicating with databases
Writing a loader
JPAEntityLoader plug-in
Writing a loader with a replica preload controller
Loaders
Tuning serialization performance
ObjectTransformer plug-in
Tuning serialization
Serialization using the DataSerializer plug-ins
Related reference:
JPA loader programming considerations

Writing a loader with a replica preload controller

A Loader with a replica preload controller is a Loader that implements the ReplicaPreloadController interface in addition to the Loader interface.

The ReplicaPreloadController interface is designed to provide a way for a replica that becomes the primary shard to know whether the previous primary shard has completed the preload process. If the preload is partially completed, the information to pick up where the previous primary left is provided. With the ReplicaPreloadController interface implementation, a replica that becomes the primary continues the preload process where the previous primary left and continues to finish the overall preload.

In a distributed WebSphere® eXtreme Scale environment, a map can have replicas and might preload large volume of data during initialization. The preload is a Loader activity and only occurs in the primary map during initialization. The preload might take a long time to complete if a large volume of data is preloaded. If the primary map has completed large portion of preload data, but is stopped for unknown reason during initialization, a replica becomes the primary. In this situation, the preloaded data that was completed by the previous primary is lost because the new primary normally performs an unconditional preload. With an unconditional preload, the new primary starts the preload process from the beginning and the previous preloaded data is ignored. If you want the new primary to pick up where the previous primary left during preload process, provide a Loader that implements the ReplicaPreloadController interface. For more information see the API documentation.

For information about Loaders, see Loaders. If you are interested in writing a regular Loader plug-in, see Writing a loader.

The ReplicaPreloadController interface has the following definition:

```
public interface ReplicaPreloadController
{
    public static final class Status
    {
        static public final Status PRELOADED_ALREADY =
                                                    new Status(K_PRELOADED_ALREADY);
        static public final Status FULL_PRELOAD_NEEDED =
                                                    new Status(K_FULL_PRELOAD_NEEDED);
        static public final Status PARTIAL_PRELOAD_NEEDED =
                                                    new Status(K_PARTIAL_PRELOAD_NEEDED);
    }

    Status checkPreloadStatus(Session session,
                              BackingMap bmap);
}
```

The following sections discuss some of the methods of the Loader and ReplicaPreloadController interface.

checkPreloadStatus method

When a Loader implements ReplicaPreloadController interface, the checkPreloadStatus method is called before the preloadMap method during map initialization. The return status of this method determines if the preloadMap method is called. If this method returns `Status#PRELOADED_ALREADY`, the preload method is not called. Otherwise, the preload method runs. Because of this behavior, this method should serve as the Loader initialization method. You must initialize the Loader properties in this method. This method must return the correct status, or the preload might not work as expected.

```
public Status checkPreloadStatus(Session session,
                                                    BackingMap backingMap) {
        // When a loader implements ReplicaPreloadController interface,
                        //     this method will be called before preloadMap method during
                        // map initialization. Whether the preloadMap method will be
                        //     called depends on teh returned status of this method. So, this
                        // method also serve as Loader's initialization method. This method
                        // has to return the right staus, otherwise the preload may not
                        // work as expected.

        // Note: must initialize this loader instance here.
        ivOptimisticCallback = backingMap.getOptimisticCallback();
        ivBackingMapName = backingMap.getName();
        ivPartitionId = backingMap.getPartitionId();
        ivPartitionManager = backingMap.getPartitionManager();
        ivTransformer = backingMap.getObjectTransformer();
        preloadStatusKey = ivBackingMapName + "_" + ivPartitionId;

        try {
            // get the preloadStatusMap to retrieve preload status that
                                    // could be set by other JVMs.
            ObjectMap preloadStatusMap = session.getMap(ivPreloadStatusMapName);

            // retrieve last recorded preload data chunk index.
            Integer lastPreloadedDataChunk = (Integer) preloadStatusMap
                                                                    .get(preloadStatusKey);

            if (lastPreloadedDataChunk == null) {
                preloadStatus = Status.FULL_PRELOAD_NEEDED;
            } else {
                preloadedLastDataChunkIndex = lastPreloadedDataChunk.intValue();
                if (preloadedLastDataChunkIndex == preloadCompleteMark) {
                    preloadStatus = Status.PRELOADED_ALREADY;
                } else {
                    preloadStatus = Status.PARTIAL_PRELOAD_NEEDED;
                }
            }
```

```
            System.out.println("TupleHeapCacheWithReplicaPreloadControllerLoader.
                                                                checkPreloadStatus()
                                        -> map = " + ivBackingMapName + ", preloadStatusKey = " + preloadStatusKey
                    + ", retrieved lastPreloadedDataChunk =" + lastPreloadedDataChunk + ",
                                                                determined preloadStatus = "
                    + getStatusString(preloadStatus));

        } catch (Throwable t) {
            t.printStackTrace();
        }

        return preloadStatus;
    }
```

preloadMap method

Running the preloadMap method depends on the returned result from checkPreloadStatus method. If the preloadMap method is called, it typically must retrieve preload status information from designated preload status map and determine how to proceed. Ideally, the preloadMap method should know if the preload is partially complete and exactly where to start. During the data preload, the preloadMap method should first check the preload status (stored in the preload status map), and then update the preload status whenever data is pushed into the cache. The preload status (that is stored in the preload status map) is retrieved by the checkPreloadStatus method when it needs to check the preload status.

```
public void preloadMap(Session session, BackingMap backingMap)
                                        throws LoaderException {
        EntityMetadata emd = backingMap.getEntityMetadata();
        if (emd != null && tupleHeapPreloadData != null) {
            // The getPreLoadData method is similar to fetching data
                                         // from database. These data will be push into cache as
                                         // preload process.
            ivPreloadData = tupleHeapPreloadData.getPreLoadData(emd);

            ivOptimisticCallback = backingMap.getOptimisticCallback();
            ivBackingMapName = backingMap.getName();
            ivPartitionId = backingMap.getPartitionId();
            ivPartitionManager = backingMap.getPartitionManager();
            ivTransformer = backingMap.getObjectTransformer();
            Map preloadMap;

            if (ivPreloadData != null) {
                try {
                    ObjectMap map = session.getMap(ivBackingMapName);

                    // get the preloadStatusMap to record preload status.
                    ObjectMap preloadStatusMap = session.
                                                                                getMap(ivPreloadStatusMapName);

                    // Note: when this preloadMap method is invoked, the
                                                                    // checkPreloadStatus has been called, Both
preloadStatus
                                                                    // and preloadedLastDataChunkIndex have been
set. And the
                                                                    // preloadStatus must be either
PARTIAL_PRELOAD_NEEDED
                                                                    // or FULL_PRELOAD_NEEDED that will require a
preload again.

                    // If large amount of data will be preloaded, the data usually
                                                                    // is divided into few chunks and the preload
process will
                                                                    // process each chunk within its own tran. This
sample only
                                                                    // preload few entries and assuming each entry
represent a chunk.
                    // so that the preload process an entry in a tran to simulate
                                                                    // chunk preloading.

                    Set entrySet = ivPreloadData.entrySet();
                    preloadMap = new HashMap();
                    ivMap = preloadMap;

                    // The dataChunkIndex represent the data chunk that is in
                                                                    // processing
                    int dataChunkIndex = -1;
                    boolean shouldRecordPreloadStatus = false;
                    int numberOfDataChunk = entrySet.size();
                    System.out.println("    numberOfDataChunk to be preloaded = "
                                                                            + numberOfDataChunk);

                    Iterator it = entrySet.iterator();
                    int whileCounter = 0;
                    while (it.hasNext()) {
                        whileCounter++;
                        System.out.println("preloadStatusKey = " + preloadStatusKey
                                                                                                    + " ,
                                                                                whileCounter = " +
whileCounter);
```

```
                        dataChunkIndex++;

                        // if the current dataChunkIndex <= preloadedLastDataChunkIndex
                        // no need to process, becasue it has been preloaded by
                                                                                // other JVM before. only need
to process dataChunkIndex
                                                                                // > preloadedLastDataChunkIndex
                        if (dataChunkIndex <= preloadedLastDataChunkIndex) {
                            System.out.println("ignore current dataChunkIndex =

" + dataChunkIndex + " that has been previously

preloaded.");
                            continue;
                        }

                        // Note: This sample simulate data chunk as an entry.
                        // each key represent a data chunk for simplicity.
                        // If the primary server or shard stopped for unknown
                                                                        // reason, the preload status that
indicates the progress
                                                                        // of preload should be available in
preloadStatusMap. A
                                                                        // replica that become a primary
can get the preload status
                                                                        // and determine how to preload
again.
                        // Note: recording preload status should be in the same
                                                                        // tran as putting data into cache; so
that if tran
                                                                        // rollback or error, the
recorded preload status is the
                                                                        // actual status.

                        Map.Entry entry = (Entry) it.next();
                        Object key = entry.getKey();
                        Object value = entry.getValue();
                        boolean tranActive = false;

                        System.out.println("processing data chunk. map = " +

this.ivBackingMapName + ", current dataChunkIndex = " +
                                                                                dataChunkIndex +
", key = " + key);

                        try {
                            shouldRecordPreloadStatus = false; // re-set to false
                            session.beginNoWriteThrough();
                            tranActive = true;

                            if (ivPartitionManager.getNumOfPartitions() == 1) {
                                // if just only 1 partition, no need to deal with
                                                                                //
partition.
                                // just push data into cache
                                map.put(key, value);
                                preloadMap.put(key, value);
                                shouldRecordPreloadStatus = true;
                            } else if (ivPartitionManager.getPartition(key) ==

ivPartitionId) {
                                // if map is partitioned, need to consider the
                                                                                //
partition key only preload data that belongs
                                                                                // to
this partition.
                                map.put(key, value);
                                preloadMap.put(key, value);
                                shouldRecordPreloadStatus = true;
                            } else {
                                // ignore this entry, because it does not belong to
                                                                                // this
partition.
                            }

                            if (shouldRecordPreloadStatus) {
                                System.out.println("record preload status. map = " +

this.ivBackingMapName + ", preloadStatusKey = " +

preloadStatusKey + ", current dataChunkIndex ="

+ dataChunkIndex);
                                if (dataChunkIndex == numberOfDataChunk) {
                                    System.out.println("record preload status. map = " +

this.ivBackingMapName + ", preloadStatusKey = " +

preloadStatusKey + ", mark complete =" +

preloadCompleteMark);
```

```
                                    // means we are at the lastest data chunk, if commit
// successfully, record preload complete.
// at this point, the preload is considered to be done
                                    // use -99 as special mark for preload complete status.
                                    preloadStatusMap.get(preloadStatusKey);
                                    // a put follow a get will become update if the get
// return an object, otherwise, it will be insert.
                                    preloadStatusMap.put(preloadStatusKey, new
Integer(preloadCompleteMark));
                                } else {
                                    // record preloaded current dataChunkIndex into
// preloadStatusMap a put follow a get will become
// update if teh get return an object, otherwise, it
// will be insert.
                                    preloadStatusMap.get(preloadStatusKey);
                                    preloadStatusMap.put(preloadStatusKey, new
Integer(dataChunkIndex));
                                }
                            }
                            session.commit();
                            tranActive = false;
                            // to simulate preloading large amount of data
                            // put this thread into sleep for 30 secs.
                            // The real app should NOT put this thread to sleep
                            Thread.sleep(10000);
                        } catch (Throwable e) {
                            e.printStackTrace();
                            throw new LoaderException("preload failed with
                                                                                    exception: " + e, e);
                        } finally {
                            if (tranActive && session != null) {
                                try {
                                    session.rollback();
                                } catch (Throwable e1) {
                                    // preload ignoring exception from rollback
                                }
                            }
                        }
                    }
                    // at this point, the preload is considered to be done for sure
                    // use -99 as special mark for preload complete status.
                    // this is a insurance to make sure the complete mark is set.
                    // besides, when partitioning, each partition does not know when
                                                                    // is its last data chunk. so the following
block serves as the
                                                                    // overall preload status complete reporting.
                    System.out.println("Overall preload status complete -> record
                                                                                    preload status. map =
" + this.ivBackingMapName + ",
                                                                                    preloadStatusKey = " +
preloadStatusKey + ", mark complete =" +
preloadCompleteMark);
                    session.begin();
                    preloadStatusMap.get(preloadStatusKey);
                    // a put follow a get will become update if teh get return an object,
                    // otherwise, it will be insert.
                    preloadStatusMap.put(preloadStatusKey, new Integer(preloadCompleteMark));
                    session.commit();
                    ivMap = preloadMap;
                } catch (Throwable e) {
                    e.printStackTrace();
                    throw new LoaderException("preload failed with exception: " + e, e);
                }
            }
        }
    }
```

Preload status map

You must use a preload status map to support the ReplicaPreloadController interface implementation. The preloadMap method should always check the preload status stored in the preload status map first and update the preload status in the preload status map whenever it pushes data into the cache. The

checkPreloadStatus method can retrieve the preload status from preload status map, determine the preload status, and return the status to its caller. The preload status map should be in the same mapSet as other maps that have replica preload controller Loaders.

Related concepts:
Replication for availability
Partitioning
Plug-ins for communicating with databases
Writing a loader
JPAEntityLoader plug-in
Using a loader with entity maps and tuples
Loaders
Related reference:
JPA loader programming considerations

Plug-ins for managing transaction life cycle events

Use the TransactionCallback plug-in to customize versioning and comparison operations of cache objects when you are using the optimistic locking strategy.

You can provide a pluggable optimistic callback object that implements the com.ibm.websphere.objectgrid.plugins.OptimisticCallback interface. For entity maps, a high performance OptimisticCallback plug-in is automatically configured.

Purpose

Use the OptimisticCallback interface to provide optimistic comparison operations for the values of a map. An OptimisticCallback implementation is required when you use the optimistic locking strategy. WebSphere® eXtreme Scale provides a default OptimisticCallback implementation. However, usually the application must plug in its own implementation of the OptimisticCallback interface. See Locking strategies for more information.

Default implementation

The eXtreme Scale framework provides a default implementation of the OptimisticCallback interface that is used if the application does not plug in an application-provided OptimisticCallback object, as demonstrated in the previous section. The default implementation always returns the special value of NULL_OPTIMISTIC_VERSION as the version object for the value and never updates the version object. This action makes optimistic comparison a no operation function. In most cases, you do not want the no operation function to occur when you are using the optimistic locking strategy. Your applications must implement the OptimisticCallback interface and plug in their own OptimisticCallback implementations so that the default implementation is not used. However, at least one scenario exists where the default provided OptimisticCallback implementation is useful. Consider the following situation:

- A loader is plugged for the backing map.
- The loader knows how to perform the optimistic comparison without assistance from an OptimisticCallback plug-in.

How can the loader know how to deal with optimistic versioning without assistance from an OptimisticCallback object? The loader has knowledge of the value class object and knows which field of the value object is used as an optimistic versioning value. For example, suppose the following interface is used for the value object for the employees map:

```
public interface Employee
{
    // Sequential sequence number used for optimistic versioning.
    public long getSequenceNumber();
    public void setSequenceNumber(long newSequenceNumber);
    // Other get/set methods for other fields of Employee object.
}
```

In this case, the loader knows that it can use the getSequenceNumber method to get the current version information for an Employee value object. The loader increments the returned value to generate a new version number before updating the persistent storage with the new Employee value. For a Java™ database connectivity (JDBC) loader, the current sequence number in the where clause of an overqualified SQL update statement is used, and it uses the new generated sequence number to set the sequence number column to the new sequence number value.

Another possibility is that the loader makes use of some backend-provided function that automatically updates a hidden column that can be used for optimistic versioning. In some cases, a stored procedure or trigger can possibly be used to help maintain a column that holds versioning information. If the loader is using one of these techniques for maintaining optimistic versioning information, then the application does not need to provide an OptimisticCallback implementation. You can use the default OptimisticCallback implementation because the loader is able to handle optimistic versioning without any assistance from an OptimisticCallback object.

Default implementation for entities

Entities are stored in the ObjectGrid using tuple objects. The default OptimisticCallback implementation behaves similarly to the behavior for non-entity maps. However, the version field in the entity is identified using the @Version annotation or the version attribute in the entity descriptor XML file.

The version attribute can be of the following types: int, Integer, short, Short, long, Long or java.sql.Timestamp. An entity should have only one version attribute defined. The version attribute should be set only during construction. After the entity is persisted, the value of the version attribute should not be modified.

If a version attribute is not configured and the optimistic locking strategy is used, then the entire tuple is implicitly versioned using the entire state of the tuple.

In the following example, the Employee entity has a long version attribute named SequenceNumber:

```
@Entity
public class Employee
{
private long sequence;
    // Sequential sequence number used for optimistic versioning.
    @Version
    public long getSequenceNumber() {
        return sequence;
    }
    public void setSequenceNumber(long newSequenceNumber) {
        this.sequence = newSequenceNumber;
    }
    // Other get/set methods for other fields of Employee object.
}
```

Writing an OptimisticCallback implementation

An OptimisticCallback implementation must implement the OptimisticCallback interface and follow the common ObjectGrid plug-in conventions

The following list provides a description or consideration for each of the methods in the OptimisticCallback interface:

NULL_OPTIMISTIC_VERSION

This special value is returned by getVersionedObjectForValue method if the default OptimisticCallback implementation is used instead of an application-provided OptimisticCallback implementation.

getVersionedObjectForValue method

The getVersionedObjectForValue method might return a copy of the value or it might return an attribute of the value that can be used for versioning purposes. This method is called whenever an object is associated with a transaction. When no Loader is set into a backing map, the backing map uses this value at commit time to perform an optimistic version comparison. The optimistic version comparison is used by the backing map to ensure that the version has not changed since this transaction first accessed the map entry that was modified by this transaction. If another transaction had already modified the version for this map entry, the version comparison fails and the backing map displays an OptimisticCollisionException exception to force rollback of the transaction. If a Loader is plugged in, the backing map does not use the optimistic versioning information. Instead, the Loader is responsible for performing the optimistic versioning comparison and updating the versioning information when necessary. The Loader typically gets the initial versioning object from the LogElement passed to the batchUpdate method on the Loader, which is called when a flush operation occurs or a transaction is committed.

The following code shows the implementation used by the EmployeeOptimisticCallbackImpl object:

```
public Object getVersionedObjectForValue(Object value)
{
    if (value == null)
    {
        return null;
    }
    else
    {
        Employee emp = (Employee) value;
        return new Long( emp.getSequenceNumber() );
    }
}
```

As demonstrated in the previous example, the sequenceNumber attribute is returned in a java.lang.Long object as expected by the Loader, which implies that the same person that wrote the Loader either wrote the EmployeeOptimisticCallbackImpl implementation or worked closely with the person that implemented the EmployeeOptimisticCallbackImpl implementation. For example, these people agreed on the value that is returned by the getVersionedObjectForValue method. As previously described, the default OptimisticCallback implementation returns the special value NULL_OPTIMISTIC_VERSION as the version object.

updateVersionedObjectForValue method

The updateVersionedObjectForValue method is called when a transaction has updated a value and a new versioned object is needed. If the getVersionedObjectForValue method returns an attribute of the value, this method typically updates the attribute value with a new version object. If the getVersionedObjectForValue method returns a copy of the value, this method typically would not update. The default OptimisticCallback does not update because the default implementation of the getVersionedObjectForValue method always returns the special value NULL_OPTIMISTIC_VERSION as the version object. The following example shows the implementation used by the EmployeeOptimisticCallbackImpl object that is used in the OptimisticCallback section:

```
public void updateVersionedObjectForValue(Object value)
{
    if ( value != null )
    {
        Employee emp = (Employee) value;
        long next = emp.getSequenceNumber() + 1;
        emp.updateSequenceNumber( next );
    }
}
```

As demonstrated in the previous example, the sequenceNumber attribute is increments by one so that the next time the getVersionedObjectForValue method is called, the java.lang.Long value that is returned has a long value that is the original sequence number value. Plus one, for example, is the next version value for this employee instance. Again, this example implies that the same person that wrote the Loader either wrote EmployeeOptimisticCallbackImpl implementation or worked closely with the person that implemented the EmployeeOptimisticCallbackImpl implementation.

serializeVersionedValue method

This method writes the versioned value to the specified stream. Depending on the implementation, the versioned value can be used to identify optimistic update collisions. In some implementations, the versioned value is a copy of the original value. Other implementations might have a sequence number or some other object to indicate the version of the value. Because the actual implementation is unknown, this method is provided to perform the proper serialization. The default implementation calls the writeObject method.

inflateVersionedValue method

This method takes the serialized version of the versioned value and returns the actual versioned value object. Depending on the implementation, the versioned value can be used to identify optimistic update collisions. In some implementations, the versioned value is a copy of the original value. Other implementations might have a sequence number or some other object to indicate the version of the value. Because the actual implementation is unknown, this method is provided to perform the proper deserialization. The default implementation calls the readObject method.

Using an application-provided OptimisticCallback implementation

You have two approaches to add an application-provided OptimisticCallback into the BackingMap configuration: programmatic configuration and XML configuration.

Programmatically plug in an OptimisticCallback implementation

The following example demonstrates how an application can programmatically plug in an OptimisticCallback object for the employee backing map in the grid1 ObjectGrid instance:

```
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogManager.createObjectGrid( "grid1" );
BackingMap bm = dg.defineMap("employees");
EmployeeOptimisticCallbackImpl cb = new EmployeeOptimisticCallbackImpl();
bm.setOptimisticCallback( cb );
```

XML configuration approach to plug in an OptimisticCallback implementation

The EmployeeOptimisticCallbackImpl object in the preceding example must implement the OptimisticCallback interface. The application can also use an XML file to plug in its OptimisticCallback object as shown in the following example:

```
<?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="grid1">
        <backingMap name="employees" pluginCollectionRef="employees" lockStrategy="OPTIMISTIC" />
    </objectGrid>
</objectGrids>

<backingMapPluginCollections>
    <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxsplugtrans_employees">
        <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxsplugtrans_OptimisticCallback"
className="com.xyz.EmployeeOptimisticCallbackImpl" />
    </backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

- Transaction processing overview
  WebSphere eXtreme Scale uses transactions as its mechanism for interaction with data.
- Introduction to plug-in slots
  A plug-in slot is a transactional storage space that is reserved for plug-ins that share transactional context. These slots provide a way for eXtreme Scale plug-ins to communicate with each other, share transactional context, and ensure that transactional resources are used correctly and consistently within a transaction.
- External transaction managers
  Typically, eXtreme Scale transactions begin with the Session.begin method and end with the Session.commit method. However, when an ObjectGrid is embedded, an external transaction coordinator can start and end transactions. In this case, you do not need to call the begin or commit methods.
- WebSphereTransactionCallback plug-in
  When you use the WebSphereTransactionCallback plug-in, enterprise applications that are running in a WebSphere Application Server environment can manage ObjectGrid transactions. This plug-in is deprecated. Use the WebSphere eXtreme Scale resource adapter instead.

Transaction processing overview

WebSphere® eXtreme Scale uses transactions as its mechanism for interaction with data.

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.

- 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.
- 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.
- 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.
- 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.
- Deadlocks
  Deadlocks can occur when two transactions try to update the same cache entry.
- 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.
- 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.
- 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.
- JMS for distributed transaction changes
  Use Java™ Message Service (JMS) for distributed transaction changes between different tiers or in environments on mixed platforms.

Related tasks:
Resolving lock timeout exceptions

Introduction to plug-in slots

A plug-in slot is a transactional storage space that is reserved for plug-ins that share transactional context. These slots provide a way for eXtreme Scale plug-ins to communicate with each other, share transactional context, and ensure that transactional resources are used correctly and consistently within a transaction.

A plug-in can store transactional context, such as database connection, Java™ Message Service (JMS) connection, and so on, in a plug-in slot. The stored transactional context can be retrieved by any plug-in that knows the plug-in slot number, which serves as the key to retrieve transactional context.

Using plug-in slots

Plug-in slots are part of the TxID Interface. See the API documentation for more information about the interface.The slots are entries on an ArrayList array. Plug-ins can reserve an entry in the ArrayList array by calling the ObjectGrid.reserveSlot method and indicating that it wants a slot on all TxID objects. After the slots are reserved, plug-ins can put transactional context into slots of each TxID object and retrieve the context later. The put and get operations are coordinated by slot numbers that are returned by the ObjectGrid.reserveSlot method.

A plug-in typically has a life cycle. Using plug-in slots has to fit into the life cycle of plug-in. Typically, the plug-in must reserve plug-in slots during the initialization stage and obtain slot numbers for each slot. During normal run time, the plug-in puts transactional context into the reserved slot in the TxID object at the appropriate point. This appropriate point is typically at the beginning of the transaction. The plug-in or other plug-ins can then get the stored transactional context by the slot number from the TxID within the transaction.

The plug-in typically performs a clean up by removing transactional context and the slots. The following snippet of code illustrates how to use plug-in slots in a TransactionCallback plug-in:

```
public class DatabaseTransactionCallback implements TransactionCallback {
    int connectionSlot;
    int autoCommitConnectionSlot;
    int psCacheSlot;
    Properties ivProperties = new Properties();

    public void initialize(ObjectGrid objectGrid) throws TransactionCallbackException {
        // In initialization stage, reserve desired plug-in slots by calling the
        //reserveSlot method of ObjectGrid and
```

```
        // passing in the designated slot name, TxID.SLOT_NAME.
        // Note: you have to pass in this TxID.SLOT_NAME that is designated
        // for application.
        try {
            // cache the returned slot numbers
            connectionSlot = objectGrid.reserveSlot(TxID.SLOT_NAME);
            psCacheSlot = objectGrid.reserveSlot(TxID.SLOT_NAME);
            autoCommitConnectionSlot = objectGrid.reserveSlot(TxID.SLOT_NAME);
        } catch (Exception e) {
        }
    }

    public void begin(TxID tx) throws TransactionCallbackException {
        // put transactional contexts into the reserved slots at the
        // beginning of the transaction.
        try {
            Connection conn = null;
            conn = DriverManager.getConnection(ivDriverUrl, ivProperties);
            tx.putSlot(connectionSlot, conn);
            conn = DriverManager.getConnection(ivDriverUrl, ivProperties);
            conn.setAutoCommit(true);
            tx.putSlot(autoCommitConnectionSlot, conn);
            tx.putSlot(psCacheSlot, new HashMap());
        } catch (SQLException e) {
            SQLException ex = getLastSQLException(e);
            throw new TransactionCallbackException("unable to get connection", ex);
        }
    }

    public void commit(TxID id) throws TransactionCallbackException {
        // get the stored transactional contexts and use them
        // then, clean up all transactional resources.
        try {
            Connection conn = (Connection) id.getSlot(connectionSlot);
            conn.commit();
            cleanUpSlots(id);
        } catch (SQLException e) {
            SQLException ex = getLastSQLException(e);
            throw new TransactionCallbackException("commit failure", ex);
        }
    }

    void cleanUpSlots(TxID tx) throws TransactionCallbackException {
        closePreparedStatements((Map) tx.getSlot(psCacheSlot));
        closeConnection((Connection) tx.getSlot(connectionSlot));
        closeConnection((Connection) tx.getSlot(autoCommitConnectionSlot));
    }

    /**
     * @param map
     */
    private void closePreparedStatements(Map psCache) {
        try {
            Collection statements = psCache.values();
            Iterator iter = statements.iterator();
            while (iter.hasNext()) {
                PreparedStatement stmt = (PreparedStatement) iter.next();
                stmt.close();
            }
        } catch (Throwable e) {
        }

    }

    /**
     * Close connection and swallow any Throwable that occurs.
     *
     * @param connection
     */
    private void closeConnection(Connection connection) {
        try {
            connection.close();
        } catch (Throwable e1) {
        }
    }

    public void rollback(TxID id) throws TransactionCallbackException
        // get the stored transactional contexts and use them
        // then, clean up all transactional resources.
        try {
            Connection conn = (Connection) id.getSlot(connectionSlot);
            conn.rollback();
            cleanUpSlots(id);
        } catch (SQLException e) {
        }
    }

    public boolean isExternalTransactionActive(Session session) {
        return false;
    }
```

```
    // Getter methods for the slot numbers, other plug-in can obtain the slot numbers
    // from these getter methods.

    public int getConnectionSlot() {
        return connectionSlot;
    }
    public int getAutoCommitConnectionSlot() {
        return autoCommitConnectionSlot;
    }
    public int getPreparedStatementSlot() {
        return psCacheSlot;
    }
```

The following snippet of code illustrates how a Loader can get the stored transactional context that is put by previous TransactionCallback plug-in example:

```
public class DatabaseLoader implements Loader
{
    DatabaseTransactionCallback tcb;
    public void preloadMap(Session session, BackingMap backingMap) throws LoaderException
    {
        // The preload method is the initialization method of the Loader.
        // Obtain interested plug-in from Session or ObjectGrid instance.
        tcb =
                        (DatabaseTransactionCallback)session.getObjectGrid().getTransactionCallback();
    }
    public List get(TxID txid, List keyList, boolean forUpdate) throws LoaderException
    {
        // get the stored transactional contexts that is put by tcb's begin method.
        Connection conn = (Connection)txid.getSlot(tcb.getConnectionSlot());
        // implement get here
        return null;
    }
    public void batchUpdate(TxID txid, LogSequence sequence) throws LoaderException,
                        OptimisticCollisionException
    {
        // get the stored transactional contexts that is put by tcb's begin method.
        Connection conn = (Connection)txid.getSlot(tcb.getConnectionSlot());
        // implement batch update here ...
    }
}
```

External transaction managers

Typically, eXtreme Scale transactions begin with the Session.begin method and end with the Session.commit method. However, when an ObjectGrid is embedded, an external transaction coordinator can start and end transactions. In this case, you do not need to call the begin or commit methods.

External transaction coordination

The TransactionCallback plug-in is extended with the isExternalTransactionActive(Session session) method that associates the eXtreme Scale session with an external transaction. The method header follows:

```
public synchronized boolean isExternalTransactionActive(Session session)
```

For example, eXtreme Scale can be set up to integrate with WebSphere® Application Server and WebSphere Extended Deployment.

Also, eXtreme Scale provides a built in plug-in called the WebSphere Plug-ins for managing transaction life cycle events, which describes how to build the plug-in for WebSphere Application Server environments, but you can adapt the plug-in for other frameworks.

The key to this seamless integration is the exploitation of the ExtendedJTATransaction API in WebSphere Application Server Version 6.1 . However, if you are using WebSphere Application Server Version 6.1, you must apply APAR PK07848 to support this method. Use the following sample code to associate an ObjectGrid session with a WebSphere Application Server transaction ID:

```
/**
* This method is required to associate an objectGrid session with a WebSphere
* Application Server transaction ID.
*/
Map/**/ localIdToSession;
public synchronized boolean isExternalTransactionActive(Session session)
{
    // remember that this localid means this session is saved for later.
    localIdToSession.put(new Integer(jta.getLocalId()), session);
    return true;
}
```

Retrieve an external transaction

Sometimes you might need to retrieve an external transaction service object for the TransactionCallback plug-in to use. In the WebSphere Application Server server, look up the ExtendedJTATransaction object from its namespace as shown in the following example:

```
public J2EETransactionCallback() {
    super();
    localIdToSession = new HashMap();
```

```
    String lookupName="java:comp/websphere/ExtendedJTATransaction";
    try
    {
        InitialContext ic = new InitialContext();
        jta = (ExtendedJTATransaction)ic.lookup(lookupName);
        jta.registerSynchronizationCallback(this);
    }
    catch(NotSupportedException e)
    {
        throw new RuntimeException("Cannot register jta callback", e);
    }
    catch(NamingException e){
        throw new RuntimeException("Cannot get transaction object");
    }
}
```

For other products, you can use a similar approach to retrieve the transaction service object.

Control commit by external callback

The TransactionCallback plug-in must receive an external signal to commit or roll back the eXtreme Scale session. To receive this external signal, use the callback from the external transaction service. Implement the external callback interface and register it with the external transaction service. For example, with WebSphere Application Server, implement the SynchronizationCallback interface, as shown in the following example:

```
public class J2EETransactionCallback implements
        com.ibm.websphere.objectgrid.plugins.TransactionCallback, SynchronizationCallback {
   public J2EETransactionCallback() {
      super();
      String lookupName="java:comp/websphere/ExtendedJTATransaction";
      localIdToSession = new HashMap();
      try {
         InitialContext ic = new InitialContext();
         jta = (ExtendedJTATransaction)ic.lookup(lookupName);
         jta.registerSynchronizationCallback(this);
      } catch(NotSupportedException e) {
         throw new RuntimeException("Cannot register jta callback", e);
      }
      catch(NamingException e) {
         throw new RuntimeException("Cannot get transaction object");
      }
   }

   public synchronized void afterCompletion(int localId, byte[] arg1,boolean didCommit) {
      Integer lid = new Integer(localId);
      // find the Session for the localId
      Session session = (Session)localIdToSession.get(lid);
      if(session != null) {
                try {
            // if WebSphere Application Server is committed when
            // hardening the transaction to backingMap.
            // We already did a flush in beforeCompletion
            if(didCommit) {
               session.commit();
            } else {
               // otherwise rollback
               session.rollback();
            }
         } catch(NoActiveTransactionException e) {
            // impossible in theory
         } catch(TransactionException e) {
            // given that we already did a flush, this should not fail
         } finally {
            // always clear the session from the mapping map.
            localIdToSession.remove(lid);
         }
      }
   }

   public synchronized void beforeCompletion(int localId, byte[] arg1) {
      Session session = (Session)localIdToSession.get(new Integer(localId));
      if(session != null) {
         try {
            session.flush();
         } catch(TransactionException e) {
            // WebSphere Application Server  does not formally define
            // a way to signal the
            // transaction has failed so do this
            throw new RuntimeException("Cache flush failed", e);
         }
      }
   }
}
```

Use eXtreme Scale APIs with the TransactionCallback plug-in

The TransactionCallback plug-in disables autocommit in eXtreme Scale. The normal usage pattern for an eXtreme Scale follows:

```
Session ogSession = ...;
ObjectMap myMap = ogSession.getMap("MyMap");
ogSession.begin();
MyObject v = myMap.get("key");
v.setAttribute("newValue");
myMap.update("key", v);
ogSession.commit();
```

When this TransactionCallback plug-in is in use, eXtreme Scale assumes that the application uses the eXtreme Scale when a container-managed transaction is present. The previous code snippet changes the following code in this environment:

```
public void myMethod() {
   UserTransaction tx = ...;
   tx.begin();
   Session ogSession = ...;
   ObjectMap myMap = ogSession.getMap("MyMap");
   yObject v = myMap.get("key");
   v.setAttribute("newValue");
   myMap.update("key", v);
   tx.commit();
}
```

The myMethod method is similar to a Web application scenario. The application uses the normal UserTransaction interface to begin, commit, and roll back transactions. The eXtreme Scale automatically begins and commits around the container transaction. If the method is an Enterprise JavaBeans (EJB) method that uses the TX_REQUIRES attribute, then remove the UserTransaction reference and the calls to begin and commit transactions and the method works the same way. In this case, the container is responsible for starting and ending the transaction.

WebSphereTransactionCallback plug-in

When you use the WebSphereTransactionCallback plug-in, enterprise applications that are running in a WebSphere® Application Server environment can manage ObjectGrid transactions. This plug-in is deprecated. Use the WebSphere eXtreme Scale resource adapter instead.

8.5+ The WebSphereTransactionCallback interface has been replaced by the WebSphere eXtreme Scale resource adapter, which enables Java Transaction API (JTA) transaction management. You can install this resource adapter on WebSphere Application Server or other Java Platform, Enterprise Edition (Java EE) application servers. The WebSphereTransactionCallback plug-in is not an enlisted JTA API, and therefore, is not designed to roll back the JTA transaction if the commit fails.

When you are using an ObjectGrid session within a method that is configured to use container-managed transactions, the enterprise container automatically begins, commits or rolls back the ObjectGrid transaction. When you are using Java™ Transaction API (JTA) UserTransaction objects, the ObjectGrid transaction is managed by the UserTransaction object automatically.

For a detailed discussion of the implementation of this plug-in, see External transaction managers.

Note: The ObjectGrid does not support 2-phase, XA transactions. This plug-in does not enlist the ObjectGrid transaction with the transaction manager. Therefore, if the ObjectGrid fails to commit, any other resources that are managed by the XA transaction do not roll back.

Programmatically plug in the WebSphereTransactionCallback object

You can enable the WebSphereTransactionCallback into the ObjectGrid configuration with programmatic configuration or XML configuration. The following code snippet uses the application to create the WebSphereTransactionCallback object and add it to an ObjectGrid:

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid", false);
WebSphereTransactionCallback wsTxCallback= new WebSphereTransactionCallback ();
myGrid.setTransactionCallback(wsTxCallback);
```

XML configuration approach to plug in the WebSphereTransactionCallback object

The following XML configuration creates the WebSphereTransactionCallback object and adds it to an ObjectGrid. The following text must be in the myGrid.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">
            <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxsplugweb_TransactionCallback"
classNam=
        "com.ibm.websphere.objectgrid.plugins.builtins.WebSphereTransactionCallback" />

        </objectGrid>
    </objectGrids>
</objectGridConfig>
```

Programming to use the OSGi framework

You can start eXtreme Scale servers and clients in an OSGi container, which allows you to dynamically add and update eXtreme Scale plug-ins to the runtime environment.

- Building eXtreme Scale dynamic plug-ins
  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.
- Upgrading agents and data models dynamically from OSGi bundles in the Liberty profile
  Upgrade your OSGi bundles without interruption in the Liberty profile.

Related concepts:
OSGi framework overview
Serializer programming overview
Serialization overview
Samples
Related tasks:
Installing the Eclipse Equinox OSGi framework with Eclipse Gemini for clients and servers
Updating OSGi services for eXtreme Scale plug-ins with xscmd
Avoiding object inflation when updating and retrieving cache data
Related information:
DataSerializer API documentation

Building eXtreme Scale dynamic plug-ins

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 BackingMapLifecyleListener 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 callsed 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 callsed 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.

Related tasks:
Upgrading agents and data models dynamically from OSGi bundles in the Liberty profile

Programming for JPA integration

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.

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.

- JPA Loaders
  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.
- Developing client-based JPA loaders
  You can implement preloading and reloading of data in your application with a Java Persistence API (JPA) utility. This capability can simplify loading the maps when the queries to the database cannot be partitioned.
- Example: Using the Hibernate plug-in to preload data into the ObjectGrid cache
  You can use the preload method of the ObjectGridHibernateCacheProvider class to preload data into the ObjectGrid cache for an entity class.
- Starting the JPA time-based updater
  When you start the Java Persistence API (JPA) time-based updater, the ObjectGrid maps are updated with the latest changes in the database.

Related concepts:
Loader considerations in a multi-master topology
Database integration: Write-behind, in-line, and side caching
Plug-ins for communicating with databases
Related tasks:
Troubleshooting loaders
Configuring JPA loaders

JPA Loaders

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.

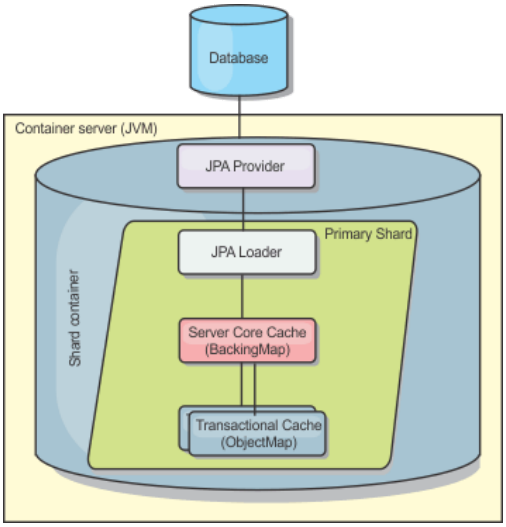
The JPALoader com.ibm.websphere.objectgrid.jpa.JPALoader and the JPAEntityLoader com.ibm.websphere.objectgrid.jpa.JPAEntityLoader plug-ins are two built-in JPA loader plug-ins that are used to synchronize the ObjectGrid maps with a database. You must have a JPA implementation, such as Hibernate or OpenJPA, 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 1. 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 details, see JPAEntityLoader plug-in.For more information, see JPAEntityLoader plug-in.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. For more information about configuring JPA loaders, see Configuring JPA loaders. For more information about programming JPA loaders, see JPA loader programming considerations.
Related tasks:
Developing client-based JPA loaders
Starting the JPA time-based updater
Related reference:
Example: Using the Hibernate plug-in to preload data into the ObjectGrid cache

Developing client-based JPA loaders

You can implement preloading and reloading of data in your application with a Java Persistence API (JPA) utility. This capability can simplify loading the maps when the queries to the database cannot be partitioned.

Before you begin

- You must be using a JPA provider with a supported database.
- Before you preload or reload maps, you must set the availability state of the ObjectGrid to `PRELOAD`. You can set the availability state with the setObjectGridState method of the StateManager interface. The StateManager interface prevents other clients from accessing the ObjectGrid when it is not yet online. After you preload or reload the map, you can set the state back to ONLINE.

- When you are preloading different maps in one ObjectGrid, set the ObjectGrid state to `PRELOAD` one time and set the value back to `ONLINE` after all maps finish data loading. This coordination can be done by the ClientLoadCallback interface. Set the ObjectGrid state to `PRELOAD` after the first preStart notification from the ObjectGrid instance, and set it back to `ONLINE` after the last postFinish notification.
- If you need to preload maps from different Java™ virtual machines, you have to coordinate between multiple Java virtual machines. Set the ObjectGrid state to `PRELOAD` one time before the first map is being preloaded in any of the Java virtual machines, and set the value back to `ONLINE` after all maps finish data loading in all the Java virtual machines. For more information, see Managing ObjectGrid availability.

About this task

When you run a preload or reload operation on your map, the following actions occur:

1. The initial action that is taken depends on if you are running a preload or reload operation.
   - **Preload operation:** The map to be preloaded is cleared. For an entity map, if any relation is configured as cascade-remove, any related maps are cleared.
   - **Reload operation:** The provided query is run on the map and the results are invalidated. For an entity map, if any relation is configured with the CascadeType.INVALIDATE option, the related entities are also invalidated from their maps.
2. Run the query to JPA for the entities in a batch.
3. For each batch, a key list and value list for each partition is built.
4. For each partition, the data grid agent is called to insert or update the data on the server side directly if it is an eXtreme Scale client. If the data grid is a local instance, the data in the maps is directly inserted or updated.

- Client-based JPA preload utility overview
  The client-based Java Persistence API (JPA) preload utility loads data into eXtreme Scale backing maps using a client connection to the ObjectGrid.
- Example: Preloading a map with the ClientLoader interface
  You can preload a map to populate the map data before clients begin accessing the map.
- Example: Reloading a map with the ClientLoader interface
  Reloading a map is the same as preloading a map, except that the isPreload argument is set to false in the ClientLoader.load method.
- Example: Calling a client loader
  You can use the preload method in the Loader interface to call a client loader.
- Example: Creating a custom client-based JPA loader
  The ClientLoader.load method in the Loader interface provides a client load function that satisfies most scenarios. However, if you want to load data without the ClientLoader.load method, you can implement your own preload utility.
- Developing a client-based JPA loader with a DataGrid agent
  When loading from the client side, using a DataGrid agent could increase performance. By using a DataGrid agent, all the data reads and writes occur in the server process. You can also design your application to make sure that DataGrid agents on multiple partitions run in parallel to further boost performance.

Related concepts:
JPA Loaders
Client-based JPA preload utility overview
Related tasks:
Starting the JPA time-based updater
Related reference:
Example: Using the Hibernate plug-in to preload data into the ObjectGrid cache
Example: Preloading a map with the ClientLoader interface
Example: Reloading a map with the ClientLoader interface
Example: Calling a client loader
Related information:
Interface ClientLoader
Interface StateManager

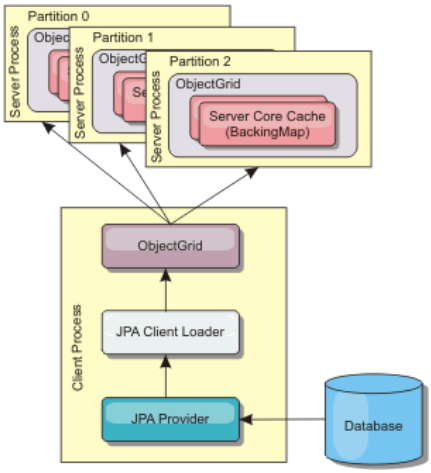
Client-based JPA preload utility overview

The client-based Java™ Persistence API (JPA) preload utility loads data into eXtreme Scale backing maps using a client connection to the ObjectGrid.

This capability can simplify loading the maps when the queries to the database cannot be partitioned. A loader, such as a JPA Loader can also be used and is ideal when the data can be loaded in parallel.

The client-based JPA preload utility can use either the OpenJPA or Hibernate JPA implementations to load the ObjectGrid from a database. Because WebSphere® eXtreme Scale does not directly interact with the database or Java Database Connectivity (JDBC), any database that OpenJPA or Hibernate supports can be used to load the ObjectGrid.

Figure 1. Client loader that uses JPA implementation to load the ObjectGrid



Typically, a user application provides a persistence unit name, an entity class name, and a JPA query to the client loader. The client loader retrieves the JPA entity manager based on the persistence unit name, uses the entity manager to query data from the database with the provided entity class and JPA query, and finally loads the data into the distributed ObjectGrid maps. When multi-level relations are involved in the query, can use a custom query string to optimize the performance. Optionally, a persistence property map could be provided to override the configured persistence properties.

A client loader can load data in two different modes, as displayed in the following table:

Table 1. Client loader modes

Mode	Description
Preload	Clears and loads all entries into the backing map. If the map is an entity map, any related entity maps will also be cleared if the ObjectGrid CascadeType.REMOVE option is enabled.
Reload	The JPA query is executed against the ObjectGrid to invalidate all the entities in the map that match the query. If the map is an entity map, any related entity maps will also be cleared if the ObjectGrid CascadeType.INVALIDATE option is enabled.

In either case, a JPA query is used to select and load the desired entities from the database and to store them in the ObjectGrid maps. If the ObjectGrid map is a non-entity map, the JPA entities will be detached and stored directly. If the ObjectGrid map is an entity map, the JPA entities are stored as ObjectGrid entity tuples. You can provide a JPA query or use the default query `select o from EntityName o`.

For more information about configuring the client-based JPA preload utility, see Developing client-based JPA loaders

Related tasks:
Developing a client-based JPA loader with a DataGrid agent
Developing client-based JPA loaders
Related reference:
Example: Preloading a map with the ClientLoader interface
Example: Reloading a map with the ClientLoader interface
Example: Calling a client loader
Example: Creating a custom client-based JPA loader
Example: Preloading a map with the ClientLoader interface
Example: Reloading a map with the ClientLoader interface
Example: Calling a client loader
Related information:
Interface ClientLoader
Interface StateManager

Example: Preloading a map with the ClientLoader interface

You can preload a map to populate the map data before clients begin accessing the map.

Client-based preload example

The following sample code snippet shows a simple client loading. In this example, the CUSTOMER map is configured as an entity map. The Customer entity class, which is configured in the ObjectGrid entity metadata descriptor XML file, has a one-to-many relation with Order entities. The Customer entity has the CascadeType.ALL option enabled on the relation to the Order entity. Before the ClientLoader.load method is called, the ObjectGrid state is set to PRELOAD. The isPreload parameter on the load method is set to `true`.

```
// Get the StateManager
StateManager stateMgr = StateManagerFactory.getStateManager();

// Set ObjectGrid state to PRELOAD before calling ClientLoader.loader
stateMgr.setObjectGridState(AvailabilityState.PRELOAD, objectGrid);

ClientLoader c = ClientLoaderFactory.getClientLoader();

// Load the data
c.load(objectGrid, "CUSTOMER", "customerPU", null,
```

```
    null, null, null, true, null);

// Set ObjectGrid state back to ONLINE
stateMgr.setObjectGridState(AvailabilityState.ONLINE, objectGrid);
```

Related concepts:
Client-based JPA preload utility overview
Client-based JPA preload utility overview
Related tasks:
Developing a client-based JPA loader with a DataGrid agent
Developing client-based JPA loaders
Related reference:
Example: Reloading a map with the ClientLoader interface
Example: Calling a client loader
Example: Creating a custom client-based JPA loader
Related information:
Interface ClientLoader
Interface StateManager

Example: Reloading a map with the ClientLoader interface

Reloading a map is the same as preloading a map, except that the isPreload argument is set to false in the ClientLoader.load method.

Client-based reload example

The following sample shows how to reload maps. Compared to the preload sample, the main difference is that a loadSql and parameters are provided. This sample only reloads the Customer data with an ID between `1000` and `2000`. The isPreload parameter on the load method is set to `false`.

```
// Get the StateManager
StateManager stateMgr = StateManagerFactory.getStateManager();

// Set ObjectGrid state to PRELOAD before calling ClientLoader.loader
stateMgr.setObjectGridState(AvailabilityState.PRELOAD, objectGrid);

ClientLoader c = ClientLoaderFactory.getClientLoader();

// Load the data
String loadSql = "select c from CUSTOMER c
    where c.custId >= :startCustId and c.custId < :endCustId ";
Map<String, Long> params = new HashMap<String, Long>();
params.put("startCustId", 1000L);
params.put("endCustId", 2000L);

c.load(objectGrid, "CUSTOMER", "customerPU", null, null,
    loadSql, params, false, null);

// Set ObjectGrid state back to ONLINE
stateMgr.setObjectGridState(AvailabilityState.ONLINE, objectGrid);
```

Remember: This query string observes both JPA query syntax and eXtreme Scale entity query syntax. This query string is important because it runs twice: to invalidate the matched ObjectGrid entities and to load the matched JPA entities.
Related concepts:
Client-based JPA preload utility overview
Client-based JPA preload utility overview
Related tasks:
Developing a client-based JPA loader with a DataGrid agent
Developing client-based JPA loaders
Related reference:
Example: Preloading a map with the ClientLoader interface
Example: Calling a client loader
Example: Creating a custom client-based JPA loader
Related information:
Interface ClientLoader
Interface StateManager

Example: Calling a client loader

You can use the preload method in the Loader interface to call a client loader.

Use the preload method in the Loader interface to call a client loader:

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

This method signals the loader to preload the data into the map. A loader implementation can use a client loader to preload the data to all its partitions. For example, the JPA loader uses the client loader to preload data into the map. See Client-based JPA preload utility overview for more information.

Example: Calling a client loader with the preloadMap method

An example of how to preload the map using the client loader in the preloadMap method follows. The example first checks whether the current partition number is the same as the preload partition. If the partition number is not the same as the preload partition, no action occurs. If the partition numbers match, the client loader is called to load data into the maps. You must call the client loader in one and only one partition.

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

....
        ObjectGrid objectGrid = session.getObjectGrid();
        int partitionId = backingMap.getPartitionId();
        int numPartitions = backingMap.getPartitionManager().getNumOfPartitions();

        // Only call client loader data in one partition
        if (partitionId == preloadPartition) {
          ClientLoader c = ClientLoaderFactory.getClientLoader();
    // Call the client loader to load the data
    try {
        c.load(objectGrid, "CUSTOMER", "customerPU",
                                        null, entityClass, null, null, true, null);
    } catch (ObjectGridException e) {
        LoaderException le = new LoaderException("Exception caught in ObjectMap " +
                                        ogName + "." + mapName);
        le.initCause(e);
        throw le;
    }
        }
}
```

Remember: Configure the backingMap attribute "preloadMode" to `true`, so the preload method runs asynchronously. Otherwise, the preload method blocks the ObjectGrid instance from being activated.

Related concepts:
Client-based JPA preload utility overview
Client-based JPA preload utility overview
Related tasks:
Developing a client-based JPA loader with a DataGrid agent
Developing client-based JPA loaders
Related reference:
Example: Preloading a map with the ClientLoader interface
Example: Reloading a map with the ClientLoader interface
Example: Creating a custom client-based JPA loader
Related information:
Interface ClientLoader
Interface StateManager

Example: Creating a custom client-based JPA loader

The ClientLoader.load method in the Loader interface provides a client load function that satisfies most scenarios. However, if you want to load data without the ClientLoader.load method, you can implement your own preload utility.

Custom loader template

Use the following template to develop your loader:

```
// Get the StateManager
StateManager stateMgr = StateManagerFactory.getStateManager();

// Set ObjectGrid state to PRELOAD before calling ClientLoader.loader
stateMgr.setObjectGridState(AvailabilityState.PRELOAD, objectGrid);


// Load the data
...<your preload utility implementation>...

// Set ObjectGrid state back to ONLINE
stateMgr.setObjectGridState(AvailabilityState.ONLINE, objectGrid);
```

Related concepts:
Client-based JPA preload utility overview
Related tasks:
Developing a client-based JPA loader with a DataGrid agent
Related reference:
Example: Preloading a map with the ClientLoader interface
Example: Reloading a map with the ClientLoader interface
Example: Calling a client loader

Developing a client-based JPA loader with a DataGrid agent

When loading from the client side, using a DataGrid agent could increase performance. By using a DataGrid agent, all the data reads and writes occur in the server process. You can also design your application to make sure that DataGrid agents on multiple partitions run in parallel to further boost performance.

About this task

For more information about the DataGrid agent, see DataGrid APIs and partitioning.
After you create the data preload implementation, you can create a generic Loader to complete the following tasks:

- Query the data from database in batches.
- Build a key list and value list for each partition.
- For each partition, call the agentMgr.callReduceAgent(agent, aKey) method to run the agent in the server in a thread. By running in a thread, you can run agents concurrently on multiple partitions.

Example

The following snippet of code is an example of how to load using a DataGrid agent:

```
import java.io.Externalizable;
import java.io.IOException;
import java.io.ObjectInput;
import java.io.ObjectOutput;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;
import java.util.List;

import com.ibm.websphere.objectgrid.NoActiveTransactionException;
import com.ibm.websphere.objectgrid.ObjectGridException;
import com.ibm.websphere.objectgrid.ObjectGridRuntimeException;
import com.ibm.websphere.objectgrid.ObjectMap;
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.TransactionException;
import com.ibm.websphere.objectgrid.datagrid.ReduceGridAgent;
import com.ibm.websphere.objectgrid.em.EntityManager;

public class InsertAgent implements ReduceGridAgent, Externalizable {

    private static final long serialVersionUID = 6568906743945108310L;

    private List keys = null;

    private List vals = null;

    protected boolean isEntityMap;

    public InsertAgent() {
    }

    public InsertAgent(boolean entityMap) {
        isEntityMap = entityMap;
    }

    public Object reduce(Session sess, ObjectMap map) {
        throw new UnsupportedOperationException(
            "ReduceGridAgent.reduce(Session, ObjectMap)");
    }

    public Object reduce(Session sess, ObjectMap map, Collection arg2) {
        Session s = null;
        try {
            s = sess.getObjectGrid().getSession();
            ObjectMap m = s.getMap(map.getName());
            s.beginNoWriteThrough();
            Object ret = process(s, m);
            s.commit();
            return ret;
        } catch (ObjectGridRuntimeException e) {
            if (s.isTransactionActive()) {
                try {
                    s.rollback();
                } catch (TransactionException e1) {
                } catch (NoActiveTransactionException e1) {
                }
            }
            throw e;
        } catch (Throwable t) {
            if (s.isTransactionActive()) {
                try {
                    s.rollback();
                } catch (TransactionException e1) {
                } catch (NoActiveTransactionException e1) {
```

```
                }
            }
            throw new ObjectGridRuntimeException(t);
        }

    }

    public Object process(Session s, ObjectMap m) {
        try {

            if (!isEntityMap) {
                // In the POJO case, it is very straightforward,
                // we can just put everything in the
                // map using insert
                insert(m);
            } else {
                // 2. Entity case.
                // In the Entity case, we can persist the entities
                EntityManager em = s.getEntityManager();
                persistEntities(em);

            }
            return Boolean.TRUE;
        } catch (ObjectGridRuntimeException e) {
            throw e;
        } catch (ObjectGridException e) {
            throw new ObjectGridRuntimeException(e);
        } catch (Throwable t) {
            throw new ObjectGridRuntimeException(t);
        }

    }

    /**
     * Basically this is fresh load.
     * @param s
     * @param m
     * @throws ObjectGridException
     */
    protected void insert(ObjectMap m) throws ObjectGridException {

        int size = keys.size();

        for (int i = 0; i < size; i++) {
            m.insert(keys.get(i), vals.get(i));
        }

    }


    protected void persistEntities(EntityManager em) {
        Iterator<Object> iter = vals.iterator();

        while (iter.hasNext()) {
            Object value = iter.next();
            em.persist(value);
        }
    }

    public Object reduceResults(Collection arg0) {
        return arg0;
    }

    public void readExternal(ObjectInput in)
      throws IOException, ClassNotFoundException {
        int v = in.readByte();
        isEntityMap = in.readBoolean();
        vals = readList(in);
        if (!isEntityMap) {
            keys = readList(in);
        }
    }

    public void writeExternal(ObjectOutput out) throws IOException {
        out.write(1);
        out.writeBoolean(isEntityMap);

        writeList(out, vals);
        if (!isEntityMap) {
            writeList(out, keys);
        }

    }

    public void setData(List ks, List vs) {
        vals = vs;
        if (!isEntityMap) {
            keys = ks;
        }
    }
```

```
    /**
     * @return Returns the isEntityMap.
     */
    public boolean isEntityMap() {
        return isEntityMap;
    }

    static public void writeList(ObjectOutput oo, Collection l)
      throws IOException {
        int size = l == null ? -1 : l.size();
        oo.writeInt(size);
        if (size > 0) {
            Iterator iter = l.iterator();
            while (iter.hasNext()) {
                Object o = iter.next();
                oo.writeObject(o);
            }
        }
    }

    public static List readList(ObjectInput oi)
      throws IOException, ClassNotFoundException {
        int size = oi.readInt();
        if (size == -1) {
            return null;
        }


        ArrayList list = new ArrayList(size);
        for (int i = 0; i < size; ++i) {
            Object o = oi.readObject();
            list.add(o);
        }
        return list;
    }
}
```

Related concepts:
Client-based JPA preload utility overview
Related reference:
Example: Preloading a map with the ClientLoader interface
Example: Reloading a map with the ClientLoader interface
Example: Calling a client loader
Example: Creating a custom client-based JPA loader

Example: Using the Hibernate plug-in to preload data into the ObjectGrid cache

You can use the preload method of the ObjectGridHibernateCacheProvider class to preload data into the ObjectGrid cache for an entity class.

Example: Using the EntityManagerFactory class

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("testPU");
ObjectGridHibernateCacheProvider.preload("objectGridName", emf, TargetEntity.class, 100, 100);
```

Important: By default, entities are not part of the second level cache. In the Entity classes that need caching, add the @cache annotation. An example follows:

```
import org.hibernate.annotations.Cache;
import org.hibernate.annotations.CacheConcurrencyStrategy;
@Entity
@Cache(usage=CacheConcurrencyStrategy.TRANSACTIONAL)
public class HibernateCacheTest { ... }
```

You can override this default by setting the shared-cache-mode element in your persistence.xml file or by using the javax.persistence.sharedCache.mode property.

Example: Using the SessionFactory class

```
org.hibernate.cfg.Configuration cfg = new Configuration();
// use addResource, addClass, and setProperty method of Configuration to prepare
// configuration required to create SessionFactor
SessionFactory sessionFactory= cfg.buildSessionFactory();
ObjectGridHibernateCacheProvider.preload("objectGridName", sessionFactory,
TargetEntity.class, 100, 100);
```

Note:

1. In a distributed system, this preload mechanism can only be invoked from one Java™ virtual machine. The preload mechanism cannot run simultaneously from multiple Java virtual machines.
2. Before running the preload, you must initialize the eXtreme Scale cache by creating EntityManager using EntityManagerFactory to have all corresponding BackingMaps created; otherwise, the preload forces the cache to be initialized with only one default BackingMap to support all entities. This means a single BackingMap is shared by all entities.

Related concepts:
JPA Loaders
Related tasks:
Developing client-based JPA loaders
Starting the JPA time-based updater

Starting the JPA time-based updater

When you start the Java™ Persistence API (JPA) time-based updater, the ObjectGrid maps are updated with the latest changes in the database.

Before you begin

Configure the time-based updater. See Configuring a JPA time-based data updater.

About this task

For more information about how the Java Persistence API (JPA) time-based data updater works, see JPA time-based data updater.

Procedure

- Start a time-based database updater.
  - **Automatically for distributed eXtreme Scale:**
    If you create the timeBasedDBUpdate configuration for the backing map, the time-based database updater is automatically started when a distributed ObjectGrid primary shard is activated. For an ObjectGrid that has multiple partitions, the time-based database updater only starts at partition 0.

  - **Automatically for local eXtreme Scale:**
    If you create the timeBasedDBUpdate configuration for the backing map, the time-based database updater is automatically started when the local map is activated.

  - **Manually:**
    You can also manually start or stop a time-based database updater using the TimeBasedDBUpdater API.

    ```
    public synchronized void startDBUpdate(ObjectGrid objectGrid, String mapName,
            String punitName, Class entityClass, String timestampField, DBUpdateMode mode) {
    ```

    1. **ObjectGrid**: the ObjectGrid instance (local or client).
    2. **mapName**: the name of the backing map for which the time-based database updater is started.
    3. **punitName**: the JPA persistence unit name for creating a JPA entity manager factory; the default value is the name of the first persistence unit defined in the persistence.xml file.
    4. **entityClass**: The entity class name used to interact with the Java Persistence API (JPA) provider; the entity class name is used to get JPA entities using entity queries.
    5. **timestampField**: A timestamp field of the entity class to identify the time or sequence when a database back end record was last updated or inserted.
    6. **mode**: The time-based database update mode; an INVALIDATE_ONLY type causes it to invalidate the entries in the ObjectGrid map if the corresponding records in the database have changed; an UPDATE_ONLY type indicates to update the existing entries in the ObjectGrid map with the latest values from the database; however, all the newly inserted records to the database are ignored; an INSERT_UPDATE type indicates to update the existing entries in the ObjectGrid map with the latest values from the database; also, all the newly inserted records to the database are inserted into the ObjectGrid map.

    If you want to stop the time-based database updater, you can call the following method to stop the updater:

    ```
    public synchronized void stopDBUpdate(ObjectGrid objectGrid, String mapName)
    ```

    The ObjectGrid and mapName parameters should be the same as those passed in the startDBUpdate method.

- Create the timestamp field in your database.
  - **DB2®**
    As a part of the optimistic locking feature, DB2 9.5 provides a row change timestamp feature. You can define a column ROWCHGTS using the ROW CHANGE TIMESTAMP format as follows:

    ```
    ROWCHGTS TIMESTAMP NOT NULL
         GENERATED ALWAYS
         FOR EACH ROW ON UPDATE AS
         ROW CHANGE TIMESTAMP
    ```

    Then you can indicate the entity field which corresponds to this column as the timestamp field by either annotation or configuration. An example follows:

    ```
    @Entity(name = "USER_DB2")
    @Table(name = "USER1")
    public class User_DB2 implements Serializable {

        private static final long serialVersionUID = 1L;

        public User_DB2() {
        }
    ```

```
    public User_DB2(int id, String firstName, String lastName) {
        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
    }

    @Id
    @Column(name = "ID")
    public int id;

    @Column(name = "FIRSTNAME")
    public String firstName;

    @Column(name = "LASTNAME")
    public String lastName;

    @com.ibm.websphere.objectgrid.jpa.dbupdate.annotation.Timestamp
    @Column(name = "ROWCHGTS", updatable = false, insertable = false)
    public Timestamp rowChgTs;
}
```

- **Oracle**
  In Oracle, there is a pseudo-column `ora_rowscn` for the system change number of the record. You can use this column for the same purpose. An example of the entity that uses the ora_rowscn field as the time-based database update timestamp field follows:

```
@Entity(name = "USER_ORA")
@Table(name = "USER1")
public class User_ORA implements Serializable  {

    private static final long serialVersionUID = 1L;

    public User_ORA() {
    }

    public User_ORA(int id, String firstName, String lastName) {
        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
    }

    @Id
    @Column(name = "ID")
    public int id;

    @Column(name = "FIRSTNAME")
    public String firstName;

    @Column(name = "LASTNAME")
    public String lastName;

    @com.ibm.websphere.objectgrid.jpa.dbupdate.annotation.Timestamp
    @Column(name = "ora_rowscn", updatable = false, insertable = false)
    public long rowChgTs;
}
```

- **Other databases**
  For other types of databases, you can create a table column to track the changes. The column values have to be manually managed by the application that updates the table.

  Take an Apache Derby database as an example: You can create a column "ROWCHGTS" to track the change numbers. Also, a latest change number is tracked for this table. Every time a record is inserted or updated, the latest change number for the table is incremented, and the ROWCHGTS column value for the record is updated with this incremented number.

```
@Entity(name = "USER_DER")
@Table(name = "USER1")
public class User_DER implements Serializable {

    private static final long serialVersionUID = 1L;

    public User_DER() {
    }

    public User_DER(int id, String firstName, String lastName) {
        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
    }

    @Id
    @Column(name = "ID")
    public int id;

    @Column(name = "FIRSTNAME")
    public String firstName;

    @Column(name = "LASTNAME")
    public String lastName;
```

```
    @com.ibm.websphere.objectgrid.jpa.dbupdate.annotation.Timestamp
    @Column(name = "ROWCHGTS", updatable = true, insertable = true)
    public long rowChgTs;
}
```

- JPA time-based data updater
  A Java Persistence API (JPA) time-based database updater updates the ObjectGrid maps with the latest changes in the database.

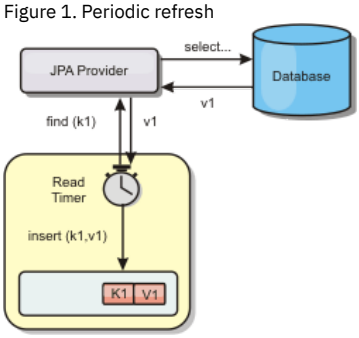
Related concepts:
JPA Loaders
Related tasks:
Developing client-based JPA loaders
Related reference:
Example: Using the Hibernate plug-in to preload data into the ObjectGrid cache

JPA time-based data updater

A Java™ Persistence API (JPA) time-based database updater updates the ObjectGrid maps with the latest changes in the database.

When changes are made directly to a database that is being fronted by WebSphere® eXtreme Scale, those changes are not concurrently reflected in the eXtreme Scale grid. To properly implement eXtreme Scale as an in-memory database processing space, take into consideration that your grid can get out of sync with the database. Time-based database updater uses the System Change Number (SCN) capability in Oracle 10g and row change timestamp in DB2® 9.5 to monitor changes in the database for invalidation and update. The updater also allows applications to have a user-defined field for the same purpose.

Figure 1. Periodic refresh



The time-based database updater periodically queries the database using JPA interfaces to get the JPA entities that represent the newly inserted and updated records in the database. To periodically update the records, every record in the database should have a timestamp to identify the time or sequence in which the record was last updated or inserted. It is not required that the timestamp be in a timestamp format. The timestamp value can be in an integer or long format, if it generates a unique, increasing value.

Several commercial databases have provided this capability.

For example, in DB2 9.5, you can define a column using the ROW CHANGE TIMESTAMP format as follows:

```
ROWCHGTS TIMESTAMP NOT NULL
      GENERATED ALWAYS
      FOR EACH ROW ON UPDATE AS
      ROW CHANGE TIMESTAMP
```

In Oracle, you can use the pseudo-column **ora_rowscn**, which represents the system change number of the record.

The time-based database updater updates the ObjectGrid maps in three different ways:

1. INVALIDATE_ONLY. Invalidate the entries in the ObjectGrid map if the corresponding records in the database have changed.
2. UPDATE_ONLY. Update the entries in the ObjectGrid map if the corresponding records in the database have changed. However, all the newly inserted records to the database are ignored.
3. INSERT_UPDATE. Update the existing entries in the ObjectGrid map with the latest values from the database. Also, all the newly inserted records to the database are inserted into the ObjectGrid map.

For more information about configuring the JPA time-based data updater, see Configuring a JPA time-based data updater.

Developing applications with the Spring framework

Learn how to integrate your eXtreme Scale applications with the popular Spring framework.

- Spring framework overview
  Spring is a framework for developing Java™ applications. WebSphere® eXtreme Scale provides support to allow Spring to manage transactions and configure the clients and servers comprising your deployed in-memory data grid.
- Managing transactions with Spring
  Spring is a popular framework for developing Java applications. WebSphere eXtreme Scale provides support to allow Spring to manage eXtreme Scale transactions and configure eXtreme Scale clients and servers.

- Spring managed extension beans
  You can declare plain old Java objects (POJOs) to use as extension points in the objectgrid.xml file. If you name the beans and then specify the class name, eXtreme Scale normally creates instances of the specified class and uses those instances as the plug-in. WebSphere eXtreme Scale can now delegate to Spring to act as the bean factory for obtaining instances of these plug-in objects.
- Spring extension beans and namespace support
  WebSphere eXtreme Scale provides a feature to declare plain old Java objects (POJOs) to use as extension points in the objectgrid.xml file and a way to name the beans and then specify the class name. Normally, instances of the specified class are created, and those objects are used as the plug-ins. Now, eXtreme Scale can delegate to Spring to obtain instances of these plug-in objects. If an application uses Spring then typically such POJOs have a requirement to be wired in to the rest of the application.
- Starting a container server with Spring
  You can start a container server using Spring managed extension beans and namespace support.
- Configuring clients in the Spring framework
  You can override client-side ObjectGrid settings with the Spring Framework.

Related concepts:
Spring framework overview
Spring extension beans and namespace support
Related reference:
Spring managed extension beans
Spring descriptor XML file
Spring objectgrid.xsd file

Spring framework overview

Spring is a framework for developing Java™ applications. WebSphere® eXtreme Scale provides support to allow Spring to manage transactions and configure the clients and servers comprising your deployed in-memory data grid.

8.5+

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. For more information, see Configuring a Spring cache provider.

Spring managed native transactions

Spring provides container-managed transactions that are similar to a Java Platform, Enterprise Edition application server. However, the Spring mechanism can use different implementations. WebSphere eXtreme Scale provides transaction manager integration which allows Spring to manage the ObjectGrid transaction life cycles. For more information, see Managing transactions with Spring.

Spring managed extension beans and namespace support

Also, eXtreme Scale integrates with Spring to allow Spring-style beans defined for extension points or plug-ins. This feature provides more sophisticated configurations and more flexibility for configuring the extension points.

In addition to Spring managed extension beans, eXtreme Scale provides a Spring namespace called "objectgrid". Beans and built-in implementations are pre-defined in this namespace, which makes it easier for users to configure eXtreme Scale. See Spring extension beans and namespace support for more details on these topics and a sample of how to start an eXtreme Scale container server using Spring configurations.

Shard scope support

With the traditional style Spring configuration, an ObjectGrid bean can either be a singleton type or prototype type. ObjectGrid also supports a new scope called the "shard" scope. If a bean is defined as shard scope, then only one bean is created per shard. All requests for beans with an ID or IDs matching that bean definition in the same shard results in that one specific bean instance being returned by the Spring container.

The following example shows that a com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl bean is defined with scope set to shard. Therefore, only one instance of the JPAPropFactoryImpl class is created per shard.

```
<bean id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxssprgfrmwk_2_jpaPropFactory"
class="com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl" scope="shard" />
```

Spring Web Flow

Spring Web Flow stores its session state in an HTTP session by default. If a web application uses eXtreme Scale for session management, then Spring automatically stores state with eXtreme Scale. Also, fault tolerance is enabled in the same manner as the session.

For more information, see HTTP session management.

Packaging

The eXtreme Scale Spring extensions are in the ogspring.jar file. This Java archive (JAR) file must be on the class path for Spring support to work. If a Java EE application that is running in a WebSphere Extended Deployment augmented WebSphere Application Server Network Deployment, put the spring.jar file and its associated files in the enterprise archive (EAR) modules. You must also place the ogspring.jar file in the same location.

Related tasks:
Configuring a Spring cache provider
Developing applications with the Spring framework
Starting a container server with Spring
Managing transactions with Spring
Related reference:
ObjectGrid descriptor XML file
Deployment policy descriptor XML file
Spring managed extension beans
Spring descriptor XML file
Spring objectgrid.xsd file

Managing transactions with Spring

Spring is a popular framework for developing Java™ applications. WebSphere® eXtreme Scale provides support to allow Spring to manage eXtreme Scale transactions and configure eXtreme Scale clients and servers.

About this task

The Spring Framework is highly integrable with eXtreme Scale, as discussed in the following sections.

Procedure

- **Native transactions:** Spring provides container-managed transactions along the style of a Java Platform, Enterprise Edition application server but has the advantage that Springs mechanism can have different implementations plugged in. This topic describes an eXtreme Scale Platform Transaction manager that can be used with Spring. This allows programmers to annotate their POJOs (plain old Java objects) and then have Spring automatically acquire Sessions from eXtreme Scale and begin, commit, rollback, suspend, and resume eXtreme Scale transactions. Spring transactions are described more fully in Chapter 10 of the official Spring reference documentation. The following explains how to create an eXtreme Scale transaction manager and use it with annotated POJOs. It also explains how to use this approach with client or local eXtreme Scale as well as a collocated Data Grid style application.
- **Transaction manager:** To work with Spring,, eXtreme Scale provides an implementation of a Spring PlatformTransactionManager. This manager can provide managed eXtreme Scale sessions to POJOs managed by Spring. Through the use of annotations, Spring manages those sessions for the POJOs in terms of transaction life cycle. The following XML snippet shows how to create a transaction Manager:

```
<aop:aspectj-autoproxy/>
        <tx:annotation-driven transaction-manager="transactionManager"/>

        <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txspringint_ObjectGridManager"
                                class="com.ibm.websphere.objectgrid.ObjectGridManagerFactory"
                                factory-method="getObjectGridManager"/>

        <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txspringint_ObjectGrid"
                                factory-bean="ObjectGridManager"
                                factory-method="createObjectGrid"/>

        <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txspringint_transactionManager"
                                class="com.ibm.websphere.objectgrid.spring.ObjectGridSpringFactory"
                                factory-method="getLocalPlatformTransactionManager"/>
        </bean>

        <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txspringint_Service"
class="com.ibm.websphere.objectgrid.spring.test.TestService">
                <property name="txManager" ref+"transactionManager"/>
        </bean>
```

  This shows the transactionManager bean being declared and wired in to the Service bean that will use Spring transactions. We will demonstrate this using annotations and this is the reason for the tx:annotation clause at the beginning.
- **Obtaining an ObjectGrid session:** A POJO that has methods managed by Spring can now obtain the ObjectGrid session for the current transaction using

```
Session s = txManager.getSession();
```

  This returns the session for the POJO to use. Beans participating in the same transaction will receive the same session when they call this method. Spring will automatically handle begin for the Session and also automatically invoke commit or rollback when necessary. You can obtain an ObjectGrid EntityManager also by simply calling getEntityManager from the Session object.
- **Setting the ObjectGrid instance for a thread:** A single Java Virtual Machine (JVM) can host many ObjectGrid instances. Each primary shard placed in a JVM has its own ObjectGrid instance. A JVM acting as a client to a remote ObjectGrid uses an ObjectGrid instance returned from the connect method's ClientClusterContext to interact with that Grid. Before invoking a method on a POJO using Spring transactions for ObjectGrid, the thread must be primed with the ObjectGrid instance to use. The TransactionManager instance has a method allowing a specific ObjectGrid instance to be specified. Once specified then any subsequent txManager.getSession calls will returns Sessions for that ObjectGrid instance.
  The following example shows a sample main for exercising this capability:

```
ClassPathXmlApplicationContext ctx = new ClassPathXmlApplicationContext(new String[]
            {"applicationContext.xml"});
    SpringLocalTxManager txManager = (SpringLocalTxManager)ctx.getBean("transactionManager");
    txManager.setObjectGridForThread(og);

    ITestService s = (ITestService)ctx.getBean("Service");
    s.initialize();
    assertEquals(s.query(), "Billy");
    s.update("Bobby");
    assertEquals(s.query(), "Bobby");
    System.out.println("Requires new test");
    s.testRequiresNew(s);
    assertEquals(s.query(), "1");
```

Here we use a Spring ApplicationContext. The ApplicationContext is used to obtain a reference to the txManager and specify an ObjectGrid to use on this thread. The code then obtains a reference to the service and invokes methods on it. Each method call at this level causes Spring to create a Session and do begin/commit calls around the method call. Any exceptions will cause a rollback.

- **SpringLocalTxManager interface:** The SpringLocalTxManager interface is implemented by the ObjectGrid Platform Transaction Manager and has all public interfaces. The methods on this interface are for selecting the ObjectGrid instance to use on a thread and obtaining a Session for the thread. Any POJOs using ObjectGrid local transactions should be injected with a reference to this manager instance and only a single instance need be created, that is, its scope should be singleton. This instance is created using a static method on ObjectGridSpringFactory. getLocalPlatformTransactionManager(). Restriction: WebSphere eXtreme Scale does not support JTA or two-phase commit for various reasons mainly to do with scalability. Thus, except at a last single-phase participant, ObjectGrid does not interact in XA or JTA type global transactions. This platform manager is intended to make using local ObjectGrid transactions as easy as possible for Spring developers.

Related concepts:
Spring framework overview
Spring extension beans and namespace support
Related reference:
Spring managed extension beans
Spring descriptor XML file
Spring objectgrid.xsd file

Spring managed extension beans

You can declare plain old Java objects (POJOs) to use as extension points in the objectgrid.xml file. If you name the beans and then specify the class name, eXtreme Scale normally creates instances of the specified class and uses those instances as the plug-in. WebSphere® eXtreme Scale can now delegate to Spring to act as the bean factory for obtaining instances of these plug-in objects.

If an application uses Spring, POJOs have a requirement to be accessible to the rest of the application.
An application can register a Spring Bean Factory instance to use for an ObjectGrid specified by name. The application creates an instance of BeanFactory or a Spring application context and then registers it with ObjectGrid using the following static method:

```
void registerSpringBeanFactoryAdapter(String objectGridName, Object springBeanFactory)
```

The previous method applies to the case when eXtreme Scale finds an extension bean whose className begins with the prefix {spring}. Such an extension bean, which could be an ObjectTransformer, Loader, TransactionCallback, and so on, uses the remainder of the name as a Spring Bean name. Then it obtains the bean instance using the Spring Bean Factory.

The eXtreme Scale deployment environment can also create a Spring bean factory from a default Spring XML configuration file. If no bean factory was registered for a given ObjectGrid, then your deployment searches for an XML file called "/<ObjectGridName>_spring.xml" automatically. For example, if your data grid is called GRID, then the XML file is called "/GRID_spring.xml' and appears in the class path in the root package. ObjectGrid constructs an ApplicationContext using the "/<ObjectGridName>_spring.xml file and constructs beans from that bean factory.

The following is an example class name:

```
"{spring}MyLoaderBean"
```

Using the previous class name allows eXtreme Scale to use Spring to search for a bean named "MyLoaderBean". You can specify Spring-managed POJOs for any extension point if the bean factory has been registered. The Spring extensions are in the ogspring.jar file. This JAR file must be on the class path for Spring support. If a J2EE application runs in WebSphere Application Server Network Deployment augmented with WebSphere Extended Deployment, then you must place the applicaitonhe application should place the spring.jar file and its associated files in the EAR modules. The ogspring.jar must also be placed in the same location.
Related concepts:
Spring framework overview
Spring extension beans and namespace support
Related tasks:
Developing applications with the Spring framework
Starting a container server with Spring
Managing transactions with Spring

Spring extension beans and namespace support

WebSphere® eXtreme Scale provides a feature to declare plain old Java™ objects (POJOs) to use as extension points in the objectgrid.xml file and a way to name the beans and then specify the class name. Normally, instances of the specified class are created, and those objects are used as the plug-ins. Now, eXtreme

Scale can delegate to Spring to obtain instances of these plug-in objects. If an application uses Spring then typically such POJOs have a requirement to be wired in to the rest of the application.

In some scenarios, you must use Spring to configure a plug-in, as in the following example:

```
<objectGrid name="Grid">
    <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxssprngxbns_TransactionCallback"
className="com.ibm.websphere.objectgrid.jpa.JPATxCallback">
        <property name="persistenceUnitName" type="java.lang.String"  value="employeePU" />
    </bean>
    ...
</objectGrid>
```

The built-in TransactionCallback implementation, the com.ibm.websphere.objectgrid.jpa.JPATxCallback class, is configured as the TransactionCallback class. This class is configured with the persistenceUnitName property as shown in the previous example. The JPATxCallback class also has the JPAPropertyFactory attribute, which is of type java.lang.Object. The ObjectGrid XML configuration cannot support this type of configuration.

The eXtreme Scale Spring integration solves this problem by delegating the bean creation to the Spring framework. The revised configuration follows:

```
<objectGrid name="Grid">
    <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxssprngxbns_TransactionCallback"
className="{spring}jpaTxCallback"/>
    ...
</objectGrid>
```

The spring file for the "Grid" object contains the following information:

```
<bean id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxssprngxbns_jpaTxCallback"
class="com.ibm.websphere.objectgrid.jpa.JPATxCallback" scope="shard">
    <property name="persistenceUnitName" value="employeeEMPU"/>
    <property name="JPAPropertyFactory" ref ="jpaPropFactory"/>
</bean>

<bean id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxssprngxbns_jpaPropFactory"
class="com.ibm.ws.objectgrid.jpa.plugins.
JPAPropFactoryImpl" scope="shard">
</bean>
```

Here, the TransactionCallback is specified as `{spring}jpaTxCallback`, and the jpaTxCallback and jpaPropFactory bean are configured in the spring file as shown in the previous example. The Spring configuration makes it possible to configure a JPAPropertyFactory bean as a parameter of the JPATxCallback object.

Default Spring bean factory

When eXtreme Scale finds a plug-in or an extension bean (such as an ObjectTransformer, Loader, TransactionCallback, and so on) with a classname value that begins with the prefix `{spring}`, then eXtreme Scale uses the remainder of the name as a Spring Bean name and obtain the bean instance using the Spring Bean Factory.

By default, if no bean factory was registered for a given ObjectGrid, then it tries to find an ObjectGridName_spring.xml file. For example, if your data grid is called "Grid" then the XML file is called /Grid_spring.xml. This file should be in the class path or in a META-INF directory which is in the class path. If this file is found, then eXtreme Scale constructs an ApplicationContext using that file and constructs beans from that bean factory.

Custom Spring bean factory

WebSphere eXtreme Scale also provides an ObjectGridSpringFactory API to register a Spring Bean Factory instance to use for a specific named ObjectGrid. This API registers an instance of BeanFactory with eXtreme Scale using the following static method:

```
void registerSpringBeanFactoryAdapter(String objectGridName, Object springBeanFactory)
```

Namespace support

Since version 2.0, Spring has a mechanism for schema-based extensions to the basic Spring XML format for defining and configuring beans. ObjectGrid uses this new feature to define and configure ObjectGrid beans. With Spring XML schema extension, some of the built-in implementations of eXtreme Scale plug-ins and some ObjectGrid beans are predefined in the "objectgrid" namespace. When writing the Spring configuration files, you do not have to specify the full class name of the built-in implementations. Instead, you can reference the predefined beans.

Also, with the attributes of the beans defined in the XML schema, you are less likely to provide a wrong attribute name. XML validation based on the XML schema can catch these kind of errors earlier in the development cycle.

These beans defined in the XML schema extensions are:

- transactionManager
- register
- server
- catalog
- catalogServerProperties
- container
- JPALoader
- JPATxCallback
- JPAEntityLoader
- LRUEvictor
- LFUEvictor

- HashIndex

These beans are defined in the objectgrid.xsd XML schema. This XSD file is shipped as com/ibm/ws/objectgrid/spring/namespace/objectgrid.xsd file in the ogspring.jar file . For detailed descriptions of the XSD file and the beans defined in the XSD file, see Spring descriptor XML file.

Use the JPATxCallback example from the previous section. In the previous section, the JPATxCallback bean is configured as the following:

```
<bean id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxssprngxbns_jpaTxCallback"
class="com.ibm.websphere.objectgrid.jpa.JPATxCallback" scope="shard">
    <property name="persistenceUnitName" value="employeeEMPU"/>
    <property name="JPAPropertyFactory" ref ="jpaPropFactory"/>
</bean>

<bean id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxssprngxbns_jpaPropFactory"
class="com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl" scope="shard">
</bean>
```

Using this namespace feature, the spring XML configuration can be written as the following:

```
<objectgrid:JPATxCallback
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxssprngxbns_jpaTxCallback"
persistenceUnitName="employeeEMPU"
        jpaPropertyFactory="jpaPropFactory" />

<bean id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxssprngxbns_jpaPropFactory"
class="com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl"
        scope="shard">
</bean>
```

Notice here that instead of specifying the com.ibm.websphere.objectgrid.jpa.JPATxCallback class as in the previous example, we directly use the pre-defined objectgrid:JPATxCallback bean. As you can see, this configuration is less verbose and more friendly to error checking.

For a description of working with Spring beans, consult Starting a container server with Spring.

Related tasks:
Developing applications with the Spring framework
Starting a container server with Spring
Managing transactions with Spring
Related reference:
Spring managed extension beans
Spring descriptor XML file
Spring objectgrid.xsd file

Starting a container server with Spring

You can start a container server using Spring managed extension beans and namespace support.

About this task

With several XML files configured for Spring, you can start basic eXtreme Scale container servers.

Procedure

1. **ObjectGrid XML file:**
   First of all, define a very simple ObjectGrid XML file which contains one ObjectGrid "Grid" and one map "Test". The ObjectGrid has an ObjectGridEventListener plug-in called "partitionListener", and the map "Test" has an Evictor plugged in called "testLRUEvictor". Notice both the ObjectGridEventListener plug-in and Evictor plug-in are configured using Spring as their names contain "{spring}".

   ```
   <?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="Grid">
               <bean
   id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txssprngcont_ObjectGridEventList
   ener" className="{spring}partitionListener" />
               <backingMap name="Test" pluginCollectionRef="test" />
           </objectGrid>
       </objectGrids>

       <backingMapPluginCollections>
           <backingMapPluginCollection
   id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txssprngcont_test">
               <bean
   id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txssprngcont_Evictor"
   className="{spring}testLRUEvictor"/>
           </backingMapPluginCollection>
       </backingMapPluginCollections>
   </objectGridConfig>
   ```

2. **ObjectGrid deployment XML file:**
   Now, create a simple ObjectGrid deployment XML file as follows. It partitions the ObjectGrid into 5 partitions, and no replica is required.

```
<?xml version="1.0" encoding="UTF-8"?>
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy ../deploymentPolicy.xsd"
 xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">
    <objectgridDeployment objectgridName="Grid">
        <mapSet name="mapSet" numInitialContainers="1" numberOfPartitions="5" minSyncReplicas="0"
            maxSyncReplicas="1" maxAsyncReplicas="0">
            <map ref="Test"/>
        </mapSet>
    </objectgridDeployment>
</deploymentPolicy>
```

3. **ObjectGrid Spring XML file:**
   Now we will use both ObjectGrid Spring managed extension beans and namespace support features to configure the ObjectGrid beans. The spring xml file is named Grid_spring.xml. Notice two schemas are included in the XML file: spring-beans-2.0.xsd is for using the Spring managed beans, and objectgrid.xsd is for using the beans predefined in the objectgrid namespace.

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"

xmlns:objectgrid="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txssprngcont_http:
//www.ibm.com/schema/objectgrid"
       xsi:schemaLocation="
       http://www.ibm.com/schema/objectgrid
                        http://www.ibm.com/schema/objectgrid/objectgrid.xsd
       http://www.springframework.org/schema/beans
                        http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <objectgrid:register
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txssprngcont_ogregister"
gridname="Grid"/>

    <objectgrid:server
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txssprngcont_server"
isCatalog="true" name="server">
        <objectgrid:catalog host="localhost" port="2809"/>
    </objectgrid:server>

    <objectgrid:container
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txssprngcont_container"
                        objectgridxml="com/ibm/ws/objectgrid/test/springshard/objectgrid.xml"
       deploymentxml="com/ibm/ws/objectgrid/test/springshard/deployment.xml"
                        server="server"/>

    <objectgrid:LRUEvictor
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txssprngcont_testLRUEvictor"
numberOfLRUQueues="31"/>

    <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txssprngcont_partitionListener"
                        class="com.ibm.websphere.objectgrid.springshard.ShardListener" scope="shard"/>
</beans>
```

   There were six beans defined in this spring XML file:

   a. *objectgrid:register*: This register the default bean factory for the ObjectGrid "Grid".
   b. *objectgrid:server*: This defines an ObjectGrid server with name "server". This server will also provide catalog service since it has an objectgrid:catalog bean nested in it.
   c. *objectgrid:catalog*: This defines an ObjectGrid catalog service endpoint, which is set to "localhost:2809".
   d. *objectgrid:container*: This defines an ObjectGrid container with specified objectgrid XML file and deployment XML file as we discussed before. The server property specifies which server this container is hosted in.
   e. *objectgrid:LRUEvictor*: This defines an LRUEvictor with the number of LRU queues to use set to 31.
   f. *bean partitionListener*: This defines a ShardListener plug-in. You must provide an implementation for this plug-in, so it cannot use the pre-defined beans. Also this scope of the bean is set to "shard", which means there is only one instance of this ShardListener per ObjectGrid shard.

4. **Starting the server:**
   The snippet below starts the ObjectGrid server, which hosts both the container service and the catalog service. As we can see, the only method we need to call to start the server is to get a bean "container" from the bean factory. This simplifies the programming complexity by moving most of the logic into Spring configuration.

```
public class ShardServer extends TestCase
{
    Container container;
    org.springframework.beans.factory.BeanFactory bf;

    public void startServer(String cep)
    {
        try
        {
            bf = new org.springframework.context.support.ClassPathXmlApplicationContext(
                "/com/ibm/ws/objectgrid/test/springshard/Grid_spring.xml", ShardServer.class);
            container = (Container)bf.getBean("container");
        }
```

```
        catch(Exception e)
        {
            throw new ObjectGridRuntimeException("Cannot start OG container", e);
        }
    }

    public void stopServer()
    {
        if(container != null)
            container.teardown();
    }
}
```

Related concepts:
Spring framework overview
Spring extension beans and namespace support
Related reference:
Spring managed extension beans
Spring descriptor XML file
Spring objectgrid.xsd file

Configuring clients in the Spring framework

You can override client-side ObjectGrid settings with the Spring Framework.

About this task

The following example XML file shows how to build an ObjectGridConfiguration element, and use it to override some client side settings. You can create a similar configuration using programmatic configuration or by configuring the ObjectGrid descriptor XML file.
8.5+ For information about how to use the ObjectGridClientBean and ObjectGridCatalogServiceDomainBean beans to support the Spring Framework Version 3.1 cache abstraction, see Configuring a Spring cache provider.

Procedure

1. Create an XML file to configure clients with the Spring framework.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
                "http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
  <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txscliconfigspring_companyGrid"
factory-bean="manager" factory-method="getObjectGrid"
    singleton="true">
    <constructor-arg type="com.ibm.websphere.objectgrid.ClientClusterContext">
      <ref bean="client" />
    </constructor-arg>
    <constructor-arg type="java.lang.String" value="CompanyGrid" />
</bean>

  <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txscliconfigspring_manager"
class="com.ibm.websphere.objectgrid.ObjectGridManagerFactory"
    factory-method="getObjectGridManager" singleton="true">
    <property name="overrideObjectGridConfigurations">
      <map>
        <entry key="DefaultDomain">
          <list>
            <ref bean="ogConfig" />
          </list>
        </entry>
      </map>
    </property>
  </bean>

  <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txscliconfigspring_ogConfig"
    class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
    factory-method="createObjectGridConfiguration">
    <constructor-arg type="java.lang.String">
      <value>CompanyGrid</value>
    </constructor-arg>
    <property name="plugins">
      <list>
          <bean class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
          factory-method="createPlugin">
          <constructor-arg type="com.ibm.websphere.objectgrid.config.PluginType"
            value="TRANSACTION_CALLBACK" />
          <constructor-arg type="java.lang.String"
            value="com.company.MyClientTxCallback" />
        </bean>
        <bean class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
```

```
          factory-method="createPlugin">
          <constructor-arg type="com.ibm.websphere.objectgrid.config.PluginType"
            value="OBJECTGRID_EVENT_LISTENER" />
          <constructor-arg type="java.lang.String" value="" />
        </bean>
      </list>
        </property>
      <property name="backingMapConfigurations">
        <list>
            <bean class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
            factory-method="createBackingMapConfiguration">
            <constructor-arg type="java.lang.String" value="Customer" />
              <property name="plugins">
                    <bean class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
                factory-method="createPlugin">
                <constructor-arg type="com.ibm.websphere.objectgrid.config.PluginType"
                        value="EVICTOR" />
                    <constructor-arg type="java.lang.String"
                value="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" />
              </bean>
            </property>
            <property name="numberOfBuckets" value="1429" />
            </bean>
            <bean class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
              factory-method="createBackingMapConfiguration">
                <constructor-arg type="java.lang.String" value="OrderLine" />
                  <property name="numberOfBuckets" value="701" />                        <property name="timeToLive"
value="800" />
                    <property name="ttlEvictorType">
                  <value type="com.ibm.websphere.objectgrid.

TTLType">LAST_ACCESS_TIME</value>
                    </property>
            </bean>
          </list>
        </property>
      </bean>

      <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txscliconfigspring_client"
factory-bean="manager" factory-method="connect"
        singleton="true">
        <constructor-arg type="java.lang.String">
            <value>localhost:2809</value>
        </constructor-arg>
          <constructor-arg
          type="com.ibm.websphere.objectgrid.security.
                                              config.ClientSecurityConfiguration">
          <null />
        </constructor-arg>
          <constructor-arg type="java.net.URL">
            <null />
        </constructor-arg>
      </bean>
</beans>
```

2. Load the XML file you created and build the ObjectGrid.

```
BeanFactory beanFactory = new XmlBeanFactory(newUrlResource
                ("file:test/companyGridSpring.xml"));
ObjectGrid companyGrid = (ObjectGrid) beanFactory.getBean("companyGrid");
```

   Read about the Spring framework overview for more information on creating an XML descriptor file.

Related concepts:
Java client overrides
Related reference:
ObjectGrid descriptor XML file
Client properties file

Monitoring

You can use the included monitoring console, APIs, MBeans, logs, and utilities to monitor the performance of your application environment.

- Statistics overview
  Statistics in WebSphere® eXtreme Scale are built on an internal statistics tree. The StatsAccessor API, Performance Monitoring Infrastructure (PMI) modules, and MBean API are built from the internal tree.
- Monitoring with the web console
  With the web console, you can chart current and historical statistics. This console provides some preconfigured charts for high-level overviews, and has a

custom reports page that you can use to build charts from the available statistics. You can use the charting capabilities in the monitoring console of WebSphere eXtreme Scale to view the overall performance of the data grids in your environment.

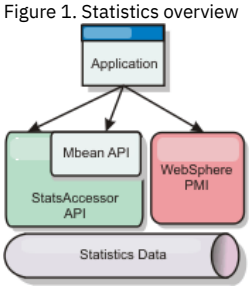
- Monitoring with CSV files
  You can enable monitoring data collected for a container server to be written to comma-separated values (CSV) files. These CSV files can contain information about the Java virtual machine (JVM), map, or ObjectGrid instance.
- Enabling statistics
  WebSphere eXtreme Scale uses an internal statistics model to track and filter data, which is the underlying structure that all data views use to gather snapshots of statistics. You can use several methods to retrieve the information from the statistics modules.
- Monitoring with the xscmd utility
  The **xscmd** utility replaces the **xsadmin** sample utility as a fully supported monitoring and administration tool. With the **xscmd** utility, you can display textual information about your WebSphere eXtreme Scale topology.
- Monitoring with WebSphere Application Server PMI
  WebSphere eXtreme Scale supports Performance Monitoring Infrastructure (PMI) when running in a WebSphere Application Server or WebSphere Extended Deployment application server. PMI collects performance data on runtime applications and provides interfaces that support external applications to monitor performance data. You can use the administrative console or the wsadmin tool to access monitoring data.
- Monitoring server statistics with managed beans (MBeans)
  You can used managed beans (MBeans) to track statistics in your environment.
- Monitoring with vendor tools
  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.
- Monitoring eXtreme Scale information in DB2
  When the JPALoader or JPAEntityLoader is used with DB2® as the back-end database, eXtreme Scale-specific information can be passed to DB2. You can view this information by a performance monitor tool such as DB2 Performance Expert to monitor the eXtreme Scale applications that are accessing the database.

Statistics overview

Statistics in WebSphere® eXtreme Scale are built on an internal statistics tree. The StatsAccessor API, Performance Monitoring Infrastructure (PMI) modules, and MBean API are built from the internal tree.

The following figure shows the general setup of statistics for WebSphere eXtreme Scale.

Figure 1. Statistics overview



Each of these APIs offer a view into the statistics tree, but are used for different reasons:

- **Statistics API**: The Statistics API allows developers direct access to statistics for flexible and customizable statistics integration solutions, such as custom MBeans or logging.
- **MBean API**: The MBean API is a specification-based mechanism for monitoring. The MBean API uses the Statistics API and runs local to the server Java™ Virtual Machine (JVM). The API and MBean structures are designed to readily integrate with other vendor utilities. Use the MBean API when you are running a distributed object grid.
- **WebSphere Application Server Performance Monitoring Infrastructure (PMI) modules**: Use PMI if you are running WebSphere eXtreme Scale within WebSphere Application Server. These modules provide a view of the internal statistics tree.

Statistics API

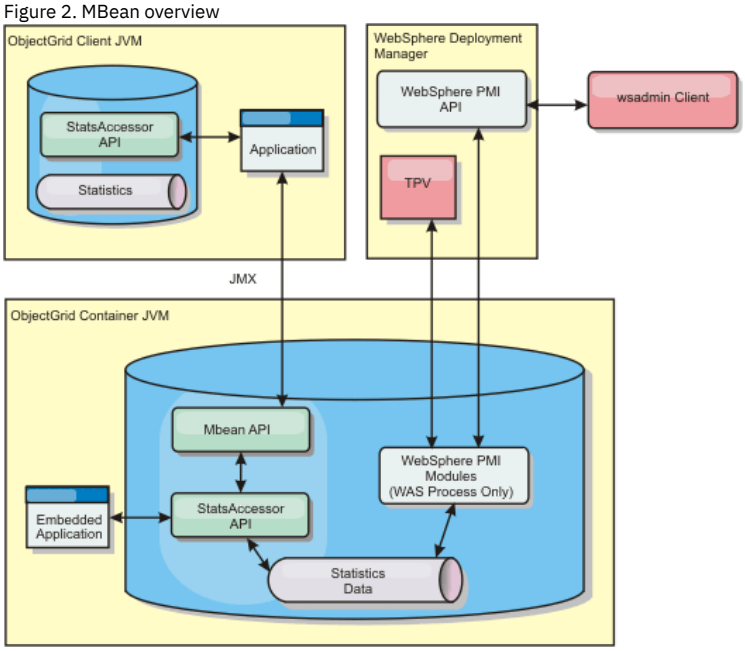
Much like a tree map, there is a corresponding path and key used to retrieve a specific module, or in this case granularity or aggregation level. For example, assume there is always an arbitrary root node in the tree and that statistics are being gathered for a map named "payroll," belonging to an ObjectGrid named "accounting." For example, to access the module for a map's aggregation level or granularity, you could pass in a String[] of the paths. In this case that would equate to String[] {root, "accounting", "payroll"}, as each String would represent the node's path. The advantage of this structure is that a user can specify the array to any node in the path and get the aggregation level for that node. So passing in String[] {root, "accounting"} would give you map statistics, but for the entire grid of "accounting." This leaves the user with both the ability to specify types of statistics to monitor, and at whatever level of aggregation is required for the application.

WebSphere Application Server PMI modules

WebSphere eXtreme Scale includes statistics modules for use with the WebSphere Application Server PMI. When a WebSphere Application Server profile is augmented with WebSphere eXtreme Scale, the augment scripts automatically integrate the WebSphere eXtreme Scale modules into the WebSphere Application Server configuration files. With PMI, you can enable and disable statistics modules, automatically aggregate statistics at various granularity, and even graph the data using the built-in Tivoli® Performance Viewer. See Monitoring with WebSphere Application Server PMI for more information.

Vendor product integration with Managed Beans (MBean)

The eXtreme Scale APIs and Managed Beans are designed to allow for easy integration with third party monitoring applications. JConsole or MC4J are some examples of lightweight Java Management Extensions (JMX) consoles that can be used to analyze information about an eXtreme Scale topology. You can also use the programmatic APIs to write adapter implementations to snapshot or track eXtreme Scale performance. WebSphere eXtreme Scale includes a sample monitoring application that allows out-of-the box monitoring capabilities, and can be used as a template for writing more advanced custom monitoring utilities.
Figure 2. MBean overview



See Sample: xsadmin utility for more information. For more information about integrating with specific vendor applications, see the following topics:

- Monitoring eXtreme Scale with IBM® Tivoli Monitoring agent
- Monitoring eXtreme Scale with Hyperic HQ
- Monitoring eXtreme Scale applications with CA Wily Introscope

Related concepts:
Monitoring with vendor tools
Related tasks:
Monitoring with the web console
Monitoring with CSV files
Enabling statistics
Monitoring with the xscmd utility
Monitoring with WebSphere Application Server PMI
Monitoring server statistics with managed beans (MBeans)
Monitoring eXtreme Scale information in DB2
Accessing Managed Beans (MBeans) using the wsadmin tool
Accessing Managed Beans (MBeans) programmatically
Monitoring server statistics with managed beans (MBeans)
Monitoring with the xscmd utility
Related reference:
Administering with Managed Beans (MBeans)
Related information:
API documentation: Package com.ibm.websphere.objectgrid.management
Interface PlacementServiceMBean

Monitoring with the web console

With the web console, you can chart current and historical statistics. This console provides some preconfigured charts for high-level overviews, and has a custom reports page that you can use to build charts from the available statistics. You can use the charting capabilities in the monitoring console of WebSphere® eXtreme Scale to view the overall performance of the data grids in your environment.

1. Starting and logging on to the web console
   Start the console server by running the **startConsoleServer** command and logging on to the server with the default user ID and password.
2. Connecting the web console to catalog servers
   To start viewing statistics in the web console, you must first connect to catalog servers that you want to monitor. Additional steps are required if your catalog servers have security enabled.
3. Viewing statistics with the web console
   You can monitor statistics and other performance information with the web console.
4. Monitoring with custom reports
   You can build custom reports to save various charts that contain statistics about the catalog service domains, data grids, and container servers in your

environment. You can save the custom reports and load them to view again later.

Related concepts:
Statistics overview
Monitoring with vendor tools
Related tasks:
Monitoring with CSV files
Enabling statistics
Monitoring with the xscmd utility
Monitoring with WebSphere Application Server PMI
Monitoring server statistics with managed beans (MBeans)
Monitoring eXtreme Scale information in DB2
Related reference:
Web console statistics
stopOgServer script
Related information:
Getting started tutorial lesson 4: Monitor your environment

Starting and logging on to the web console

Start the console server by running the **startConsoleServer** command and logging on to the server with the default user ID and password.

Before you begin

- **Web browser requirements**
  Use one of the following browsers with the web console:
  - Mozilla Firefox, version 3.5.x and later
  - Mozilla Firefox, version 3.6.x and later
  - Microsoft Internet Explorer, version 7 or 8

About this task

If your catalog servers are running in a stand-alone environment, with a Java™ security manager, then the user ID that you use to log in must correspond to a principal that has the following permissions in the Java security policy file:

```
grant principal javax.security.auth.x500.X500Principal "CN=manager,O=acme,OU=OGSample"
     {
        permission javax.management.MBeanPermission "*",
"getAttribute,setAttribute,invoke,queryNames,addNotificationListener,removeNotificationListener";
     };
```

Procedure

1. Optional: If you want to run your console server on a port other than the default port, edit the web console configuration. Click Settings > Configuration > System. The default port for the console server is `7080` for HTTP and `7443` for HTTPS. If you edit these values after the console server is already started, restart the server to use the new port numbers.
2. Start the console server. The **startConsoleServer.bat|sh** script for starting the console server is in the *wxs_install_root*/ObjectGrid/bin directory of your installation.
3. Log on to the console.
   a. From your web browser, go to `https://your.console.host:7443`, replacing `your.console.host` with the host name of the server onto which you installed the console. If you had changed the port number, remember to update the port number in the URL.
   b. Log on to the console.
      - **User ID:** `admin`
      - **Password:** `admin`

      The console welcome page is displayed.
4. Edit the console configuration. Click Settings > Configuration to review the console configuration. The console configuration includes information such as:
   - Trace string for the WebSphere® eXtreme Scale client, such as `*=all=disabled`
   - The Administrator name and password
   - The Administrator e-mail address

What to do next

- Connect your catalog servers to the web console to start tracking statistics. See Connecting the web console to catalog servers for more information.
- If you need to stop the web console server, run the **stopConsoleServer.bat|sh** script. This script is in the *wxs_install_root*/ObjectGrid/bin directory of your installation.

Next topic: Connecting the web console to catalog servers
Related reference:
Web console statistics
stopOgServer script
Related information:
Getting started tutorial lesson 4: Monitor your environment

Connecting the web console to catalog servers

To start viewing statistics in the web console, you must first connect to catalog servers that you want to monitor. Additional steps are required if your catalog servers have security enabled.

Before you begin

- The web console server must be running. See Starting and logging on to the web console for more information.
- You must have at least one catalog server running to which you want to connect. See Starting a stand-alone catalog service for more information.

Procedure

1. If your catalog servers have Secure Sockets Layer (SSL) enabled, you must configure a keystore and a truststore in the client properties file. You can enable SSL for a catalog server by setting the transportType attribute to `SSL-Required` or `SSL-Supported` in the server properties file. For more information about the SSL properties you can set on the server, see the Server properties file.
   a. Configure a keystore and truststore, and then exchange, or cross-import the public certificates. For example, you might copy the truststore and keystore to a location on the server that is running the web console.
   b. Edit the client properties file on the web console server to include the properties for SSL configuration. For more information about the SSL properties you can set for the client, see the Client properties file. For example, you can make a copy of the client properties file located in the*wxs_install_root*/ObjectGridProperties/sampleclient.properties and edit that file. The following properties are required for outbound SSL connections from the web console:

```
#-------------------------------------------------------------------------------
# SSL Configuration
#
# - contextProvider     (IBMJSSE2, IBMJSSE, IBMJSSEFIPS, etc.)
# - protocol            (SSL, SSLv2, SSLv3, TLS, TLSv1, etc.)
# - keyStoreType        (JKS, JCEK, PKCS12, etc.)
# - trustStoreType      (JKS, JCEK, PKCS12, etc.)
# - keyStore            (fully qualified path to key store file)
# - trustStore          (fully qualified path to trust store file)
# - alias               (string specifying ssl certificate alias to use from keyStore)
# - keyStorePassword    (string specifying password to the key store - encoded or not)
# - trustStorePassword  (string specifying password to the trust store - encoded or not)
#
# Uncomment these properties to set the SSL configuration.
#-------------------------------------------------------------------------------
#alias=clientprivate
#contextProvider=IBMJSSE
#protocol=SSL
#keyStoreType=JKS
#keyStore=etc/test/security/client.private
#keyStorePassword={xor}PDM2OjErLyg\=
#trustStoreType=JKS
#trustStore=etc/test/security/server.public
#trustStorePassword={xor}Lyo9MzY8
```

   Windows Important: If you are using Windows, you must escape any backslash (\ ) characters in the path. For example, if you want to use the path C:\opt\ibm, enter C:\\opt\\ibm in the properties file. Windows directories with spaces are not supported.
2. Establish and maintain connections to catalog servers that you want to monitor. Repeat the following steps to add each catalog server to the configuration.
   a. Click Settings > eXtreme Scale Catalog Servers.
   b. Add a new catalog server.
      i. Click the add icon () to register an existing catalog server.
      ii. Provide information, such as the host name and listener port. See Planning for network ports for more information about port configuration and defaults.
      iii. Click OK.
      iv. Verify that the catalog server has been added to the navigation tree.
3. Group the catalog servers that you created into a catalog service domain. You must create a catalog service domain when security is enabled in your catalog servers because security settings are configured in the catalog service domain.
   a. Click Settings > eXtreme Scale Domains page.
   b. Add a new catalog service domain.
      i. Click the add icon () to register a catalog service domain. Enter a name for the catalog service domain.
      ii. After you create the catalog service domain, you can edit the properties. The catalog service domain properties follow:

      Name
      : Indicates the host name of the domain, as assigned by the administrator.

      Catalog servers
      : Lists one or more catalog servers that belong to the selected domain. You can add the catalog servers that you created in the previous step.

      Generator class
      : Specifies the name of the class that implements the CredentialGenerator interface. This class is used to get credentials for clients. If you specify a value in this field, the value overrides the crendentialGeneratorClass property in the client.properties file.

      Generator properties
      : Specifies the properties for the CredentialGenerator implementation class. The properties are set to the object with the setProperties(String) method. The credentialGeneratorprops value is used only if the value of the credentialGeneratorClass property is not null. If you specify a value in this field, the value overrides the credentialGeneratorProps property in the client.properties file.

eXtreme Scale client properties path
Specifies the path to the client properties file that you edited to include SSL properties. For example, you might indicate the /ObjectGridProperties/sampleclient.properties file. If you want to stop the console from trying to use SSL connections, you can delete the value in this field. After you set the path, the console uses an unsecured connection.

iii. Click OK.
iv. Verify that the domain has been added to the navigation tree.

To view information about an existing catalog service domain, click the name of the catalog service domain in the navigation tree on the Settings > eXtreme Scale Domains page.

4. View the connection status. The Current domain field indicates the name of the catalog service domain that is currently being used to display information in the web console. The connection status displays next to the name of the catalog service domain.

Previous topic: Starting and logging on to the web console
Next topic: Viewing statistics with the web console
Related reference:
Web console statistics
stopOgServer script
Related information:
Getting started tutorial lesson 4: Monitor your environment

Viewing statistics with the web console

You can monitor statistics and other performance information with the web console.

Before you begin

Before you can view statistics with the web console, you must complete the following tasks:

1. Start the web console server. See Starting and logging on to the web console for more information.
2. Connect your catalog servers to the web console server. See Connecting the web console to catalog servers for more information.
3. Run active data grids and applications within the servers that are managed by your catalog service domain.

About this task

After you create your data grids and configure your applications to use the data grids, allow some time to pass for the statistics to become available. For example, with a dynamic cache data grid, statistics are not available until a WebSphere® Application Server that is running a dynamic cache connects to the dynamic cache. In general, wait up to one minute after a major configuration change to see the changes in your statistics.

Tip: To view more specific information about any data point in a chart, you can move the mouse pointer over the data point.

Procedure

- To view the current server statistics, click Monitor > Server Overview.
- To view the performance of all of your data grids, click Monitor > Data grid domain overview.
- To view individual data grids, click Monitor > Data grid overview > *data_grid_name*. This page shows a summary that includes the number of cache entries, the average transaction time, and average throughput.
- To view further details about a specific data grid, click Monitor > Data grid details. A tree displays with all of the data grids in your configuration. You can drill down into a specific data grid to view the maps that are a part of that data grid. You can either click a data grid name or a map for further information.
- To choose which statistics you would like your custom report to contain, click Monitor > Custom reports.
  Use this view to construct detailed data charts of the various statistics. Use the tree to explore the available data grids and servers and their associated statistics. A menu opens when you click or press enter on a node that references data that can be charted. Create a new chart containing the statistics, or add the statistics into an existing chart with compatible statistics. See Monitoring with custom reports for more information.

- Web console statistics
  Depending on the view you are using in the web console, you can view different statistics about your configuration. These statistics include the used memory, the top used data grids, and the number of cache entries.

Previous topic: Connecting the web console to catalog servers
Next topic: Monitoring with custom reports
Related reference:
Web console statistics
stopOgServer script
Related information:
Getting started tutorial lesson 4: Monitor your environment

Web console statistics

Depending on the view you are using in the web console, you can view different statistics about your configuration. These statistics include the used memory, the top used data grids, and the number of cache entries.

- Data grid domain overview
- Data grid overview
- Data grid details
- Server overview
- Custom reports: Catalog service domain statistics
  - Custom reports: Container server statistics
  - Custom reports: Data grid statistics
  - Custom reports: Map statistics

Data grid domain overview

Data grid domain overview statistics are displayed on the Monitor > Data Grid Domain Overview page. Click one of the following tabs for more information about the data grid domain:

Used Capacity tab
: In the Current Data Grid Used Capacity Distribution chart, a picture of the Total Pool, and the Largest Used Capacity Consumers are displayed. Only the top 25 data grids are displayed. In the Used Capacity Over Time chart, the number of bytes that are consumed by the data grid is displayed.

Average Throughput tab
: The 5 Most Active Data Grids by Average Transaction Time in Milliseconds chart contains a list of the top five data caches, organized by the average transaction time. The Average Throughput Over time chart displays the average, maximum, and minimum throughput within the last hour, day, and week.

Average Transaction Time tab
: The 5 Slowest Data Grids chart displays data about the slowest data grids. The Average Transaction Time Over Time chart displays the average , maximum, and minimum transaction time within the last hour, day, and week.

Data grid overview

To view statistics for an individual data grid, click Monitor > Data Grid Overview > *data_grid_name*.

Current summary over last 30 seconds
: Displays the current number of cache entries, average transaction time, average throughput, and cache hit rate for the selected data grid.

Used Capacity tab
: The Current summary over last 30 seconds chart displays the number of cache entries and used capacity in bytes over a specified time range.

Cache Usage tab
: The Cache Usage chart helps to visualize the number of successful queries to the cache, and displays cache attempts, cache hits, and the cache hit rate over a specified time range.

Average Throughput tab
: The Average Throughput vs. Average Transaction Time chart displays the transaction time and throughput over a specified time range.

Data grid details

Data grid statistics are displayed on the Monitor > Data Grid Details page. You can look at data for a selected grid and the maps that are within that grid.

Current summary over last 30 seconds
: Displays the current used capacity, number of cache entries, average throughput, and average transaction time for the selected data grid.

Current eXtreme Scale Object Grid Map Used Capacity Distribution
: View a total pool, which includes the capacity by zone and the total capacity in each zone. Only the top 25 ObjectGrid maps are displayed. You can also view the largest used capacity consumers by each map.

Current Zone Used Capacity Distribution
: View a total pool, which includes the total pool and the top used capacity consumers in the zone of the selected data grid. You can also view the largest used capacity consumers by each zone.

Map statistics:

Current summary over last 30 seconds
: Displays the current used capacity, number of cache entries, average throughput, and average transaction time for the selected map.

Current Partition Used Capacity Distribution
: View a partition, which includes the total pool and the top used capacity consumers. Only the top 25 partitions are displayed. You can also view the largest used capacity consumers by each partition.

Server overview

Server statistics are displayed on the Monitor > Server Overview page.

Current Server Used Memory Distribution
: This chart is composed of two views. Total Pool displays the current amount of used (real) memory in the server run time. Largest Used memory Consumers breaks down the used memory by server; however only the top 25 servers that are using the most memory are displayed.

Total Memory Over Time
: Displays the real memory usage in the server run time.

Used Memory Over Time
: Displays the amount of used memory in the server run time.

Custom reports: Catalog service domain statistics

You can view catalog service domain statistics by creating a custom report. Click Monitor > Custom Reports.

Average Transaction Time (ms)
: Displays the average time required to complete a transaction in this domain.

Average Transaction Throughput (trans/sec)
: Displays the average number of transactions per second in this domain.

Maximum Transaction Time (ms)
: Displays the time spent by the *most* time-consuming transaction in this domain.

Minimum Transaction Time (ms)
: Displays the time spent by the *least* time-consuming transaction in this domain.

Total Transaction Time (ms)
: Displays total time spent on transactions in this domain, since the time the domain was initialized.

Custom reports: Container server statistics

You can view container server statistics by creating a custom report. Click Monitor > Custom Reports.

Average Transaction Time (ms)
: Displays the average time required to complete a transaction for this catalog server.

Average Transaction Throughput (trans/sec)
: Displays the average number of transactions per second for this catalog server.

Maximum Transaction Time (ms)
: Displays the time spent by the *most* time-consuming transaction for this catalog server.

Minimum Transaction Time (ms)
: Displays the time spent by the *least* time-consuming transaction for this catalog server.

Total Transaction Time (ms)
: Displays total time spent on transactions for this catalog server, since the time for this catalog server was initialized.

Total Entries in Cache
: Displays the current number of objects cached in the grids overseen by this catalog server.

Hit rate (percentage)
: Displays the hit rate (hit ratio) for the selected data grid. A high hit rate is desirable. The hit rate indicates how well the grid is helping to avoid accessing the persistent store.

Used Bytes
: Displays memory consumption by this map. The used bytes statistics are accurate only when you are using simple objects or the COPY_TO_BYTES copy mode.

Minimum Used Bytes
: Displays the low point in memory consumption by this catalog service and its maps. The used bytes statistics are accurate only when you are using simple objects or the COPY_TO_BYTES copy mode.

Maximum Used Bytes
: Displays the high point in memory consumption by this catalog service and its maps. The used bytes statistics are accurate only when you are using simple objects or the COPY_TO_BYTES copy mode.

Total Number of Hits
: Displays the total number of times the requested data was found in the map, avoiding the need to access persistent store.

Total Number of Gets
: Displays the total number of times the map had to access the persistent store to obtain data.

Free Heap (MB)
: Displays the actual amount of heap available to the JVM being used by the catalog server.

Total Heap
: Displays the amount of heap available to the JVM being used by this catalog server.

Number of Available Processors
: Displays the number of processors that are available to this catalog service and its maps. For the highest stability, run your servers at 60% processor loading and JVM heaps at 60% heap loading. Spikes can then drive the processor usage to 80–90%, but do not regularly run your servers higher than these levels

Maximum Heap Size (MB)
: Displays the maximum amount of heap available to the JVM being used by this catalog server.

Used Memory
: Displays the used memory in the JVM being used by this catalog server.

Custom reports: Data grid statistics

You can view data grid statistics by creating a custom report. Click Monitor > Custom Reports.

Average Transaction Time (ms)
: Displays the average time required to complete transactions involving this grid.

Average Transaction Throughput (trans/sec)
: Displays the average number of transactions per second completed by this grid.

Maximum Transaction Time (ms)
: Displays the time spent by the *most* time-consuming transaction completed by this grid.

Minimum Transaction time (ms)
: Displays the time spent by the *least* time-consuming transaction completed by this grid.

Total Transaction Time (ms)
: Displays the total amount of transaction processing time for this grid.

Custom reports: Map statistics

You can view map statistics by creating a custom report. Click Monitor > Custom Reports.

Total Entries in Cache
: Displays the current number of objects cached in this map.

Hit Rate (percentage)
: Displays the hit rate (hit ratio) for the selected map. A high hit rate is desirable. The hit rate indicates how well the map is helping to avoid accessing the persistent store.

Used Bytes
: Displays memory consumption by this map. The used bytes statistics are accurate only when you are using simple objects or the COPY_TO_BYTES copy mode.

Minimum Used Bytes
: Displays the minimum consumption (in Bytes) for this map. The used bytes statistics are accurate only when you are using simple objects or the COPY_TO_BYTES copy mode.

Maximum Used Bytes
: Displays the maximum consumption (in Bytes) for this map. The used bytes statistics are accurate only when you are using simple objects or the COPY_TO_BYTES copy mode.

Total Number of Hits
: Displays the total number of times the requested data was found in the map, avoiding the need to access persistent store.

Total Number of Gets
: Displays the total number of times the map had to access the persistent store to obtain data.

Free Heap (MB)
: Displays the current amount of heap available to this map, in the JVM being used by the catalog server.

Total Heap (MB)
: Displays the total amount of heap available to this map, in the JVM being used by the catalog server. For the highest stability, run your servers at 60% processor loading and JVM heaps at 60% heap loading. Spikes can then drive the processor usage to 80–90%, but do not regularly run your servers higher than these levels

Number of Available Processors
: Displays the number of processors available to this map. For the highest stability, run your servers at 60% processor loading and JVM heaps at 60% heap loading. Spikes can then drive the processor usage to 80–90%, but do not regularly run your servers higher than these levels

Maximum Heap Size (MB)
: Displays the maximum amount of heap available to this map, in the JVM being used by the catalog server.

Used Memory (MB)
: Displays the used amount of memory in this map.

Related tasks:
Viewing statistics with the web console
Monitoring with the web console
Starting and logging on to the web console
Connecting the web console to catalog servers
Monitoring with the xscmd utility
Administering with the xscmd utility
Related information:
Getting started tutorial lesson 4: Monitor your environment

Monitoring with custom reports

You can build custom reports to save various charts that contain statistics about the catalog service domains, data grids, and container servers in your environment. You can save the custom reports and load them to view again later.

Before you begin

Before you can view statistics with the web console, you must complete the following tasks:

1. Start the web console server. See Starting and logging on to the web console for more information.
2. Connect your catalog servers to the web console server. See Connecting the web console to catalog servers for more information.
3. Run active data grids and applications within the servers that are managed by your catalog service domain.

Procedure

- Create a custom report.
    1. Click Monitor > Custom Reports. A list of the eXtreme Scale domains that you have defined are listed in a tree format. You can expand each of these domains to display the available statistics that you can add to the custom report.
    2. Add charts with the statistics you want to track. Available statistics are indicated by the chart icon (). Click one of the statistics that you want to track. Choose Add to new chart or Add to existing chart. Depending on your selection, the selected statistic either displays in a new chart tab or in the selected existing chart. You can only add a metric to an existing chart if the metrics already on the chart and the new metric use the same units.
- Save a custom report. Saving the custom report saves the statistics in all of the tabs you have created. To save the report, click Save.
- Load a custom report. Click Load and choose the saved custom report that you want to view.

Previous topic: Viewing statistics with the web console

Monitoring with CSV files

You can enable monitoring data collected for a container server to be written to comma-separated values (CSV) files. These CSV files can contain information about the Java virtual machine (JVM), map, or ObjectGrid instance.

About this task

By enabling monitoring data to be written to CSV files, you can download and analyze historical data for individual container servers. Data begins being collected when you start the server with the server properties that enable the CSV files. You can then download the CSV files at any time and use the files as you choose.

Procedure

1. Update the server properties file with the following properties that are related to enabling the CSV files.

```
parameter=default value
jvmStatsLoggingEnabled=true
maxJVMStatsFiles=5
maxJVMStatsFileSize=100
jvmStatsFileName=jvmstats
jvmStatsWriteRate=10

mapStatsLoggingEnabled=true
maxMapStatsFiles=5
maxMapStatsFileSize=100
mapStatsFileName=mapstats
mapStatsWriteRate=10

ogStatsLoggingEnabled=true
maxOGStatsFiles=5
maxOGStatsFileSize=100
ogStatsFileName=ogstats
ogStatsWriteRate=10
```

   For more information about these properties, see Server properties file.
2. Restart the server to pick up the changes to the server properties file.
3. Import the CSV file into the program that you are using to process the data, such as a spreadsheet.

What to do next

For more information about the data that is contained in the CSV files, see CSV file statistics definitions.

- CSV file statistics definitions
  The CSV files that you can download for a server include statistics that you can use to build historical charts or other information.

Related concepts:
Statistics overview
Monitoring with vendor tools
Related tasks:
Monitoring with the web console
Enabling statistics
Monitoring with the xscmd utility
Monitoring with WebSphere Application Server PMI
Monitoring server statistics with managed beans (MBeans)
Monitoring eXtreme Scale information in DB2
Related reference:
Server properties file
startOgServer script

CSV file statistics definitions

The CSV files that you can download for a server include statistics that you can use to build historical charts or other information.

Java virtual machine (JVM) statistics log

TimeStamp (column 1)
: Specifies the date and time of the statistics snapshot that was taken for the Java virtual machine (JVM).

ServerName (column 2)
: Specifies the server name of the JVM.

Hostname (column 3)
: Specifies the host name of the JVM.

FreeMemory (column 4)
: Specifies the number of available bytes for the JVM.

MaxMemory (column 5)
: Specifies the maximum number of bytes that can be allocated for the JVM.

TotalMemory (column 6)
: Displays the real memory usage in the server run time.

AvailProcs (column 7)
: Displays the number of processors that are available to this catalog service and its maps. For the highest stability, run your servers at 60% processor loading and JVM heaps at 60% heap loading. Spikes can then drive the processor usage to 80–90%, but do not regularly run your servers higher than these levels

Map statistics log

TimeStamp (column 1)
: Specifies the date and time of the statistics snapshot that was taken for the map.

MapName (column 2)
: Specifies the name of the map.

OgName (column 3)
: Specifies the name of the data grid to which this map belongs.

PartitionId (column 4)
: Specifies the ID of the partition.

MapSetName (column 5)
: Specifies the map set to which this map belongs.

HitRate (column 6)
: Displays the hit rate (hit ratio) for the selected map. A high hit rate is desirable. The hit rate indicates how well the data grid is helping to avoid accessing the persistent store.

Count (column 7)
: Indicates the number of entries in the data grid since the server started. For example, a value of 100 indicates that the entry is the 100th sample entry that has been gathered since the server started.

TotalGetCount (column 8)
: Displays the total number of times the map had to access the persistent store to obtain data.

TotalHitCount (column 9)
: Displays the total number of times the requested data was found in the map, avoiding the need to access persistent store.

StartTime (column 10)
: Specifies the time that the counters began from last reset call. The resets occur when the server starts or restarts.

LastCount (column 11)
: Specifies the amount of time since the last data sample was taken.

LastTotalGetCount (column 12)
: Indicates the current total number of get operations from the cache minus the number of get operations in the previous time period.

LastTotalHitCount (column 13)
: Indicates the current total number of hits from the cache minus the number of hits in the previous time period.

UsedBytes (column 14)
: Displays memory consumption by this map. The used bytes statistics are accurate only when you are using simple objects or the COPY_TO_BYTES copy mode.

MinUsedBytes (column 15)
: Displays the low point in memory consumption by this catalog service and its maps. The used bytes statistics are accurate only when you are using simple objects or the COPY_TO_BYTES copy mode.

MaxUsedBytes (column 16)
: Displays the high point in memory consumption by this catalog service and its maps. The used bytes statistics are accurate only when you are using simple objects or the COPY_TO_BYTES copy mode.

LastUsedBytes (column 17)
: Indicates the current UsedBytes value minus the UsedBytes value from the previous statistics collection period.

SampleLen (column 18)
: Indicates the length, in milliseconds, of the time period during with the data was sampled.

ObjectGrid statistics log

TimeStamp (column 1)
: Specifies the date and time of the statistics snapshot that was taken for the data grid.

OgName (column 2)
: Specifies the name of the data grid.

PartitionId (column 3)
: Specifies the partition ID.

Count (column 4)
: Indicates a count of the entries in the data grid that have been gathered since the server started. For example, a value of 100 indicates that the entry is the 100th sample entry that has been gathered since the server started.

Hostname (column 5)
: Specifies the host name.

DomainName (column 6)
: Specifies the catalog service domain to which this data grid belongs.

MaxTime (column 7)
: Displays the time spent by the *most* time-consuming transaction for this server.

MinTime (column 8)
: Displays the time spent by the *least* time-consuming transaction for this server.

MeanTime (column 9)
: Specifies the average time spent on a transaction.

TotalTime (column 10)
: Displays total time spent on transactions for this server, since the time for this server was initialized.

AvgTransTime (column 11)
: Displays the average time required to complete a transaction for this server.

AvgThroughPut (column 12)
Displays the average number of transactions per second for this server.
SumOfSquares (column 13)
Specifies the sum of squares value for the transaction time. This value measures the deviation from the mean at the given point in time.
SampleLen (column 14)
Indicates the length, in milliseconds, of the time period during with the data was sampled.
LastDataSample (column 15)
Specifies the time since the last sample was taken.
LastTotalTime (column 16)
Specifies the current total time minus the previous total time for the data sample.
StartTime (column 17)
Indicates the time that the statistics began to be collected since the last reset of the data. The data is reset when the server restarts.

Enabling statistics

WebSphere® eXtreme Scale uses an internal statistics model to track and filter data, which is the underlying structure that all data views use to gather snapshots of statistics. You can use several methods to retrieve the information from the statistics modules.

About this task

For a list of all of the modules on which you can enable statistics, see StatsSpec class.

Procedure

- Enable statistics with the server properties file. You can use the statsSpec property in the server properties file for the container server to set the statistics specification when you start the server. For more information, see Server properties file.
- Enable statistics with the **xscmd** utility. You can use the **-c setStatsSpec** command to set the statistics specification at run time. For more information, see Administering with the xscmd utility.
- Enable statistics programmatically with the StatsSpec interface. For more information, see Monitoring with the statistics API.
- Enable statistics with JMX with the setStatsSpec operation on the DynamicServerMBean. For more information, see Interface DynamicServerMBean.

Example

Some examples of statsSpec strings that you might specify using the properties file, **xscmd** utility, or StatsSpec interface follow:

Enable all statistics for all modules:

```
all=enabled
```

Disable all statistics for all modules:

```
all=disabled
```

Enable statistics for all statistics in the OGStatsModule:

```
og.all=enabled
```

Enable statistics for all statistics in the OGStatsModule and MapStatsModule:

```
og.all=enabled;map.all=enabled
```

Enable statistics for only Map Used bytes statistic, and disable everything else:

```
all=disabled;map.usedbytes=enabled
```

- Statistics modules
  WebSphere eXtreme Scale uses an internal statistics model to track and filter data, which is the underlying structure that all data views use to gather snapshots of statistics.
- Monitoring with the statistics API
  The Statistics API is the direct interface to the internal statistics tree. Statistics are disabled by default, but can be enabled by setting a StatsSpec interface. A StatsSpec interface defines how WebSphere eXtreme Scale should monitor statistics.

Related concepts:
Statistics overview
Monitoring with vendor tools
Statistics modules
Related tasks:
Monitoring with the web console
Monitoring with CSV files
Monitoring with the xscmd utility
Monitoring with WebSphere Application Server PMI
Monitoring server statistics with managed beans (MBeans)
Monitoring eXtreme Scale information in DB2
Related reference:
Server properties file

startOgServer script
Related information:
StatsSpec class

Statistics modules

WebSphere® eXtreme Scale uses an internal statistics model to track and filter data, which is the underlying structure that all data views use to gather snapshots of statistics.

Overview

Statistics in WebSphere eXtreme Scale are tracked and contained within StatsModules components. Within the statistics model, several types of statistics modules exist:

OGStatsModule
: Provides statistics for an ObjectGrid instance, including transaction response times.

MapStatsModule
: Provides statistics for a single map, including the number of entries and hit rate.

QueryStatsModule
: Provides statistics for queries, including plan creation and run times.

AgentStatsModule
: Provides statistics for DataGrid API agents, including serialization times and run times.

HashIndexStatsModule
: Provides statistics for HashIndex query and maintenance run times.

SessionStatsModule
: Provides statistics for the HTTP session manager plug-in.

For details about the statistics modules, see the Statistics API.

Statistics in a local environment

The model is organized like an n-ary tree (a tree structure with the same degree for all nodes) comprised of all of the StatsModule types mentioned in the previous list. Because of this organization structure, every node in the tree is represented by the StatsFact interface. The StatsFact interface can represent an individual module or a group of modules for aggregation purposes. For example, if several leaf nodes in the tree represent particular MapStatsModule objects, the parent StatsFact node to these nodes contains aggregated statistics for all of the children modules. After you fetch a StatsFact object, you can then use interface to retrieve the corresponding StatsModule.

Much like a tree map, you use a corresponding path or key to retrieve a specific StatsFact. The path is a String[] value that consists of every node that is along the path to the requested fact. For example, you created an ObjectGrid called ObjectGridA, which contains two Maps: MapA and MapB. The path to the StatsModule for MapA would look like `[ObjectGridA, MapA]`. The path to the aggregated statistics for both maps would be: `[ObjectGridA]`.

Statistics in a distributed environment

In a distributed environment, the statistics modules are retrieved using a different path. Because a server can contain multiple partitions, the statistics tree needs to track the partition to which each module belongs. As a result, the path to look up a particular StatsFact object is different. Using the previous example, but adding in that the maps exist within partition 1, the path is `[1, ObjectGridA, MapA]` for retrieving that StatsFact object for MapA.
Related tasks:
Enabling statistics
Monitoring with the statistics API
Administering with the xscmd utility
Related reference:
Server properties file
startOgServer script
Related information:
StatsSpec class

Monitoring with the statistics API

The Statistics API is the direct interface to the internal statistics tree. Statistics are disabled by default, but can be enabled by setting a StatsSpec interface. A StatsSpec interface defines how WebSphere® eXtreme Scale should monitor statistics.

About this task

You can use the local StatsAccessor API to query data and access statistics on any ObjectGrid instance that is in the same Java™ virtual machine (JVM) as the running code. For more information about the specific interfaces, see the API documentation. Use the following steps to enable monitoring of the internal statistics tree.

Procedure

1. Retrieve the StatsAccessor object. The StatsAccessor interface follows the singleton pattern. So, apart from problems related to the classloader, one StatsAccessor instance should exist for each JVM. This class serves as the main interface for all local statistics operations. The following code is an example of how to retrieve the accessor class. Call this operation before any other ObjectGrid calls.

```
public class LocalClient
{

   public static void main(String[] args) {

      // retrieve a handle to the StatsAccessor
      StatsAccessor accessor = StatsAccessorFactory.getStatsAccessor();

   }

}
```

2. Set the data grid StatsSpec interface. Set this JVM to collect all statistics at the ObjectGrid level only. You must ensure that an application enables all statistics that might be needed before you begin any transactions. The following example sets the StatsSpec interface using both a static constant field and using a spec String. Using a static constant field is simpler because the field has already defined the specification. However, by using a spec String, you can enable any combination of statistics that are required.

```
public static void main(String[] args) {

      // retrieve a handle to the StatsAccessor
      StatsAccessor accessor = StatsAccessorFactory.getStatsAccessor();

      // Set the spec via the static field
      StatsSpec spec = new StatsSpec(StatsSpec.OG_ALL);
      accessor.setStatsSpec(spec);

      // Set the spec via the spec String
      StatsSpec spec = new StatsSpec("og.all=enabled");
      accessor.setStatsSpec(spec);

}
```

3. Send transactions to the grid to force data to be collected for monitoring. To collect useful data for statistics, you must send transactions to the data grid. The following code excerpt inserts a record into MapA, which is in ObjectGridA. Because the statistics are at the ObjectGrid level, any map within the ObjectGrid yields the same results.

```
public static void main(String[] args) {

      // retrieve a handle to the StatsAccessor
      StatsAccessor accessor = StatsAccessorFactory.getStatsAccessor();

      // Set the spec via the static field
      StatsSpec spec = new StatsSpec(StatsSpec.OG_ALL);
      accessor.setStatsSpec(spec);

      ObjectGridManager manager =
                                ObjectGridmanagerFactory.getObjectGridManager();
      ObjectGrid grid = manager.getObjectGrid("ObjectGridA");
      Session session = grid.getSession();
      Map map = session.getMap("MapA");

      // Drive insert
      session.begin();
      map.insert("SomeKey", "SomeValue");
      session.commit();
}
```

4. Query a StatsFact by using the StatsAccessor API. Every statistics path is associated with a StatsFact interface. The StatsFact interface is a generic placeholder that is used to organize and contain a StatsModule object. Before you can access the actual statistics module, the StatsFact object must be retrieved.

```
public static void main(String[] args)
{

      // retrieve a handle to the StatsAccessor
      StatsAccessor accessor = StatsAccessorFactory.getStatsAccessor();

      // Set the spec via the static field
      StatsSpec spec = new StatsSpec(StatsSpec.OG_ALL);
      accessor.setStatsSpec(spec);

      ObjectGridManager manager =
                                ObjectGridManagerFactory.getObjectGridManager();
      ObjectGrid grid = manager.getObjectGrid("ObjectGridA");
      Session session = grid.getSession();
      Map map = session.getMap("MapA");

      // Drive insert
      session.begin();
      map.insert("SomeKey", "SomeValue");
      session.commit();
```

```
        // Retrieve StatsFact

        StatsFact fact = accessor.getStatsFact(new String[] {"EmployeeGrid"},
                                    StatsModule.MODULE_TYPE_OBJECT_GRID);

    }
```

5. Interact with the StatsModule object. The StatsModule object is contained within the StatsFact interface. You can obtain a reference to the module by using the StatsFact interface. Since the StatsFact interface is a generic interface, you must cast the returned module to the expected StatsModule type. Because this task collects eXtreme Scale statistics, the returned StatsModule object is cast to an OGStatsModule type. After the module is cast, you have access to all of the available statistics.

```
public static void main(String[] args) {

        // retrieve a handle to the StatsAccessor
        StatsAccessor accessor = StatsAccessorFactory.getStatsAccessor();

        // Set the spec via the static field
        StatsSpec spec = new StatsSpec(StatsSpec.OG_ALL);
        accessor.setStatsSpec(spec);

        ObjectGridManager manager =
                                    ObjectGridmanagerFactory.getObjectGridManager();
        ObjectGrid grid = manager.getObjectGrid("ObjectGridA");
        Session session = grid.getSession();
        Map map = session.getMap("MapA");

        // Drive insert
        session.begin();
        map.insert("SomeKey", "SomeValue");
        session.commit();

        // Retrieve StatsFact
        StatsFact fact = accessor.getStatsFact(new String[] {"EmployeeGrid"},
                                    StatsModule.MODULE_TYPE_OBJECT_GRID);

        // Retrieve module and time
        OGStatsModule module = (OGStatsModule)fact.getStatsModule();
        ActiveTimeStatistic timeStat =
                                    module.getTransactionTime("Default", true);
        double time = timeStat.getMeanTime();

    }
```

Related concepts:
Statistics modules
Related reference:
Server properties file
startOgServer script
Related information:
StatsSpec class

Monitoring with the xscmd utility

The **xscmd** utility replaces the **xsadmin** sample utility as a fully supported monitoring and administration tool. With the **xscmd** utility, you can display textual information about your WebSphere® eXtreme Scale topology.

Before you begin

- For the **xscmd** 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.
- See Administering with the xscmd utility for more information about starting the **xscmd** utility.

About this task

You can use the **xscmd** 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 out filtered metrics regarding the map size of the *MapA* map. To see all possible command options, run the **xscmd** utility without any arguments or with the -help option.

Procedure

Monitor the environment with the **xscmd** utility.

- To enable statistics for all of the servers, run the following command:
  - UNIX `./xscmd.sh -c setStatsSpec -spec ALL=enabled`
  - Windows `xscmd.bat -c setStatsSpec -spec ALL=enabled`
- To display all online container servers for a data grid, run the following command:

- UNIX `./xscmd.sh -c showPlacement -g ObjectGridA -ms MapSetA`
- Windows `xscmd.bat -c showPlacement -g ObjectGridA -ms MapSetA`

All container information is displayed.

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 **startOgServer** script or you can call the setJMXServicePort method on the ServerProperties interface.

- To display information about the maps for the ObjectGridA data grid, run the following command:
  - UNIX `./xscmd.sh -c showMapSizes -g ObjectGridA -ms MapSetA`
  - Windows `xscmd.bat -c showMapSizes -g ObjectGridA -ms MapSetA`
- To connect to the catalog service and display information about the MapA map for the entire catalog service domain, run the following command:
  - UNIX `./xscmd.sh -c showMapSizes -g ObjectGridA -ms MapSetA -m MapA -cep CatalogMachine:6645`
  - Windows `xscmd.bat -c showMapSizes -g ObjectGridA -ms MapSetA -m MapA -cep CatalogMachine:6645`

  The **xscmd** utility connects to the MBean server that is running on a catalog server. By connecting to a single catalog server, you can retrieve information about the entire catalog service domain. A catalog server can run as a stand-alone process, WebSphere Application Server process, or embedded within a custom application process. Use the -cep option to specify the catalog service host name and port. If you include a list of catalog servers for the -cep option, the catalog servers must be within the same catalog service domain. You can retrieve statistics for one catalog service domain at a time.

- To display the configured and runtime placement of your configuration, run one of the following commands:
  - `xscmd -c placementServiceStatus`
  - `xscmd -c placementServiceStatus -g ObjectGridA -ms MapSetA`
  - `xscmd -c placementServiceStatus -ms MapSetA`
  - `xscmd -c placementServiceStatus -g ObjectGridA`

  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.

Related concepts:
Statistics overview
Monitoring with vendor tools
Related tasks:
Monitoring with the web console
Monitoring with CSV files
Enabling statistics
Monitoring with WebSphere Application Server PMI
Monitoring server statistics with managed beans (MBeans)
Monitoring eXtreme Scale information in DB2
Accessing Managed Beans (MBeans) using the wsadmin tool
Accessing Managed Beans (MBeans) programmatically
Monitoring server statistics with managed beans (MBeans)
Configuring security profiles for the xscmd utility
Administering with the xscmd utility
Related reference:
Web console statistics
stopOgServer script
Administering with Managed Beans (MBeans)
xsadmin tool to xscmd tool migration
Related information:
Module 5: Use the xscmd utility to monitor data grids and maps
Module 5: Use the xscmd tool to monitor data grids and maps
Getting started tutorial lesson 4: Monitor your environment
API documentation: Package com.ibm.websphere.objectgrid.management
Interface PlacementServiceMBean

Monitoring with WebSphere Application Server PMI

WebSphere® eXtreme Scale supports Performance Monitoring Infrastructure (PMI) when running in a WebSphere Application Server or WebSphere Extended Deployment application server. PMI collects performance data on runtime applications and provides interfaces that support external applications to monitor performance data. You can use the administrative console or the wsadmin tool to access monitoring data.

Before you begin

You can use PMI to monitor your environment when you are using WebSphere eXtreme Scale combined with WebSphere Application Server.

About this task

WebSphere eXtreme Scale uses the custom PMI feature of WebSphere Application Server to add its own PMI instrumentation. With this approach, you can enable and disable WebSphere eXtreme Scale PMI with the administrative console or with Java™ Management Extensions (JMX) interfaces in the wsadmin tool. In addition, you can access WebSphere eXtreme Scale statistics with the standard PMI and JMX interfaces that are used by monitoring tools, including the Tivoli® Performance Viewer.

Procedure

1. Enable eXtreme Scale PMI. You must enable PMI to view the PMI statistics. See Enabling PMI for more information.
2. Retrieve eXtreme Scale PMI statistics. View the performance of your eXtreme Scale applications with the Tivoli Performance Viewer. See Retrieving PMI statistics for more information.

What to do next

For more information about the wsadmin tool, see Accessing Managed Beans (MBeans) using the wsadmin tool.

- Enabling PMI
  You can use WebSphere Application Server Performance Monitoring Infrastructure (PMI) to enable or disable statistics at any level. For example, you can choose to enable the map hit rate statistic for a particular map, but not the number of entry statistic or the loader batch update time statistic. You can enable PMI in the administrative console or with scripting.
- Retrieving PMI statistics
  By retrieving PMI statistics, you can see the performance of your eXtreme Scale applications.
- PMI modules
  You can monitor the performance of your applications with the performance monitoring infrastructure (PMI) modules.
- Accessing Managed Beans (MBeans) using the wsadmin tool
  You can use the wsadmin utility provided in WebSphere Application Server to access managed bean (MBean) information.

Related concepts:
Statistics overview
Monitoring with vendor tools
Related tasks:
Monitoring with the web console
Monitoring with CSV files
Enabling statistics
Monitoring with the xscmd utility
Monitoring server statistics with managed beans (MBeans)
Monitoring eXtreme Scale information in DB2

Enabling PMI

You can use WebSphere® Application Server Performance Monitoring Infrastructure (PMI) to enable or disable statistics at any level. For example, you can choose to enable the map hit rate statistic for a particular map, but not the number of entry statistic or the loader batch update time statistic. You can enable PMI in the administrative console or with scripting.

Before you begin

- Your application server must be started and have an eXtreme Scale-enabled application installed.
- To enable PMI with wsadmin scripting, you also must be able to log in and use the wsadmin tool. For more information about the wsadmin tool, see WebSphere Application Server Information Center: Scripting the application serving environment (wsadmin).

About this task

Use WebSphere Application Server PMI to provide a granular mechanism with which you can enable or disable statistics at any level. For example, you can choose to enable the map hit rate statistic for a particular map, but not the number of entry or the loader batch update time statistics. This section shows how to use the administrative console and wsadmin scripts to enable ObjectGrid PMI.

Procedure

- **Enable PMI in the administrative console.**
  1. In the administrative console, click Monitoring and Tuning > Performance Monitoring Infrastructure > *server_name*.
  2. Verify that Enable Performance Monitoring Infrastructure (PMI) is selected. This setting is enabled by default. If the setting is not enabled, select the check box, then restart the server.
  3. Click Custom. In the configuration tree, select the ObjectGrid and ObjectGrid Maps module. Enable the statistics for each module.

  The transaction type category for ObjectGrid statistics is created at runtime. You can see only the subcategories of the ObjectGrid and Map statistics on the Runtime tab.
- **Enable PMI with scripting.**
  1. Open a command line prompt. Navigate to the *was_root*/bin directory. Type **wsadmin** to start the wsadmin command line tool.
  2. Modify the eXtreme Scale PMI runtime configuration. Verify that PMI is enabled for the server using the following commands:

     ```
     wsadmin>set s1 [$AdminConfig getid /Cell:CELL_NAME/Node:NODE_NAME/
                                    Server:APPLICATION_SERVER_NAME/]
     wsadmin>set pmi [$AdminConfig list PMIService $s1]
     wsadmin>$AdminConfig show $pmi.
     ```

     If PMI is not enabled, run the following commands to enable PMI:

     ```
     wsadmin>$AdminConfig modify $pmi {{enable true}}
     wsadmin>$AdminConfig save
     ```

     If you need to enable PMI, restart the server.
  3. Set variables for changing the statistic set to a custom set using the following commands:

```
wsadmin>set perfName [$AdminControl completeObjectName type=Perf,
process=APPLICATION_SERVER_NAME,*]
wsadmin>set perfOName [$AdminControl makeObjectName $perfName]
wsadmin>set params [java::new {java.lang.Object[]} 1]
wsadmin>$params set 0 [java::new java.lang.String custom]
wsadmin>set sigs [java::new {java.lang.String[]} 1]
wsadmin>$sigs set 0 java.lang.String
```

4. Set statistic set to custom using the following command:

```
wsadmin>$AdminControl invoke_jmx $perfOName setStatisticSet $params $sigs
```

5. Set variables to enable the objectGridModule PMI statistic using the following commands:

```
wsadmin>set params [java::new {java.lang.Object[]} 2]
wsadmin>$params set 0 [java::new java.lang.String objectGridModule=1]
wsadmin>$params set 1 [java::new java.lang.Boolean false]
wsadmin>set sigs [java::new {java.lang.String[]} 2]
wsadmin>$sigs set 0 java.lang.String
wsadmin>$sigs set 1 java.lang.Boolean
```

6. Set the statistics string using the following command:

```
wsadmin>set params2 [java::new {java.lang.Object[]} 2]
wsadmin>$params2 set 0 [java::new java.lang.String mapModule=*]
wsadmin>$params2 set 1 [java::new java.lang.Boolean false]
wsadmin>set sigs2 [java::new {java.lang.String[]} 2]
wsadmin>$sigs2 set 0 java.lang.String
wsadmin>$sigs2 set 1 java.lang.Boolean
```

7. Set the statistics string using the following command:

```
wsadmin>$AdminControl invoke_jmx $perfOName setCustomSetString $params2 $sigs2
```

These steps enable eXtreme Scale runtime PMI, but do not modify the PMI configuration. If you restart the application server, the PMI settings are lost except for the main PMI enablement.

Example

You can perform the following steps to enable PMI statistics for the sample application:

1. Launch the application using the `http://host:port/ObjectGridSample` Web address, where host and port are the host name and HTTP port number of the server where the sample is installed.
2. In the sample application, click ObjectGridCreationServlet, and then click action buttons 1, 2, 3, 4, and 5 to generate actions to the ObjectGrid and maps. Do not close this servlet page right now.
3. In the administrative console, click Monitoring and Tuning > Performance Monitoring Infrastructure > *server_name* Click the Runtime tab.
4. Click the Custom radio button.
5. Expand the ObjectGrid Maps module in the runtime tree, then click the clusterObjectGrid link. Under the ObjectGrid Maps group, there is an ObjectGrid instance called clusterObjectGrid, and under the clusterObjectGrid group four maps exist: counters, employees, offices, and sites. In the ObjectGrids instance, there is the clusterObjectGrid instance, and under that instance is a transaction type called DEFAULT.
6. You can enable the statistics of interest to you. For example, you can enable number of map entries for employees map, and transaction response time for the DEFAULT transaction type.

What to do next

After PMI is enabled, you can view PMI statistics with the administrative console or through scripting.

Retrieving PMI statistics

By retrieving PMI statistics, you can see the performance of your eXtreme Scale applications.

Before you begin

- Enable PMI statistics tracking for your environment. See Enabling PMI for more information.
- The paths in this task are assuming you are retrieving statistics for the sample application, but you can use these statistics for any other application with similar steps.
- If you are using the administrative console to retrieve statistics, you must be able to log in to the administrative console. If you are using scripting, you must be able to log in to wsadmin.

About this task

You can retrieve PMI statistics to view in Tivoli® Performance Viewer by completing steps in the administrative console or with scripting.For more information about the statistics that can be retrieved, see PMI modules.

Procedure

- Retrieve PMI statistics in the administrative console.
  1. In the administrative console, click Monitoring and tuning > Performance viewer > Current activity
  2. Select the server that you want to monitor using Tivoli Performance Viewer, then enable the monitoring.
  3. Click the server to view the Performance viewer page.
  4. Expand the configuration tree. Click ObjectGrid Maps > clusterObjectGrid select employees. Expand ObjectGrids > clusterObjectGrid and select DEFAULT.
  5. In the ObjectGrid sample application, go to the ObjectGridCreationServlet servlet , click button 1, then populate maps. You can view the statistics in the viewer.
- Retrieve PMI statistics with scripting.
  1. On a command line prompt, navigate to the *was_root*/bin directory. Type `wsadmin` to start the wsadmin tool.
  2. Set variables for the environment using the following commands:

```
wsadmin>set perfName [$AdminControl completeObjectName type=Perf,*]
wsadmin>set perfOName [$AdminControl makeObjectName $perfName]
wsadmin>set mySrvName [$AdminControl completeObjectName type=Server,
        name=APPLICATION_SERVER_NAME,*]
```

  3. Set variables to get mapModule statistics using the following commands:

```
wsadmin>set params [java::new {java.lang.Object[]} 3]
wsadmin>$params set 0 [$AdminControl makeObjectName $mySrvName]
wsadmin>$params set 1 [java::new java.lang.String mapModule]
wsadmin>$params set 2 [java::new java.lang.Boolean true]
wsadmin>set sigs [java::new {java.lang.String[]} 3]
wsadmin>$sigs set 0 javax.management.ObjectName
wsadmin>$sigs set 1 java.lang.String
wsadmin>$sigs set 2 java.lang.Boolean
```

  4. Get mapModule statistics using the following command:

```
wsadmin>$AdminControl invoke_jmx $perfOName getStatsString $params $sigs
```

  5. Set variables to get objectGridModule statistics using the following commands:

```
wsadmin>set params2 [java::new {java.lang.Object[]} 3]
wsadmin>$params2 set 0 [$AdminControl makeObjectName $mySrvName]
wsadmin>$params2 set 1 [java::new java.lang.String objectGridModule]
wsadmin>$params2 set 2 [java::new java.lang.Boolean true]
wsadmin>set sigs2 [java::new {java.lang.String[]} 3]
wsadmin>$sigs2 set 0 javax.management.ObjectName
wsadmin>$sigs2 set 1 java.lang.String
wsadmin>$sigs2 set 2 java.lang.Boolean
```

  6. Get objectGridModule statistics using the following command:

```
wsadmin>$AdminControl invoke_jmx $perfOName getStatsString $params2 $sigs2
```

Results

You can view statistics in the Tivoli Performance Viewer.

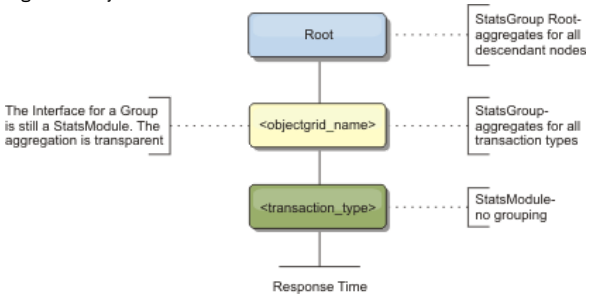
PMI modules

You can monitor the performance of your applications with the performance monitoring infrastructure (PMI) modules.

objectGridModule

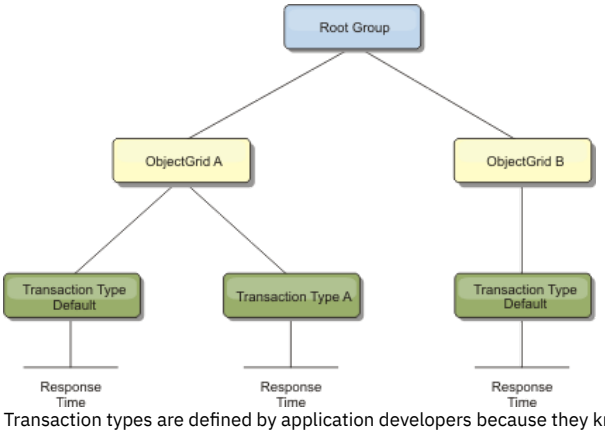
The objectGridModule contains a time statistic: transaction response time. A transaction is defined as the duration between the Session.begin method call and the Session.commit method call. This duration is tracked as the transaction response time. The root element of the objectGridModule, "root", serves as the entry point to the WebSphere® eXtreme Scale statistics. This root element has ObjectGrids as its child elements, which have transaction types as their child elements. The response time statistic is associated with each transaction type.

Figure 1. ObjectGridModule module structure



The following diagram shows an example of the ObjectGridModule structure. In this example, two ObjectGrid instances exist in the system: ObjectGrid A and ObjectGrid B. The ObjectGrid A instance has two types of transactions: A and default. The ObjectGrid B instance has only the default transaction type.

Figure 2. ObjectGridModule module structure example



Transaction types are defined by application developers because they know what types of transactions their applications use. The transaction type is set using the following Session.setTransactionType(String) method:

```
/**
* Sets the transaction type for future transactions.
*
* After this method is called, all of the future transactions have the
* same type until another transaction type is set. If  no transaction
* type is set, the default  TRANSACTION_TYPE_DEFAULT transaction type
* is used.
*
* Transaction types are used mainly for statistical data tracking purpose.
* Users can predefine types of transactions that run in an
* application. The idea is to categorize transactions with the same characteristics
* to one category (type), so one transaction response time statistic can be
* used to track each transaction type.
*
* This tracking is useful when your application has different types of
* transactions.
* Among them, some types of transactions, such as update transactions, process
* longer than other transactions, such as read-only transactions. By using the
* transaction type, different transactions are tracked by different statistics,
* so the statistics can be more useful.
*
* @param tranType the transaction type for future transactions.
*/
void setTransactionType(String tranType);
```

The following example sets transaction type to `updatePrice`:

```
// Set the transaction type to updatePrice
// The time between session.begin() and session.commit() will be
// tracked in the time statistic for "updatePrice".
session.setTransactionType("updatePrice");
session.begin();
map.update(stockId, new Integer(100));
session.commit();
```

The first line indicates that the subsequent transaction type is updatePrice. An updatePrice statistic exists under the ObjectGrid instance that corresponds to the session in the example. Using Java™ Management Extensions (JMX) interfaces, you can get the transaction response time for updatePrice transactions. You can also get the aggregated statistic for all types of transactions on the specified ObjectGrid instance.

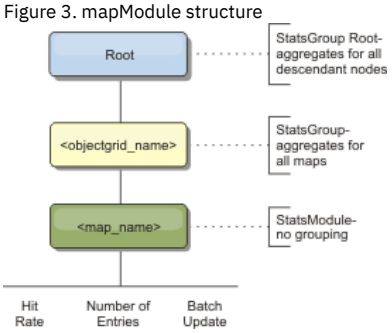
mapModule

The mapModule contains three statistics that are related to eXtreme Scale maps:

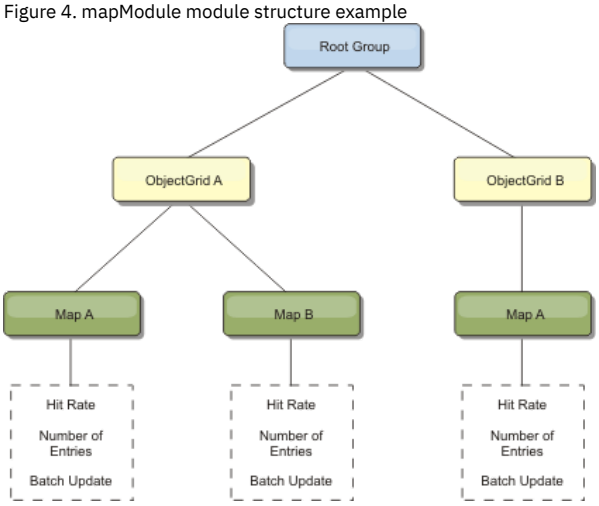
- **Map hit rate** - *BoundedRangeStatistic*: Tracks the hit rate of a map. Hit rate is a float value between 0 and 100 inclusively, which represents the percentage of map hits in relation to map get operations.
- **Number of entries**-*CountStatistic*: Tracks the number of entries in the map.
- **Loader batch update response time**-*TimeStatistic*: Tracks the response time that is used for the loader batch-update operation.

The root element of the mapModule, "root", serves as the entry point to the ObjectGrid Map statistics. This root element has ObjectGrids as its child elements, which have maps as their child elements. Every map instance has the three listed statistics. The mapModule structure is shown in the following diagram:

Figure 3. mapModule structure



The following diagram shows an example of the mapModule structure:
Figure 4. mapModule module structure example

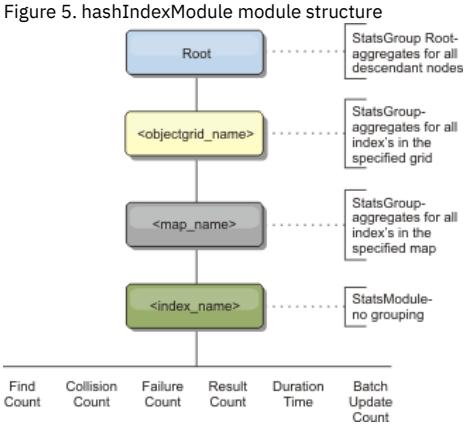


hashIndexModule

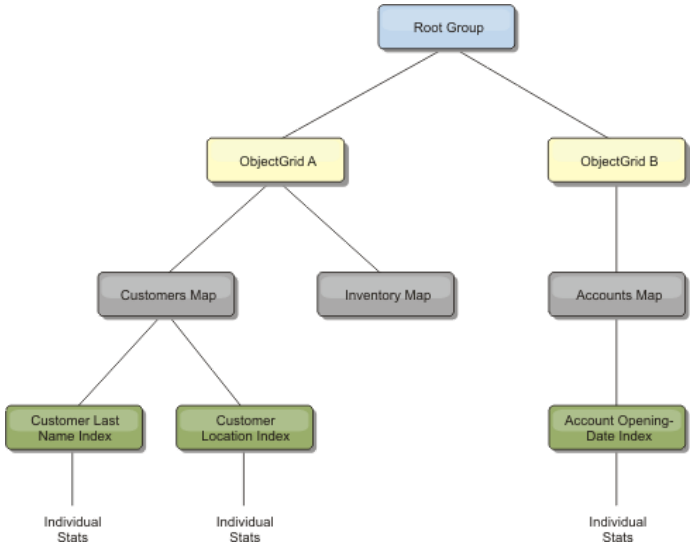
The hashIndexModule contains the following statistics that are related to Map-level indexes:

- **Find Count**-*CountStatistic*: The number of invocations for the index find operation.
- **Collision Count**-*CountStatistic*: The number of collisions for the find operation.
- **Failure Count**-*CountStatistic*: The number of failures for the find operation.
- **Result Count**-*CountStatistic*: The number of keys returned from the find operation.
- **BatchUpdate Count**-*CountStatistic*: The number of batch updates against this index. When the corresponding map is changed in any manner, the index will have its doBatchUpdate() method called. This statistic will tell you how frequently your index is changing or being updated.
- **Find Operation Duration Time**-*TimeStatistic*: The amount of time the find operation takes to complete

The root element of the hashIndexModule, "root", serves as the entry point to the HashIndex statistics. This root element has ObjectGrid instances as its child elements, ObjectGrid instances have maps as their child elements, which have HashIndexes as their child elements and leaf nodes in the tree. Every HashIndex instance has the three listed statistics. The hashIndexModule structure is shown in the following diagram:
Figure 5. hashIndexModule module structure



The following diagram shows an example of the hashIndexModule structure:
Figure 6. hashIndexModule module structure example



agentManagerModule

The agentManagerModule contains statistics that are related to map-level agents:

- **Reduce Time**: *TimeStatistic* - The amount of time for the agent to finish the reduce operation.
- **Total Duration Time**: *TimeStatistic* - The total amount of time for the agent to complete all operations.
- **Agent Serialization Time**: *TimeStatistic* - The amount of time to serialize the agent.
- **Agent Inflation Time**: *TimeStatistic* - The amount of time it takes to inflate the agent on the server.
- **Result Serialization Time**: *TimeStatistic* - The amount of time to serialize the results from the agent.
- **Result Inflation Time**: *TimeStatistic* - The amount of time to inflate the results from the agent.
- **Failure Count**: *CountStatistic* - The number of times that the agent failed.
- **Invocation Count**: *CountStatistic* - The number of times the AgentManager has been invoked.
- **Partition Count**: *CountStatistic* - The number of partitions to which the agent is sent.

The root element of the agentManagerModule, "root", serves as the entry point to the AgentManager statistics. This root element has ObjectGrids as its child elements, ObjectGrids have maps as their child elements, which finally have AgentManager instances as their child elements and leaf nodes of the tree. Every AgentManager instance has statistics.

Figure 7. agentManagerModule structure

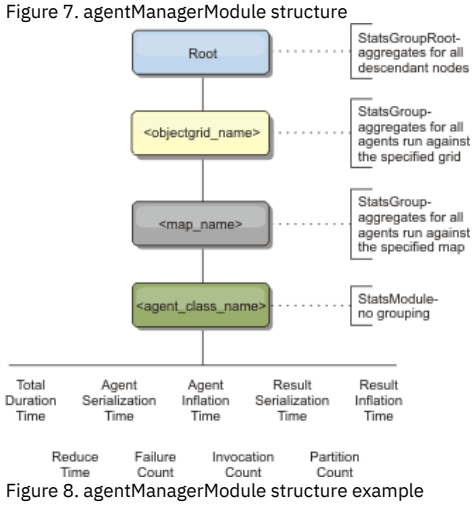
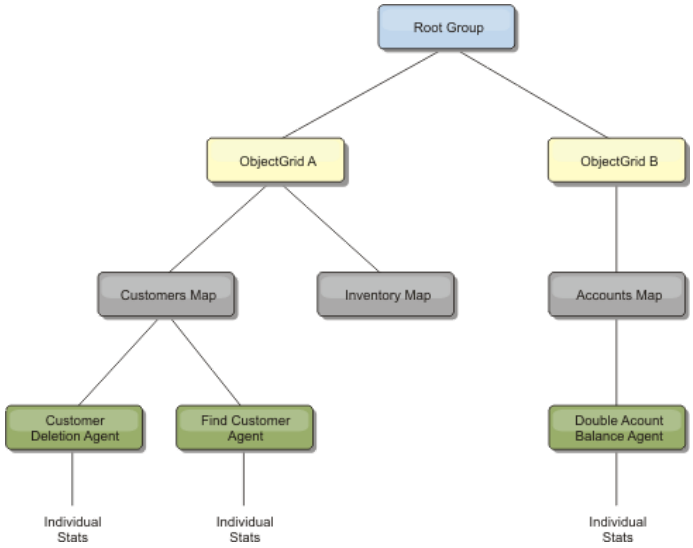


Figure 8. agentManagerModule structure example



queryModule

The queryModule contains statistics that are related to eXtreme Scale queries:

- **Plan Creation Time**: *TimeStatistic* - The amount of time to create the query plan.
- **Execution Time**: *TimeStatistic* - The amount of time to run the query.
- **Execution Count**: *CountStatistic* - The number of times the query has been run.
- **Result Count**: *CountStatistic* - The count for each the result set of each query run.
- **FailureCount**: *CountStatistic* - The number of times the query has failed.

The root element of the queryModule, "root", serves as the entry point to the Query Statistics. This root element has ObjectGrids as its child elements, which have Query objects as their child elements and leaf nodes of the tree. Every Query instance has the three listed statistics.

Figure 9. queryModule structure

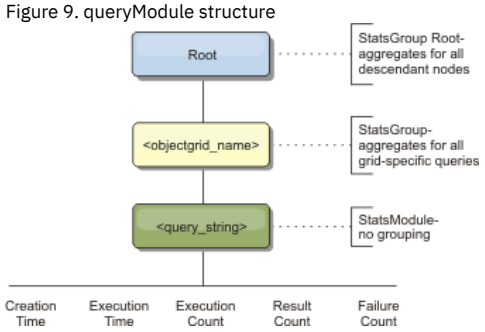
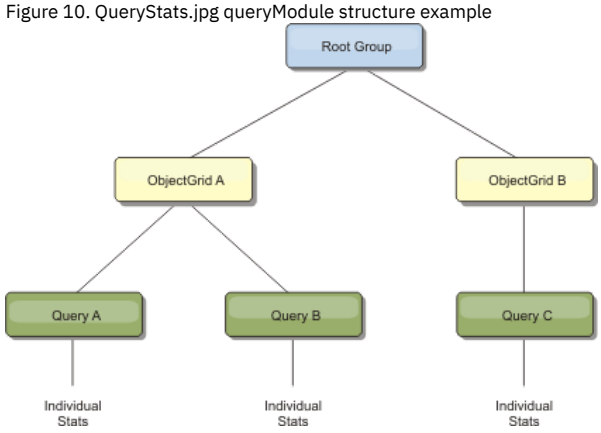


Figure 10. QueryStats.jpg queryModule structure example



Accessing Managed Beans (MBeans) using the wsadmin tool

You can use the wsadmin utility provided in WebSphere® Application Server to access managed bean (MBean) information.

Procedure

Run the wsadmin tool from the bin directory in your WebSphere Application Server installation. The following example retrieves a view of the current shard placement in a dynamic eXtreme Scale. You can run the wsadmin tool from any installation where eXtreme Scale is running. You do not have to run the wsadmin tool on the catalog service.

```
$ wsadmin.sh -lang jython
wsadmin>placementService = AdminControl.queryNames
        ("com.ibm.websphere.objectgrid:*,type=PlacementService")
wsadmin>print AdminControl.invoke(placementService,
        "listObjectGridPlacement","library ms1")

<objectGrid name="library" mapSetName="ms1">
  <container name="container-0" zoneName="DefaultDomain"
               hostName="host1.company.org" serverName="server1">
     <shard type="Primary" partitionName="0"/>
     <shard type="SynchronousReplica" partitionName="1"/>
  </container>
  <container name="container-1" zoneName="DefaultDomain"
               hostName="host2.company.org" serverName="server2">
     <shard type="SynchronousReplica" partitionName="0"/>
     <shard type="Primary" partitionName="1"/>
  </container>
  <container name="UNASSIGNED" zoneName="_ibm_SYSTEM"
               hostName="UNASSIGNED" serverName="UNNAMED">
    <shard type="SynchronousReplica" partitionName="0"/>
    <shard type="AsynchronousReplica" partitionName="0"/>
  </container>
</objectGrid>
```

Related concepts:
Statistics overview
Related tasks:
Accessing Managed Beans (MBeans) programmatically
Accessing Managed Beans (MBeans) programmatically
Monitoring server statistics with managed beans (MBeans)
Monitoring with the xscmd utility
Related reference:
Administering with Managed Beans (MBeans)
Related information:
API documentation: Package com.ibm.websphere.objectgrid.management
Interface PlacementServiceMBean

Monitoring server statistics with managed beans (MBeans)

You can used managed beans (MBeans) to track statistics in your environment.

Before you begin

For the attributes to be recorded, you must enable statistics. You can enable statistics on the server, or enable HTTP session statistics to track attributes on your client application. For more information on how to enable HTTP session statistics, see xref.

You can enable statistics in one of the following ways:

- **With the server properties file:**
  You can enable statistics in the server properties file with a key-value entry of `statsSpec=<StatsSpec>`. Some examples of possible settings follow:
  - To enable all statistics, use `statsSpec=all=enabled`
  - To enable only ObjectGrid statistics, use `statsSpec=og.all=enabled`. To see a description of all possible statistics specifications, see the StatsSpec API.

  For more information about the server properties file, see Server properties file.
- **With a managed bean:**
  You can enable statistics using the StatsSpec attribute on the ObjectGrid MBean. For more information, see the StatsSpec API.
- **Programmatically:**
  You can also enable statistics programmatically with the StatsAccessor interface, which is retrieved with the StatsAccessorFactory class. Use this interface in a client environment or when you need to monitor a data grid that is running in the current process.

Procedure

- **Access MBean statistics using the wsadmin tool.**
  For more information, see Accessing Managed Beans (MBeans) using the wsadmin tool.
- **Access MBean statistics programmatically.**
  For more information, see Accessing Managed Beans (MBeans) programmatically.

Example

For an example of how to use managed beans, see Sample: xsadmin utility.

Related concepts:
Statistics overview
Monitoring with vendor tools
Related tasks:
Monitoring with the web console
Monitoring with CSV files
Enabling statistics
Monitoring with the xscmd utility
Monitoring with WebSphere Application Server PMI
Monitoring eXtreme Scale information in DB2
Accessing Managed Beans (MBeans) using the wsadmin tool
Accessing Managed Beans (MBeans) programmatically
Monitoring with the xscmd utility
Related reference:
Administering with Managed Beans (MBeans)
Related information:
API documentation: Package com.ibm.websphere.objectgrid.management
Interface PlacementServiceMBean

Monitoring with vendor tools

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.

- Monitoring with the WebSphere eXtreme Scale Agent for IBM Tivoli Monitoring
  IBM Tivoli Monitoring is a feature-rich monitoring solution that you can use to monitor databases, operating systems and servers in distributed and host environments. WebSphere eXtreme Scale includes a customized agent that you can use to introspect eXtreme Scale management beans. This solution works effectively for eXtreme Scale in a stand-alone environment and a WebSphere Application Server deployment.
- Monitoring eXtreme Scale applications with CA Wily Introscope
  CA Wily Introscope is a third-party management product that you can use to detect and diagnose performance problems in enterprise application environments. eXtreme Scale includes details on configuring CA Wily Introscope to introspect select portions of the eXtreme Scale run time to quickly view and validate eXtreme Scale applications. CA Wily Introscope works effectively for both stand-alone and WebSphere Application Server deployments.
- Monitoring eXtreme Scale with Hyperic HQ
  Hyperic HQ is a third-party monitoring solution that is available freely as an open source solution or as an enterprise product. WebSphere eXtreme Scale includes a plug-in that allows Hyperic HQ agents to discover eXtreme Scale container servers and to report and aggregate statistics using eXtreme Scale management beans. You can use Hyperic HQ to monitor stand-alone eXtreme Scale deployments.

Related concepts:
Statistics overview
Interoperability with other products
Installation topologies
Tuning garbage collection with WebSphere Real Time
Related tasks:
Monitoring with the web console
Monitoring with CSV files
Enabling statistics
Monitoring with the xscmd utility
Monitoring with WebSphere Application Server PMI
Monitoring server statistics with managed beans (MBeans)
Monitoring eXtreme Scale information in DB2
Configuring the HTTP session manager for various application servers
Configuring HTTP session manager with WebSphere Portal
Configuring the HTTP session manager with WebSphere Application Server
Configuring WebSphere eXtreme Scale with WebSphere Application Server
Related information:
Configure WebSphere Commerce to use WebSphere eXtreme Scale for dynamic cache to improve performance and scale
WebSphere Business Process Management and Connectivity integration

Monitoring with the WebSphere eXtreme Scale Agent for IBM Tivoli Monitoring

IBM® Tivoli® Monitoring is a feature-rich monitoring solution that you can use to monitor databases, operating systems and servers in distributed and host environments. WebSphere® eXtreme Scale includes a customized agent that you can use to introspect eXtreme Scale management beans. This solution works effectively for eXtreme Scale in a stand-alone environment and a WebSphere Application Server deployment.

Before you begin

- Install WebSphere eXtreme Scale Version 7.0.0 or later.
  Also, statistics must be enabled in order to collect statistical data from WebSphere eXtreme Scale servers. Various options for enabling statistics are described in Monitoring server statistics with managed beans (MBeans) and Sample: xsadmin utility

- Install IBM Tivoli Monitoring Version 6.2.1 with fix pack 2 or later.
- Install the Tivoli OS agent on each server or host on which eXtreme Scale servers run.
- Install the monitoring agent for eXtreme Scale, which you can download for free from the IBM Integrated Service Management Library.

Complete the following steps to install and configure the monitoring agent for eXtreme Scale for IBM Tivoli Monitoring:

Procedure

1. Install WebSphere eXtreme Scale Agent for IBM Tivoli Monitoring.
   Download the installation image and extract its files to a temporary directory.

2. Install the monitoring agent for application support files.
   Install the monitoring agent for WebSphere eXtreme Scale application support on each of the following deployments.

   - Tivoli Enterprise Portal Server (TEPS)
   - Tivoli Enterprise Desktop Client (TEPD)
   - Tivoli Enterprise Monitoring Server (TEMS)

   a. From the temporary directory that you created, start a new command window and run the appropriate executable file for your platform. The installation script automatically detects your Tivoli deployment type (TEMS, TEPD, or TEPS). You can install any type on a single host or on multiple hosts; and all of the three deployment types require the installation of **Monitoring Agent for WebSphere eXtreme Scale** application support files.
   b. In the Installer window, verify that the selections for the Tivoli Components deployed are correct. Click Next.
   c. If you are prompted, submit your administrative credentials followed by the hostname. Click Next.
   d. Select the **Monitoring Agent for WebSphere eXtreme Scale**. Click Next.
   e. You are notified of what installation actions are to be performed. Click Next, and you can see the progress of the installation until completion.

   After completing the procedure, all application support files are installed.
3. Install the agent on each of the eXtreme Scale nodes.
   You install a Tivoli OS agent on each of the computers. You do not need to configure or start this agent. Use the same installation image from the previous step to run the platform specific executable file.

   As a guideline, you need to install only one agent per host. Each agent is capable of supporting many instances of eXtreme Scale servers. For best performance, use one agent instance for monitoring about 50 eXtreme Scale servers.

   a. From the installation wizard welcome screen, click **Next** to open the screen to specify installation path information.
   b. For the Tivoli Monitoring installation directory field, enter or browse to C:\IBM\ITM (or /opt/IBM/ITM). Then for the Enter the location where your installable media... field, verify that the displayed value is correct and click Next.
   c. Select the components you want to add, such as Perform a local install of the solution... and click Next.
   d. Select the applications for which to add support for by selecting the application, such as **Monitoring Agent for WebSphere eXtreme Scale**, and click Next.
   e. You can see the progress until application support is added successfully.

   Note: Repeat these steps on each of the eXtreme Scale nodes. You can also use silent installation. See the IBM Tivoli Monitoring Information Center for more information about silent installation.
4. Configure the monitoring agent for eXtreme Scale.
   Each of the agents installed need to be configured to monitor any catalog server, eXtreme Scale server, or both.

   The steps to configure Windows and UNIX platforms are different. Configuration for the Windows platform is completed with the **Manage Tivoli Monitoring Services** user interface. Configuration for UNIX platforms is command-line based.

   Windows Use the following steps to initially configure the agent on Windows

   a. From the **Manage Tivoli Enterprise Monitoring Services** window, click Start > All Programs > IBM Tivoli Monitoring > Manage Tivoli Monitoring Services.
   b. Right click on **Monitoring Agent for WebSphere eXtreme Scale** and select Configure using defaults, which opens a window to create a unique instance of the agent.
   c. Choose a unique name: for example, `instance1`, and click Next.

   - If you plan to monitor stand-alone eXtreme Scale servers, complete the following steps:
     - Update the Java™ parameters, ensure that the Java Home value is correct. JVM arguments can be left empty. Click Next.
     - Select the type of MBean server connection type, Use `JSR-160-Complaint Server` for stand-alone eXtreme Scale servers. Click Next.
     - If security is enabled, update User ID and Password values. Leave the JMX service URL value as is. You override this value later. Leave the JMX Class Path Information field as it is. Click Next.

     To configure the servers for the agent on Windows, complete the following steps:
     - Set up subnode instances of eXtreme Scale servers in the **WebSphere eXtreme Scale Grid Servers** pane. If no container servers exist on your computer, click **Next** to proceed to the catalog service pane.
     - If multiple eXtreme Scale container servers exist on your computer, configure the agent to monitor each one server.
     - You can add as many eXtreme Scale servers as you require, if their names and ports are unique, by clicking New. (When an eXtreme Scale server is started, a JMXPort value must be specified.)
     - After you configure the container servers, click **Next**, which brings you to the **WebSphere eXtreme Scale Catalog Servers** pane.
     - If you have no catalog servers, click OK. If you have catalog servers, add a new configuration for each server, as you did with the container servers. Again, choose a unique name, preferably the same name that is used when starting the catalog service. Click OK to finish.
   - If you plan to monitor servers for the agent on eXtreme Scale servers that are embedded within a WebSphere Application Server process, complete the following steps:
     - Update the Java parameters, ensure that the Java Home value is correct. JVM arguments can be left empty. Click Next.
     - Select the MBean server connection type. Select the WebSphere Application Server version that is appropriate for your environment. Click Next.
     - Ensure that the WebSphere Application Server information in the panel is correct. Click Next.
     - Add only one subnode definition. Give the subnode definition a name, but do not update the port definition. Within WebSphere Application Server environment, data can be collected from all the application server that are managed by the node agent that is running on the computer. Click Next.

- If there no catalog servers exist in the environment, click OK. If you have catalog servers, add a new configuration for each catalog server, similarly to the container servers. Choose a unique name for the catalog service, preferably the same name that you use when starting the catalog service. Click OK to finish.

Note: The container servers do not need to be collocated with the catalog service.
Now that the agent and servers are configured and ready, on the next window, right click on `instance1` to start the agent.

UNIX To configure the agent on the UNIX platform on the command line, complete the following steps:

An example follows for stand-alone servers that uses a JSR160 Compliant connection type. The example shows three eXtreme Scale containers on the single host (rhea00b02) and the JMX listener addresses are 15000,15001 and 15002 respectively. There are no catalog servers.

Output from the configuration utility displays in *`monospace italics`*, while the user response is in **`monospace bold`**. (If no user response was required, the default was selected by pressing the enter key.)

```
rhea00b02 # ./itmcmd config -A xt
Agent configuration started...
Enter instance name (default is: ): inst1
Edit "Monitoring Agent for WebSphere eXtreme Scale" settings? [ 1=Yes, 2=No ] (default is: 1):
Edit 'Java' settings? [ 1=Yes, 2=No ] (default is: 1):
Java home (default is: C:\Program Files\IBM\Java50): /opt/OG61/java
Java trace level [ 1=Error, 2=Warning, 3=Information, 4=Minimum Debug, 5=Medium Debug, 6=Maximum Debug,
                                7=All ] (default is: 1):
JVM arguments (default is: ):
Edit 'Connection' settings? [ 1=Yes, 2=No ] (default is: 1):
MBean server connection type [ 1=JSR-160-Compliant Server, 2=WebSphere Application Server version 6.0,
3=WebSphere Application Server version 6.1, 4=WebSphere Application Server version 7.0 ] (default is: 1): 1
Edit 'JSR-160-Compliant Server' settings? [ 1=Yes, 2=No ] (default is: 1):
JMX user ID (default is: ):
Enter JMX password (default is: ):
Re-type : JMX password (default is: ):
JMX service URL (default is: service:jmx:rmi:///jndi/rmi://localhost:port/objectgrid/MBeanServer):
----------------------------------------
JMX Class Path Information
JMX base paths (default is: ):
JMX class path (default is: ):
JMX JAR directories (default is: ):
Edit 'WebSphere eXtreme Scale Catalog Service' settings? [ 1=Yes, 2=No ] (default is: 1): 2
Edit 'WebSphere eXtreme Scale Grid Servers' settings? [ 1=Yes, 2=No ] (default is: 1): 1
No 'WebSphere eXtreme Scale Grid Servers' settings available?
Edit 'WebSphere eXtreme Scale Grid Servers' settings, [1=Add, 2=Edit, 3=Del, 4=Next, 5=Exit] (default is: 4): 1
WebSphere eXtreme Scale Grid Servers (default is: ): rhea00b02_c0
JMX service URL (default is: service:jmx:rmi:///jndi/rmi://localhost:<port>/objectgrid/MBeanServer):
service:jmx:rmi:///jndi/rmi://localhost:15000/objectgrid/MBeanServer

'WebSphere eXtreme Scale Grid Servers' settings:  WebSphere eXtreme Scale Grid Servers=ogx
Edit 'WebSphere eXtreme Scale Grid Servers' settings, [1=Add, 2=Edit, 3=Del, 4=Next, 5=Exit] (default is: 4): 1
WebSphere eXtreme Scale Grid Servers (default is: ): rhea00b02_c1
JMX service URL (default is: service:jmx:rmi:///jndi/rmi://localhost:<port>/objectgrid/MBeanServer):
service:jmx:rmi:///jndi/rmi://localhost:15001/objectgrid/MBeanServer

'WebSphere eXtreme Scale Grid Servers' settings:  WebSphere eXtreme Scale Grid Servers= rhea00b02_c1
Edit 'WebSphere eXtreme Scale Grid Servers' settings, [1=Add, 2=Edit, 3=Del, 4=Next, 5=Exit] (default is: 4): 1
WebSphere eXtreme Scale Grid Servers (default is: ): rhea00b02_c2
JMX service URL (default is: service:jmx:rmi:///jndi/rmi://localhost:<port>/objectgrid/MBeanServer):
service:jmx:rmi:///jndi/rmi://localhost:15002/objectgrid/MBeanServer

'WebSphere eXtreme Scale Grid Servers' settings:  WebSphere eXtreme Scale Grid Servers= rhea00b02_c2
Edit 'WebSphere eXtreme Scale Grid Servers' settings, [1=Add, 2=Edit, 3=Del, 4=Next, 5=Exit] (default is: 4): 5

Will this agent connect to a TEMS? [1=YES, 2=NO] (Default is: 1):
TEMS Host Name (Default is: rhea00b00):

Network Protocol [ip, sna, ip.pipe or ip.spipe] (Default is: ip.pipe):

     Now choose the next protocol number from one of these:
     - ip
     - sna
     - ip.spipe
     - 0 for none
Network Protocol 2 (Default is: 0):
IP.PIPE Port Number (Default is: 1918):
Enter name of KDC_PARTITION (Default is: null):

Configure connection for a secondary TEMS? [1=YES, 2=NO] (Default is: 2):
Enter Optional Primary Network Name or 0 for "none" (Default is: 0):
Agent configuration completed...
```

The previous example creates an agent instance called “inst1”, and updates the Java Home settings. The eXtreme Scale container servers are configured, but the catalog service is not configured.

Note: The previous procedure creates a text file of the following format in the directory: <ITM_install>/config/<host>_xt_<instance name>.cfg.
Example: rhea00b02_xt_inst1.cfg

It is best to edit this file with your choice of plain text editor. An example of the content of such the file follows:

```
INSTANCE=inst2 [SECTION=KQZ_JAVA [ { JAVA_HOME=/opt/OG61/java }  { JAVA_TRACE_LEVEL=ERROR }  ]
SECTION=KQZ_JMX_CONNECTION_SECTION [ { KQZ_JMX_CONNECTION_PROPERTY=KQZ_JMX_JSR160_JSR160 }  ]
SECTION=KQZ_JMX_JSR160_JSR160 [ { KQZ_JMX_JSR160_JSR160_CLASS_PATH_TITLE= }
```

```
{ KQZ_JMX_JSR160_JSR160_SERVICE_URL=service:jmx:rmi:///jndi/rmi://localho
st:port/objectgrid/MBeanServer }  { KQZ_JMX_JSR160_JSR160_CLASS_PATH_SEPARATOR= }  ]
SECTION=OGS:rhea00b02_c1 [ { KQZ_JMX_JSR160_JSR160_SERVICE_URL=service:jmx:
rmi:///jndi/rmi://localhost:15001/objectgrid/MBeanServer }  ]
SECTION=OGS:rhea00b02_c0 [ { KQZ_JMX_JSR160_JSR160_SERVICE_URL=service:jmx:
rmi:///jndi/rmi://localhost:15002/objectgrid/MBeanServer }  ]
SECTION=OGS:rhea00b02_c2 [ { KQZ_JMX_JSR160_JSR160_SERVICE_URL=service:jmx:
rmi:///jndi/rmi://localhost:15002/objectgrid/MBeanServer }  ]]
```

An example that shows a configuration on a WebSphere Application Server deployment follows:

```
rhea00b02 # ./itmcmd config -A xt
Agent configuration started...
Enter instance name (default is: ): inst1
Edit "Monitoring Agent for WebSphere eXtreme Scale" settings? [ 1=Yes, 2=No ] (default is: 1): 1
Edit 'Java' settings? [ 1=Yes, 2=No ] (default is: 1): 1
Java home (default is: C:\Program Files\IBM\Java50): /opt/WAS61/java
Java trace level [ 1=Error, 2=Warning, 3=Information, 4=Minimum Debug, 5=Medium Debug, 6=Maximum Debug,
                                  7=All ] (default is: 1):
JVM arguments (default is: ):
Edit 'Connection' settings? [ 1=Yes, 2=No ] (default is: 1):
MBean server connection type [ 1=JSR-160-Compliant Server, 2=WebSphere Application Server version 6.0,
3=WebSphere Application Server version 6.1, 4=WebSphere Application Server version 7.0 ] (default is: 1): 4
Edit 'WebSphere Application Server version 7.0' settings? [ 1=Yes, 2=No ] (default is: 1):WAS user ID (default is:
):
Enter WAS password (default is: ):
Re-type : WAS password (default is: ):
WAS host name (default is: localhost): rhea00b02
WAS port (default is: 2809):
WAS connector protocol [ 1=rmi, 2=soap ] (default is: 1):
WAS profile name (default is: ): default
----------------------------------------
WAS Class Path Information
WAS base paths (default is: C:\Program Files\IBM\WebSphere\AppServer;/opt/IBM/WebSphere/AppServer): /opt/WAS61
WAS class path (default is:
runtimes/com.ibm.ws.admin.client_6.1.0.jar;runtimes/com.ibm.ws.ejb.thinclient_7.0.0.jar):
WAS JAR directories (default is: lib;plugins):
Edit 'WebSphere eXtreme Scale Grid Servers' settings? [ 1=Yes, 2=No ] (default is: 1):
No 'WebSphere eXtreme Scale Grid Servers' settings available?
Edit 'WebSphere eXtreme Scale Grid Servers' settings, [1=Add, 2=Edit, 3=Del, 4=Next, 5=Exit] (default is: 4): 1
WebSphere eXtreme Scale Grid Servers (default is: ): rhea00b02
JMX service URL (default is: service:jmx:rmi:///jndi/rmi://localhost:<port>/objectgrid/MBeanServer):

'WebSphere eXtreme Scale Grid Servers' settings:  WebSphere eXtreme Scale Grid Servers=rhea00b02
Edit 'WebSphere eXtreme Scale Grid Servers' settings, [1=Add, 2=Edit, 3=Del, 4=Next, 5=Exit] (default is: 4): 5
Edit 'WebSphere eXtreme Scale Catalog Service' settings? [ 1=Yes, 2=No ] (default is: 1): 2
Will this agent connect to a TEMS? [1=YES, 2=NO] (Default is: 1):
TEMS Host Name (Default is: rhea00b02):

Network Protocol [ip, sna, ip.pipe or ip.spipe] (Default is: ip.pipe):

     Now choose the next protocol number from one of these:
     - ip
     - sna
     - ip.spipe
     - 0 for none
Network Protocol 2 (Default is: 0):
IP.PIPE Port Number (Default is: 1918):
Enter name of KDC_PARTITION (Default is: null):

Configure connection for a secondary TEMS? [1=YES, 2=NO] (Default is: 2):
Enter Optional Primary Network Name or 0 for "none" (Default is: 0):
Agent configuration completed...
rhea00b02 #
```

For WebSphere Application Server deployments, you do not need to create multiple sub nodes. The monitoring agent for eXtreme Scale connects to the node agent to gather all the information from application servers for which it is responsible.
`SECTION=CAT` signifies a catalog service line whereas `SECTION=OGS` signifies an eXtreme Scale server configuration line.

5. Configure the JMX port for all eXtreme Scalecontainer servers.
When eXtreme Scale container servers are started, without specifying the -JMXServicePort argument, an MBean server is assigned a dynamic port. The agent needs to know in advance with which JMX port to communicate. The agent does not work with dynamic ports.

   When you start the servers, you must specify the `-JMXServicePort <port_number>` argument when you start the eXtreme Scale server using the start server command. Running this command ensures that the JMX server within the process listens to a static pre-defined port.

   For the previous examples in a UNIX installation, two eXtreme Scale servers need to be started with ports set:
   a. "-JMXServicePort" "15000" (for rhea00b02_c0)
   b. "-JMXServicePort" "15001" (for rhea00b02_c1)
   a. Start the eXtreme Scale agent.
      Assuming the `inst1` instance was created, as in the previous example, issue the following commands.

      i. `cd <ITM_install>/bin`
      ii. `itmcmd agent -o inst1 start xt`
   b. Stop the eXtreme Scale agent.
      Assuming “inst1” was the instance created, as in the previous example, issue the following commands.

      i. `cd <ITM_install>/bin`

ii. `itmcmd agent -o inst1 stop xt`

6. Enable Statistics for all eXtreme Scale container servers.
   The agent uses the eXtreme Scale statistics MBeans to record statistics. The eXtreme Scale statistics specification must be enabled using one of the following methods.
   - Configure server properties to enable all statistics when the container servers are started: all=enabled.
   - Use the xsadmin sample utility to enable statistics for all active containers using the -setstatsspec all=enabled parameters.

Results

After all servers are configured and started, MBeans data is displayed on the IBM Tivoli Portal console. Predefined workspaces show graphs and data metrics at each node level.

The following workspaces are defined: **eXtreme Scale Grid Servers** node for all nodes monitored.

- eXtreme Scale Transactions View
- eXtreme Scale Primary Shard View
- eXtreme Scale Memory View
- eXtreme Scale ObjectMap View

You can also configure your own workspace. For more information, see the information about customizing workspaces in the IBM Tivoli Monitoring Information Center.

Monitoring eXtreme Scale applications with CA Wily Introscope

CA Wily Introscope is a third-party management product that you can use to detect and diagnose performance problems in enterprise application environments. eXtreme Scale includes details on configuring CA Wily Introscope to introspect select portions of the eXtreme Scale run time to quickly view and validate eXtreme Scale applications. CA Wily Introscope works effectively for both stand-alone and WebSphere® Application Server deployments.

Overview

To monitor eXtreme Scale applications with CA Wily Introscope, you must put settings into the ProbeBuilderDirective (PBD) files that give you access to the monitoring information for eXtreme Scale.
Attention: The instrumentation points for Introscope might change with each fix pack or release. When you install a new fix pack or release, check the documentation for any changes in the instrumentation points.
You can configure CA Wily Introscope ProbeBuilderDirective (PBD) files to monitor your eXtreme Scale applications. CA Wily Introscope is an application management product with which you can proactively detect, triage, and diagnose performance problems in your complex, composite, and web application environments. You can use one or more of the following settings in your PBD file to monitor the catalog service.

Catalog

HAControllerImpl
: The HAControllerImpl class handles core group lifecycle and feedback events. You can monitor this class to get an indication of the core group structure and changes.

CatalogServiceImpl
: The CatalogServiceImpl class handles the catalog routing events. You can monitor this class to get an indication of route changes and updates.

PlacementServiceImpl
: The PlacementServiceImpl class coordinates the containers. You can use the methods on this class to monitor server join and placement events.

BalanceGridEventListener
: The BalanceGridEventListener class controls the catalog leadership. You can monitor this class to get an indication of which catalog service is acting as the leader.

Container

ServerAgent
: The ServerAgent class is responsible for communicating core group events with the catalog service. You can monitor various heartbeat calls to spot major events.

ShardImpl
: The ShardImpl class has the processMessage method. The processMessage method is the method for client requests. With this method, you can get server-side response time and request counts. By watching the counts across all the servers and monitoring heap utilization, you can determine if the grid is balanced.

ObjectGridImpl
: The ObjectGridImpl class has the queryRevision and applyRevision methods. The queryRevision method is used by primaries to build an updated revision package to send to its replicas. The applyRevision method is used by the replicas to apply the updated revisions to their ObjectGrid.

BaseMap
: The BaseMap class has the evictMapEntries method that is called when the evictor wants to remove entries from the map.

Client

ORBClientCoreMessageHandler

The ORBClientCoreMessageHandler class is responsible for sending application requests to the containers. You can monitor the sendMessage method for client response time and number of requests.

ClusterStore
: The ClusterStore class holds the routing information on the client side.

BaseMap
: The BaseMap class has the evictMapEntries method that is called when the evictor wants to remove entries from the map.

SelectionServiceImpl
: The SelectionServiceImpl class makes the routing decisions. If the client is making failover decisions, you can use this class to see the actions that are completed from the decisions.

SessionImpl
: The SessionImpl class has the getMap method. The getMap method is called by most of client applications and can be monitored for response time and number of requests.

ObjectGridImpl
: The ObjectGridImpl class has the getSession method that you can monitor to see the number of requests to this method.

```
Catalog server classes
TraceOneMethodOfClass: com.ibm.ws.objectgrid.hamanager.HAControllerImpl changeDefinedCompleted
BlamePointTracerDifferentMethods "WXS Catalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.hamanager.HAControllerImpl viewChangeCompleted
BlamePointTracerDifferentMethods "WXS Catalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.hamanager.HAControllerImpl viewAboutToChange
BlamePointTracerDifferentMethods "WXS Catalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.catalog.placement.CatalogServiceImpl heartbeat
BlamePointTracerDifferentMethods "WXS Catalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.catalog.placement.CatalogServiceImpl classifyServer
BlamePointTracerDifferentMethods "WXS Catalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.catalog.placement.CatalogServiceImpl importRouteInfo
BlamePointTracerDifferentMethods "WXS Catalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.catalog.placement.CatalogServiceImpl getObjectGridRouteInfo
BlamePointTracerDifferentMethods "WXS Catalog|{classname}|
{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.catalog.placement.CatalogServiceImpl registerContainer
BlamePointTracerDifferentMethods "WXS Catalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.catalog.placement.PlacementServiceImpl joinPlacementGroup
BlamePointTracerDifferentMethods "WXS Catalog|{classname}|
{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.catalog.placement.PlacementServiceImpl place
BlamePointTracerDifferentMethods "WXS Catalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.catalog.placement.BalanceGridEventListener shardActivated
BlamePointTracerDifferentMethods "WXS Catalog|{classname}|
{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.catalog.placement.BalanceGridEventListener shardDeactivate
BlamePointTracerDifferentMethods "WXS Catalog|{classname}|
{method}"

Container server classes
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent membershipChanged BlamePointTracerDifferentMethods
"WXS Container|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent memberActivated BlamePointTracerDifferentMethods "WXS
Container|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent memberDeActivated BlamePointTracerDifferentMethods
"WXS Container|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent batchProcess BlamePointTracerDifferentMethods "WXS
Container|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent changeDefinedCompleted
BlamePointTracerDifferentMethods "WXS Container|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.ShardImpl processMessage BlamePointTracerDifferentMethods "WXS Container|
{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.ObjectGridImpl queryRevision BlamePointTracerDifferentMethods "WXS
Container|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.ObjectGridImpl applyRevision BlamePointTracerDifferentMethods "WXS
Container|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.map.BaseMap evictMapEntries BlamePointTracerDifferentMethods "WXS
Container|{classname}|{method}"

Client classes
TraceOneMethodOfClass: com.ibm.ws.objectgrid.client.ORBClientCoreMessageHandler sendMessage
BlamePointTracerDifferentMethods "WXS Client|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.corba.cluster.ClusterStore bootstrap BlamePointTracerDifferentMethods "WXS
Client|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.map.BaseMap evictMapEntries BlamePointTracerDifferentMethods "WXS Client|
{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.cluster.orb.routing.SelectionServiceImpl routeFailed
BlamePointTracerDifferentMethods "WXS Client|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.SessionImpl getMap BlamePointTracerDifferentMethods "WXS Client|
{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.ObjectGridImpl getSession BlamePointTracerDifferentMethods "WXS Client|
{classname}|{method}"
```

Monitoring eXtreme Scale with Hyperic HQ

Hyperic HQ is a third-party monitoring solution that is available freely as an open source solution or as an enterprise product. WebSphere® eXtreme Scale includes a plug-in that allows Hyperic HQ agents to discover eXtreme Scale container servers and to report and aggregate statistics using eXtreme Scale management beans. You can use Hyperic HQ to monitor stand-alone eXtreme Scale deployments.

Before you begin

- This set of instructions is for Hyperic Version 4.0. If you have a newer version of Hyperic, see the Hyperic documentation for information such as the path names and how to start agents and servers.
- Download the Hyperic server and agent installations. One server installation must be running. To detect all of the eXtreme Scale servers, a Hyperic agent must be running on each machine on which an eXtreme Scale server is running. See the Hyperic website for download information and documentation support.
- You must have access to the objectgrid-plugin.xml and hqplugin.jar files. These files are in the *wxs_install_root*/hyperic/etc directory.

About this task

By integrating eXtreme Scale with Hyperic HQ monitoring software, you can graphically monitor and display metrics about the performance of your environment. You set up this integration by using a plug-in implementation on each agent.

Procedure

1. Start your eXtreme Scale servers. The Hyperic plug-in looks at the local processes to attach to the Java™ virtual machines that are running eXtreme Scale. To properly attach to the Java virtual machines, each server must be started with the -jmxServicePort option. For information about starting servers with the -jmxServicePort option, see Starting and stopping stand-alone servers.
2. Put the extremescale-plugin.xml file and the wxshyperic.jar file in the appropriate server and agent plug-in directories in your Hyperic configuration. To integrate with Hyperic, both the agent and server installations must have access to the plug-in and Java archive (JAR) files. Although the server can dynamically swap configurations, you should complete the integration before you start any of the agents.
    a. Place the extremescale-plugin.xml file in the server plugin directory, which is at the following location:

    ***hyperic_home*/server_home/hq-engine/server/default/deploy/hq.ear/hq-plugins**

    b. Place the extremescale-plugin.xml file in the agent plugin directory, which is at the following location:

    ***agent_home*/bundles/gent-4.0.2-939/pdk/plugins**

    c. Put the wshyperic.jar file in the agent lib directory, which is at the following location

    ***agent_home*/bundles/gent-4.0.2-939/pdk/lib**

3. Configure the agent. The agent.properties file serves as a configuration point for the agent runtime. This property is in the *agent_home*/conf directory. The following keys are optional, but of importance to the eXtreme Scale plug-in:

    - **autoinventory.defaultScan.interval.millis=*<time_in_milliseconds>***

      Sets the interval in milliseconds between Agent discoveries.

    - **log4j.logger.org.hyperic.hq.plugin.extremescale.XSServerDetector=DEBUG**

      : Enables verbose debug statements from the eXtreme Scale plug-in.
    - `username=<username>`: Sets the Java Management Extensions (JMX) user name if security is enabled.
    - `password=<password>`: Sets the JMX password if security is enabled.
    - `sslEnabled=<true|false>`: Tells the plug-in whether or not to use Secure Sockets Layer (SSL). The value is `false` by default.
    - `trustPath=<path>`: Sets the trust path for the SSL connection.
    - `trustType=<type>`: Sets the trust type for the SSL connection.
    - `trustPass=<password>`: Sets the trust password for the SSL connection.
4. Start the agent discovery. The Hyperic agents send discoveries and metrics information to the server. Use the server to customize data views and group logical inventory objects to generate useful information. After the server is available, you must run the launch script or start the Windows service for the agent:
    - Linux *agent_home*/bin/hq-agent.sh start
    - Windows Start the agent with the Windows service.

    After you start the agents, the servers are detected and groups are configured. You can log into the server console and choose which resources to add to the inventory database for the server. The server console is at the following URL by default: `http://<server_host_name>:7080/`
5. Statistics must be enabled for Hyperic to collect statistical data.
    Use the SetStatsSpec control action on the Hyperic console for eXtreme Scale. Navigate to the resource, then use the Control Action drop-down list on the Control tabbed page to specify a `SetStatsSpec` setting with `ALL=enabled` in the Control Arguments text box.

    Catalog servers are not detected by the filter set in the Hyperic console. See the information about the statsSpec property in Server properties file, which enable statistics as soon as the containers start. Various options for enabling statistics are described in Monitoring server statistics with managed beans (MBeans) and Sample: xsadmin utility

6. Monitor servers with the Hyperic console. After the servers are added to the inventory model, their services are no longer needed.
    - **Dashboard view**: When you viewed the resource detection events, you logged into the main dashboard view. The dashboard view is a generic view that acts as a message center that you can customize. You can export graphs or inventory objects to this main dashboard.
    - **Resources view**: You can query and view the entire inventory model from this page. After the services have been added, you can see every eXtreme Scale server properly labeled and listed together under the servers section. You can click on the individual servers to see the basic metrics.
7. View the entire server inventory on the Resource View page. On this page, you can then select multiple ObjectGrid servers and group them together. After you group a set of resources, their common metrics can be graphed to show overlays and differences among group members. To display an overlay, select

the metrics on the display of your Server Group. The metric then displays in the charting area. To display an overlay for all group members, click the underlined metric name. You can export any of the charts, node views, and comparative overlays to the main dashboard with the Tools menu.

Monitoring eXtreme Scale information in DB2

When the JPALoader or JPAEntityLoader is used with DB2® as the back-end database, eXtreme Scale-specific information can be passed to DB2. You can view this information by a performance monitor tool such as DB2 Performance Expert to monitor the eXtreme Scale applications that are accessing the database.

Before you begin

See Collecting trace for more information about the different methods for setting trace that you can use.

About this task

When the loader is configured to use DB2 as the back-end database, the following eXtreme Scale information can be passed to DB2 for monitoring purposes:

- **User:** Specifies the name of the user that authenticates to eXtreme Scale. When basic authentication is not used, the principals from the authentication are used.
- **Workstation Name:** Specifies the host name, IP of the eXtreme Scale container server.
- **Application Name:** Specifies the name of the ObjectGrid, Persistence Unit name (if set).
- **Accounting Information:** Specifies the thread ID, transaction type, transaction id, and the connection string.

Read about the DB2 Performance Expert to learn how to monitor database access.

Procedure

- To enable all eXtreme Scale client information, set the following trace strings:

  `ObjectGridClientInfo*=event=enabled`

- To enable all but user information, use one of the following settings:
  - `ObjectGridClientInfo*=event=enabled,ObjectGridClientInfoUser=event=disabled`

    or

  - `ObjectGridClientInfo=event=enabled`

Results

After you turn on the trace function, data displays in the performance monitor tool such as DB2 Performance Expert.

Example

In the following example, user `bob` is authenticated as an eXtreme Scale user. The application is accessing the `mygrid` data grid using the `DB2Hibernate` persistence unit. The container server is named `XS_Server1`. The resulting information follows:

- **User**=`bob`
- **Workstation Name**=`XS_Server1,192.168.1.101`
- **Application Name**=`mygrid,DB2Hibernate`
- **Accounting Information**=`1, DEFAULT,FE7954BD-0126-4000-E000-2298094151DB,com.ibm.db2.jcc.t4.b@71787178`

In the following example, user `bob` is authenticated using a WebSphere® Application Server token. The application is accessing the `mygrid` data grid using the `DB2OpenJPA` persistence unit name. The container server is named `XS_Server2`. The resulting information follows:

- **User**

  `=acme.principal.UserPrincipal[Bob],acme.principal.GroupPrincipal[admin]`

- **Workstation Name**=`XS_Server2,192.168.1.102`
- **Application Name**=`mygrid,DB2OpenJPA`
- **Accounting Information**=`188,DEFAULT,FE72BC63-0126-4000-E000-851C092A4E33,com.ibm.ws.rsadapter.jdbc.WSJccSQLJConnection@2b432b43`

Related concepts:
Statistics overview
Monitoring with vendor tools
Loaders
Plug-ins for communicating with databases
Related tasks:
Monitoring with the web console
Monitoring with CSV files

Enabling statistics
Monitoring with the xscmd utility
Monitoring with WebSphere Application Server PMI
Monitoring server statistics with managed beans (MBeans)

Tuning performance

You can tune settings in your environment to increase the overall performance of your WebSphere® eXtreme Scale environment.

- Tuning operating systems and network settings
  Network tuning can reduce Transmission Control Protocol (TCP) stack delay by changing connection settings and can improve throughput by changing TCP buffers.
- ORB properties
  Object Request Broker (ORB) properties modify the transport behavior of the data grid. These properties can be set with an orb.properties file, as settings in the WebSphere Application Server administrative console, or as custom properties on the ORB in the WebSphere Application Server administrative console.
- Tuning Java virtual machines
  You must take into account several specific aspects of Java™ virtual machine (JVM) tuning for WebSphere eXtreme Scale best performance. In most cases, few or no special JVM settings are required. If many objects are being stored in the data grid, adjust the heap size to an appropriate level to avoid running out of memory.
- Tuning the heartbeat interval setting for failover detection
  You can configure the amount of time between system checks for failed servers with the heartbeat interval setting. This setting applies to catalog servers only.
- Tuning garbage collection with WebSphere Real Time
  Using WebSphere eXtreme Scale with WebSphere Real Time increases consistency and predictability at a cost of performance throughput in comparison to the default garbage collection policy employed in the standard IBM® Java™ SE Runtime Environment (JRE). The cost versus benefit proposition can vary. WebSphere eXtreme Scale creates many temporary objects that are associated with each transaction. These temporary objects deal with requests, responses, log sequences, and sessions. Without WebSphere Real Time, transaction response time can go up to hundreds of milliseconds. However, using WebSphere Real Time with WebSphere eXtreme Scale can increase the efficiency of garbage collection and reduce response time to 10% of the stand-alone configuration response time.
- Tuning the dynamic cache provider
  The WebSphere eXtreme Scale dynamic cache provider supports the following configuration parameters for performance tuning.
- Tuning the cache sizing agent for accurate memory consumption estimates
  WebSphere eXtreme Scale supports sizing the memory consumption of BackingMap instances in distributed data grids. Memory consumption sizing is not supported for local data grid instances. The value that is reported by WebSphere eXtreme Scale for a given map is very close to the value that is reported by heap dump analysis. If map object is complex, the sizings might be less accurate. The CWOBJ4543 message is displayed in the log for any cache entry object that cannot be accurately sized because it is overly complex. You can get a more accurate measurement by avoiding unnecessary map complexity.
- Tuning and performance for application development
  To improve performance for your in-memory data grid or database processing space, you can investigate several considerations such using the best practices for product features such as locking, serialization, and query performance.

Tuning operating systems and network settings

Network tuning can reduce Transmission Control Protocol (TCP) stack delay by changing connection settings and can improve throughput by changing TCP buffers.

Operating systems

The tuning settings might improve WebSphere® eXtreme Scale performance. Tune according to your network and application load.

Windows:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\
Tcpip\Parameters
MaxFreeTcbs = dword:00011940
MaxHashTableSize = dword:00010000
MaxUserPort = dword:0000fffe
TcpTimedWaitDelay = dword:0000001e
```

Solaris:

```
ndd -set /dev/tcp tcp_time_wait_interval 60000
fndd -set /dev/tcp tcp_keepalive_interval 15000
ndd -set /dev/tcp tcp_fin_wait_2_flush_interval 67500
ndd -set /dev/tcp tcp_conn_req_max_q 16384
ndd -set /dev/tcp tcp_conn_req_max_q0 16384
ndd -set /dev/tcp tcp_xmit_hiwat 400000
ndd -set /dev/tcp tcp_recv_hiwat 400000
ndd -set /dev/tcp tcp_cwnd_max 2097152
```

```
ndd -set /dev/tcp tcp_ip_abort_interval 20000
ndd -set /dev/tcp tcp_rexmit_interval_initial 4000
ndd -set /dev/tcp tcp_rexmit_interval_max 10000
ndd -set /dev/tcp tcp_rexmit_interval_min 3000
ndd -set /dev/tcp tcp_max_buf 4194304
```

AIX®:

```
/usr/sbin/no -o tcp_sendspace=65536
/usr/sbin/no -o tcp_recvspace=65536
/usr/sbin/no -o udp_sendspace=65536
/usr/sbin/no -o udp_recvspace=65536
/usr/sbin/no -o somaxconn=10000
/usr/sbin/no -o tcp_nodelayack=1
/usr/sbin/no -o tcp_keepinit=40
/usr/sbin/no -o tcp_keepintvl=10
```

Linux:

```
sysctl -w net.ipv4.tcp_timestamps=0
sysctl -w net.ipv4.tcp_tw_reuse=1
sysctl -w net.ipv4.tcp_tw_recycle=1
sysctl -w net.ipv4.tcp_fin_timeout=30
sysctl -w net.ipv4.tcp_keepalive_time=1800
sysctl -w net.ipv4.tcp_rmem="4096 87380 8388608"
sysctl -w net.ipv4.tcp_wmem="4096 87380 8388608"
sysctl -w net.ipv4.tcp_max_syn_backlog=4096
```

HP-UX:

```
ndd -set /dev/tcp tcp_ip_abort_cinterval 20000
```

Jumbo frames

For Ethernet networks, enabling jumbo frames (frame size or Maximum Transmission Unit (MTU) of 9000 bytes) on all systems (hosts and switches) can provide a significant performance improvement, especially when the application uses large payload sizes. Check the operating instructions for the particular host and switches in your network for information about how to enable jumbo frames. The steps to configure jumbo frames are particular to each equipment type.
Note: Enabling jumbo frames on some hosts in the configuration and not others can cause the switch to become a bottleneck point. The switch must convert between frame sizes on different ports. Therefore, it is best to enable jumbo frames on all hosts in the configuration or none of the hosts in the configuration.

ORB properties

Object Request Broker (ORB) properties modify the transport behavior of the data grid. These properties can be set with an orb.properties file, as settings in the WebSphere® Application Server administrative console, or as custom properties on the ORB in the WebSphere Application Server administrative console.

orb.properties

The orb.properties file is in the java/jre/lib directory. When you modify the orb.properties file in a WebSphere Application Server java/jre/lib directory, the ORB properties are updated on the node agent and any other Java virtual machines (JVM) that are using the Java runtime environment (JRE). If you do not want this behavior, use custom properties or the ORB settings WebSphere Application Server administrative console.

Default WebSphere Application Server settings

WebSphere Application Server has some properties defined on the ORB by default. These settings are on the application server container services and the deployment manger. These default settings override any settings that you create in the orb.properties file. For each described property, see the **Where to specify** section to determine the location to define the suggested value.

File descriptor settings

For UNIX and Linux systems, a limit exists for the number of open files that are allowed per process. The operating system specifies the number of open files permitted. If this value is set too low, a memory allocation error occurs on AIX®, and too many files opened are logged.

In the UNIX system terminal window, set this value higher than the default system value. For large SMP machines with clones, set to unlimited.

For AIX configurations set this value to unlimited with the command: `ulimit -n unlimited`.

For Solaris configurations set this value to `16384` with the command: `ulimit -n 16384`.

To display the current value use the command: `ulimit -a`.

Baseline settings

The following settings are a good baseline but not necessarily the best settings for every environment. Understand the settings to help make a good decision on what values are appropriate in your environment.

```
com.ibm.CORBA.RequestTimeout=30
com.ibm.CORBA.ConnectTimeout=10
com.ibm.CORBA.FragmentTimeout=30
com.ibm.CORBA.LocateRequestTimeout=10
com.ibm.CORBA.ThreadPool.MinimumSize=256
com.ibm.CORBA.ThreadPool.MaximumSize=256
com.ibm.CORBA.ThreadPool.IsGrowable=false
com.ibm.CORBA.ConnectionMultiplicity=1
com.ibm.CORBA.MinOpenConnections=1024
com.ibm.CORBA.MaxOpenConnections=1024
com.ibm.CORBA.ServerSocketQueueDepth=1024
com.ibm.CORBA.FragmentSize=0
com.ibm.CORBA.iiop.NoLocalCopies=true
com.ibm.CORBA.NoLocalInterceptors=true
com.ibm.CORBA.SocketWriteTimeout=30
```

Property descriptions

Timeout Settings

The following settings relate to the amount of time that the ORB waits before giving up on request operations. Use these settings to prevent excess threads from being created in an abnormal situation.

Request timeout

Property name: com.ibm.CORBA.RequestTimeout

Valid value: Integer value for number of seconds.

Suggested value: 30

Where to specify: WebSphere Application Server administrative console

Description: Indicates how many seconds any request waits for a response before giving up. This property influences the amount of time a client takes to fail over if a network outage failure occurs. If you set this property too low, requests might time out inadvertently. Carefully consider the value of this property to prevent inadvertent timeouts.

Connect timeout

Property name: com.ibm.CORBA.ConnectTimeout

Valid value: Integer value for number of seconds.

Suggested value: 10

Where to specify: orb.properties file

Description: Indicates how many seconds a socket connection attempt waits before giving up. This property, like the request timeout, can influence the time a client takes to fail over if a network outage failure occurs. In general, set this property to a smaller value than the request timeout value because the amount of time to establish connections is relatively constant.

Fragment timeout

Property name: com.ibm.CORBA.FragmentTimeout

Valid value: Integer value for number of seconds.

Suggested value: 30

Where to specify: orb.properties file

Description: Indicates how many seconds a fragment request waits before giving up. This property is similar to the request timeout property.

Socket write timeout

Property name: com.ibm.CORBA.SocketWriteTimeout

Valid value: Integer value for number of seconds.

Suggested value: 30

Where to specify: orb.properties file

Description: Indicates how many seconds a socket write waits before giving up. This property is similar to the request timeout property.

Thread Pool Settings

These properties constrain the thread pool size to a specific number of threads. The threads are used by the ORB to spin off the server requests after they are received on the socket. Setting these property values too low results in an increased socket queue depth and possibly timeouts.

Connection multiplicity

Property name: com.ibm.CORBA.ConnectionMultiplicity

Valid value: Integer value for the number of connections between the client and server. The default value is 1. Setting a larger value sets multiplexing across multiple connections.

Suggested value: 1

Where to specify: orb.properties file

Description: Enables the ORB to use multiple connections to any server. In theory, setting this value promotes parallelism over the connections. In practice, performance does not benefit from setting the connection multiplicity. Do not set this parameter.

Open connections

Property names: com.ibm.CORBA.MinOpenConnections, com.ibm.CORBA.MaxOpenConnections

Valid value: An integer value for the number of connections.

Suggested value: 1024

Where to specify: WebSphere Application Server administrative console

Description: Specifies a minimum and maximum number of open connections. The ORB keeps a cache of connections that have been established with clients. These connections are purged when this value is passed. Purging connections might cause poor behavior in the data grid.

Is Growable

Property name: com.ibm.CORBA.ThreadPool.IsGrowable

Valid value: Boolean; set to `true` or `false`.

Suggested value: `false`

Where to specify: orb.properties file

Description: If set to `true`, the thread pool that the ORB uses for incoming requests can grow beyond what the pool supports. If the pool size is exceeded, new threads are created to handle the request but the threads are not pooled. Prevent thread pool growth by setting the value to `false`.

Server socket queue depth

Property name: com.ibm.CORBA.ServerSocketQueueDepth

Valid value: An integer value for the number of connections.

Suggested value: `1024`

Where to specify: orb.properties file

Description: Specifies the length of the queue for incoming connections from clients. The ORB queues incoming connections from clients. If the queue is full, then connections are refused. Refusing connections might cause poor behavior in the data grid.

Fragment size

Property name: com.ibm.CORBA.FragmentSize

Valid value: An integer number that specifies the number of bytes. The default is 1024.

Suggested value: `0`

Where to specify: orb.properties file

Description: Specifies the maximum packet size that the ORB uses when sending a request. If a request is larger than the fragment size limit, then that request is divided into request fragments that are each sent separately and reassembled on the server. Fragmenting requests is helpful on unreliable networks where packets might need to be resent. However, if the network is reliable, dividing the requests into fragments might cause unnecessary processing.

No local copies

Property name: com.ibm.CORBA.iiop.NoLocalCopies

Valid value: Boolean; set to `true` or `false`.

Suggested value: `true`

Where to specify: WebSphere Application Server administrative console, Pass by reference setting.

Description: Specifies whether the ORB passes by reference. The ORB uses pass by value invocation by default. Pass by value invocation causes extra garbage and serialization costs to the path when an interface is started locally. By setting this value to true, the ORB uses a pass by reference method that is more efficient than pass by value invocation.

No Local Interceptors

Property name: com.ibm.CORBA.NoLocalInterceptors

Valid value: Boolean; set to `true` or `false`.

Suggested value: `true`

Where to specify: orb.properties file

Description: Specifies whether the ORB starts request interceptors even when making local requests (intra-process). The interceptors that WebSphere eXtreme Scale uses are for security and route handling are not required if the request is handled within the process. Interceptors that go between processes are only required for Remote Procedure Call (RPC) operations. By setting the no local interceptors, you can avoid the extra processing that using local interceptors introduces.
Attention: If you are usingWebSphere eXtreme Scale security, set the com.ibm.CORBA.NoLocalInterceptors property value to `false`. The security infrastructure uses interceptors for authentication.

Related tasks:
Configuring Object Request Brokers
Configuring the Object Request Broker with stand-alone WebSphere eXtreme Scale processes
Configuring a custom Object Request Broker

Tuning Java virtual machines

You must take into account several specific aspects of Java™ virtual machine (JVM) tuning for WebSphere® eXtreme Scale best performance. In most cases, few or no special JVM settings are required. If many objects are being stored in the data grid, adjust the heap size to an appropriate level to avoid running out of memory.

IBM eXtremeMemory

By configuring eXtremeMemory, you can store objects in native memory instead of on the Java heap. Configuring eXtremeMemory enables eXtremeIO, a new transport mechanism. By moving objects off the Java heap, you can avoid garbage collection pauses, leading to more constant performance and predicable response times. For more information, see Configuring IBM eXtremeMemory.

Tested platforms

Performance testing occurred primarily on AIX® (32 way), Linux (four way), and Windows (eight way) computers. With high-end AIX computers, you can test heavily multi-threaded scenarios to identify and fix contention points.

Garbage collection

WebSphere eXtreme Scale creates temporary objects that are associated with each transaction, such as request and response, and log sequence. Because these objects affect garbage collection efficiency, tuning garbage collection is critical.

All modern JVMs use parallel garbage collection algorithms, which means that using more cores can reduce pauses in garbage collection. A physical server with eight cores has a faster garbage collection than a physical with four cores.

When the application must manage a large amount of data for each partition, then garbage collection might be a factor. A read mostly scenario performs even with large heaps (20 GB or more) if a generational collector is used. However, after the tenure heap fills, a pause proportional to the live heap size and the number of processors on the computer occurs. This pause can be large on smaller computers with large heaps.

IBM virtual machine for Java garbage collection

For the IBM® virtual machine for Java, use the **optavgpause** collector for high update rate scenarios (100% of transactions modify entries). The **gencon** collector works much better than the **optavgpause** collector for scenarios where data is updated relatively infrequently (10% of the time or less). Experiment with both collectors to see what works best in your scenario. Run with verbose garbage collection turned on to check the percentage of the time that is being spent collecting garbage. Scenarios have occurred where 80% of the time is spent in garbage collection until tuning fixed the problem.

Use the -Xgcpolicy parameter to change the garbage collection mechanism. The value of the -Xgcpolicy parameter can be set to: -Xgcpolicy:gencon or -Xgcpolicy:optavgpause, depending on which garbage collector you want to use.

- In a WebSphere Application Server configuration, set the -Xgcpolicy parameter in the administrative console. Click Servers > Application servers > server_name > Process definition > Java Virtual Machine. Add the parameter in the Generic JVM arguments field.
- In a stand-alone configuration, pass the -jvmArgs parameter to the start server script to specify the garbage collector. The -jvmArgs parameter must be the last parameter that is passed to the script.

Other garbage collection options

Attention: If you are using an Oracle JVM, adjustments to the default garbage collection and tuning policy might be necessary.
WebSphere eXtreme Scale supports WebSphere Real Time Java. With WebSphere Real Time Java, the transaction processing response for WebSphere eXtreme Scale is more consistent and predictable. As a result, the impact of garbage collection and thread scheduling is greatly minimized. The impact is reduced to the degree that the standard deviation of response time is less than 10% of regular Java.

JVM performance

WebSphere eXtreme Scale can run on different versions of Java Platform, Standard Edition. WebSphere eXtreme Scale supports Java SE Version 5 or later . For improved developer productivity and performance, use Java SE Version 5 or later , or Java SE Version 7 to take advantage of annotations and improved garbage collection. WebSphere eXtreme Scale works on 32-bit or 64-bit Java virtual machines.

WebSphere eXtreme Scale is tested with a subset of the available virtual machines, however, the supported list is not exclusive. You can run WebSphere eXtreme Scale on any vendor JVM at Edition 5 or later. However, if a problem occurs with a vendor JVM, you must contact the JVM vendor for support. If possible, use the JVM from the WebSphere run time on any platform that WebSphere Application Server supports.

In general, use the latest available version of Java Platform, Standard Edition for the best performance.

Heap size

The recommendation is 1 to 2 GB heaps with a JVM per four cores. The optimum heap size number depends on the following factors:

- Number of live objects in the heap.
- Complexity of live objects in the heap.
- Number of available cores for the JVM.

For example, an application that stores 10 K byte arrays can run a much larger heap than an application that uses complex graphs of POJOs.

Note:
When running on Solaris, you must choose between a 32-bit or a 64-bit environment. If you do not specify either version, then the JVM runs as a 32-bit environment. If you are running WebSphere eXtreme Scale on Solaris and you encounter a heap size limit, then the -d64 option should be used to force the JVM to run in 64-bit mode which will support heap sizes greater than 3.5 GB.

Thread count

The thread count depends on a few factors. A limit exists for how many threads a single shard can manage. A shard is an instance of a partition, and can be a primary or a replica. With more shards for each JVM, you have more threads with each additional shard providing more concurrent paths to the data. Each shard is as concurrent as possible although there is a limit to the concurrency.

Object Request Broker (ORB) requirements

The IBM SDK includes an IBM ORB implementation that has been tested with WebSphere Application Server and WebSphere eXtreme Scale. To ease the support process, use an IBM-provided JVM. Other JVM implementations use a different ORB. The IBM ORB is only supplied with IBM-provided Java virtual machines. WebSphere eXtreme Scale requires a working ORB to operate. You can use WebSphere eXtreme Scale with ORBs from other vendors. However, if you have a problem with a vendor ORB, you must contact the ORB vendor for support. The IBM ORB implementation is compatible with third partyJava virtual machines and can be substituted if needed.

orb.properties tuning

In the lab, the following file was used on data grids of up to 1500 JVMs. The orb.properties file is in the lib folder of the runtime environment.

```
# IBM JDK properties for ORB
org.omg.CORBA.ORBClass=com.ibm.CORBA.iiop.ORB
org.omg.CORBA.ORBSingletonClass=com.ibm.rmi.corba.ORBSingleton

# WS Interceptors
org.omg.PortableInterceptor.ORBInitializerClass.com.ibm.ws.objectgrid.corba.ObjectGridInitializer

# WS ORB & Plugins properties
com.ibm.CORBA.ForceTunnel=never
com.ibm.CORBA.RequestTimeout=10
com.ibm.CORBA.ConnectTimeout=10

# Needed when lots of JVMs connect to the catalog at the same time
com.ibm.CORBA.ServerSocketQueueDepth=2048

# Clients and the catalog server can have sockets open to all JVMs
com.ibm.CORBA.MaxOpenConnections=1016

# Thread Pool for handling incoming requests, 200 threads here
com.ibm.CORBA.ThreadPool.IsGrowable=false
com.ibm.CORBA.ThreadPool.MaximumSize=200
com.ibm.CORBA.ThreadPool.MinimumSize=200
com.ibm.CORBA.ThreadPool.InactivityTimeout=180000

# No splitting up large requests/responses in to smaller chunks
com.ibm.CORBA.FragmentSize=0
```

Related concepts:
Java SE considerations
Java EE considerations
Tuning garbage collection with WebSphere Real Time
Related reference:
startOgServer script
Related information:
Tuning the IBM virtual machine for Java

Tuning the heartbeat interval setting for failover detection

You can configure the amount of time between system checks for failed servers with the heartbeat interval setting. This setting applies to catalog servers only.

About this task

Configuring failover depends on the type of environment you are using. If you are using a stand-alone environment, you can configure failover with the command line. If you are using a WebSphere® Application Server Network Deployment environment, you must configure failover in the WebSphere Application Server Network Deployment administrative console.

Procedure

- Configure failover for stand-alone environments.
  - With the -heartbeat parameter in the **startOgServer** script when you start the catalog server.

- With the heartBeatFrequencyLevel property in the server properties file for the catalog server.

Use one of the following values:

Table 1. Valid heartbeat values

Value	Action	Description
-1	Aggressive	Specifies an aggressive heartbeat level. With this value, failures are detected more quickly, but more processor and network resources are used. This level is more sensitive to missing heartbeats when the server is busy. Failovers are typically detected within 5 seconds.
0	Typical (default)	Specifies a heartbeat level at a typical rate. With this value, failover detection occurs at a reasonable rate without overusing resources. Failovers are typically detected within 30 seconds.
1	Relaxed	Specifies a relaxed heartbeat level. With this value, a decreased heartbeat frequency increases the time to detect failures, but also decreases processor and network use. Failovers are typically detected within 180 seconds.

An aggressive heartbeat interval can be useful when the processes and network are stable. If the network or processes are not optimally configured, heartbeats might be missed, which can result in a false failure detection.

- Configure failover for WebSphere Application Server environments.
You can configure WebSphere Application Server Network Deployment Version 6.1 and later to allow WebSphere eXtreme Scale to fail over very quickly. The default failover time for hard failures is approximately 200 seconds. A hard failure is a physical computer or server crash, network cable disconnection or operating system error. Failures because of process crashes or soft failures typically fail over in less than one second. Failure detection for soft failures occurs when the network sockets from the dead process are closed automatically by the operating system for the server hosting the process.

  Core group heartbeat configuration

  WebSphere eXtreme Scale running in a WebSphere Application Server process inherits the failover characteristics from the core group settings of the application server. The following sections describe how to configure the core group heartbeat settings for different versions of WebSphere Application Server Network Deployment:

  - **Update the core group settings for WebSphere Application Server Network Deployment Version 6.1 and 7.0:**
  Specify the heartbeat interval in seconds on WebSphere Application Server versions from Version 6.0 through Version 6.1.0.12 or in milliseconds starting with Version 6.1.0.13. You must also specify the number of missed heartbeats. This value indicates how many heartbeats can be missed before a peer Java™ virtual machine (JVM) is considered as failed. The hard failure detection time is approximately the product of the heartbeat interval and the number of missed heartbeats.

    These properties are specified using custom properties on the core group using the WebSphere administrative console. See Core group custom properties for configuration details. These properties must be specified for all core groups used by the application:
    - The heartbeat interval is specified using either the IBM_CS_FD_PERIOD_SEC custom property for seconds or the IBM_CS_FD_PERIOD_MILLIS custom property for milliseconds (requires Version 6.1.0.13 or later).
    - The number of missed heartbeats is specified using the IBM_CS_FD_CONSECUTIVE_MISSED custom property.

    The default value for the IBM_CS_FD_PERIOD_SEC property is 20 and for the IBM_CS_FD_CONSECUTIVE_MISSED property is 10. If the IBM_CS_FD_PERIOD_MILLIS property is specified, then it overrides any of the set IBM_CS_FD_PERIOD_SEC custom properties. The values of these properties are positive integer values.

    Use the following settings to achieve a 1500 ms failure detection time for WebSphere Application Server Network Deployment Version 6.x servers:
    - Set `IBM_CS_FD_PERIOD_MILLIS = 750` (WebSphere Application Server Network Deployment V6.1.0.13 and later)
    - Set `IBM_CS_FD_CONSECUTIVE_MISSED = 2`
  - **Update the core group settings for WebSphere Application Server Network Deployment Version 7.0**

    WebSphere Application Server Network Deployment Version 7.0 provides two core group settings that can be adjusted to increase or decrease failover detection:

    - **Heartbeat transmission period.** The default is 30000 milliseconds.
    - **Heartbeat timeout period.** The default is 180000 milliseconds.

    For more details on how change these settings, see the WebSphere Application Server Network Deployment Information center: Discovery and failure detection settings.

    Use the following settings to achieve a 1500 ms failure detection time for WebSphere Application Server Network Deployment Version 7 servers:

    - Set the heartbeat transmission period to 750 milliseconds.
    - Set the heartbeat timeout period to 1500 milliseconds.

What to do next

When these settings are modified to provide short failover times, there are some system-tuning issues to be aware of. First, Java is not a real-time environment. It is possible for threads to be delayed if the JVM is experiencing long garbage collection times. Threads might also be delayed if the machine hosting the JVM is heavily loaded (due to the JVM itself or other processes running on the machine). If threads are delayed, heartbeats might not be sent on time. In the worst case, they might be delayed by the required failover time. If threads are delayed, false failure detections occur. The system must be tuned and sized to ensure that false failure detections do not happen in production. Adequate load testing is the best way to ensure this.

Note: The current version of eXtreme Scale supports WebSphere Real Time.
Related concepts:
High availability
Core groups

High availability catalog service
Catalog server quorums
Replication for availability
Installation topologies
Catalog service
Related reference:
Server properties file
startOgServer script

Tuning garbage collection with WebSphere Real Time

Using WebSphere® eXtreme Scale with WebSphere Real Time increases consistency and predictability at a cost of performance throughput in comparison to the default garbage collection policy employed in the standard IBM® Java™ SE Runtime Environment (JRE). The cost versus benefit proposition can vary. WebSphere eXtreme Scale creates many temporary objects that are associated with each transaction. These temporary objects deal with requests, responses, log sequences, and sessions. Without WebSphere Real Time, transaction response time can go up to hundreds of milliseconds. However, using WebSphere Real Time with WebSphere eXtreme Scale can increase the efficiency of garbage collection and reduce response time to 10% of the stand-alone configuration response time.

- WebSphere Real Time in a stand-alone environment
  You can use WebSphere Real Time with WebSphere eXtreme Scale. By enabling WebSphere Real Time, you can get more predictable garbage collection along with a stable, consistent response time and throughput of transactions in a stand-alone eXtreme Scale environment.
- WebSphere Real Time in WebSphere Application Server
  You can use WebSphere® Real Time with eXtreme Scale in a WebSphere Application Server Network Deployment environment version 7.0. By enabling WebSphere Real Time, you can get more predictable garbage collection along with a stable, consistent response time and throughput of transactions.

Related concepts:
Interoperability with other products
Monitoring with vendor tools
Installation topologies
Tuning Java virtual machines
Java SE considerations
Java EE considerations
Related tasks:
Configuring the HTTP session manager for various application servers
Configuring HTTP session manager with WebSphere Portal
Configuring the HTTP session manager with WebSphere Application Server
Configuring WebSphere eXtreme Scale with WebSphere Application Server
Related reference:
startOgServer script
Related information:
Configure WebSphere Commerce to use WebSphere eXtreme Scale for dynamic cache to improve performance and scale
WebSphere Business Process Management and Connectivity integration
Tuning the IBM virtual machine for Java

WebSphere Real Time in a stand-alone environment

You can use WebSphere® Real Time with WebSphere eXtreme Scale. By enabling WebSphere Real Time, you can get more predictable garbage collection along with a stable, consistent response time and throughput of transactions in a stand-alone eXtreme Scale environment.

Advantages of WebSphere Real Time

WebSphere eXtreme Scale creates many temporary objects that are associated with each transaction. These temporary objects deal with requests, responses, log sequences, and sessions. Without WebSphere Real Time, transaction response time can go up to hundreds of milliseconds. However, using WebSphere Real Time with WebSphere eXtreme Scale can increase the efficiency of garbage collection and reduce response time to 10% of the stand-alone configuration response time.

Enabling WebSphere Real Time

Install WebSphere Real Time and stand-alone WebSphere eXtreme Scale onto the computers on which you plan to run eXtreme Scale. Set the JAVA_HOME environment variable to point to a standard Java™ SE Runtime Environment (JRE).

Set the JAVA_HOME environment variable to point to the installed WebSphere Real Time. Then enable WebSphere Real Time as follows.

1. Edit the stand-alone installation objectgridRoot/bin/setupCmdLine.sh | .bat file by removing the comment from the following line.
   `WXS_REAL_TIME_JAVA="-Xrealtime -Xgcpolicy:metronome -Xgc:targetUtilization=80"`
2. Save the file.

Now you have enabled WebSphere Real Time. If you want to disable WebSphere Real Time, you can add the comment back to the same line.

Best practices

WebSphere Real Time allows eXtreme Scale transactions to have a more predictable response time. Results show that the deviation of an eXtreme Scale transaction's response time improves significantly with WebSphere Real Time compared to standard Java with its default garbage collector. Enabling WebSphere Real Time with eXtreme Scale is optimal if your application's stability and response time are essential.

The best practices described in this section explain how to make WebSphere eXtreme Scale more efficient through tuning and code practices depending on your expected load.

- Set right level of processor usage for your application and garbage collector.
  WebSphere Real Time provides capacity to control the processor usage so that garbage collection impact on your application is controlled and minimized. Use the `-Xgc:targetUtilization=NN` parameter to specify `NN` percentage of the processor that is used by your application in every 20 seconds. The default for WebSphere eXtreme Scale is 80%, but you can modify the script in objectgridRoot/bin/setupCmdLine.sh file to set different number such as `70`, which provides more processor capacity to the garbage collector. Deploy enough servers to maintain processor load under 80% for your applications.

- Set a larger size of heap memory.
  WebSphere Real Time uses more memory than regular Java, so plan your WebSphere eXtreme Scale with a large heap memory and set the heap size when you start catalog servers and containers with the `-jvmArgs -XmxNNNM` parameterin the **ogStartServer** command. For example, to you might use `-jvmArgs -Xmx500M` parameter to start catalog servers, and use appropriate memory size to start containers. You can set the memory size to 60-70% of your expected data size per JVM. If you do not set this value, a OutOfMemoryError error could result. Optionally, you also can use the `-jvmArgs -Xgc:noSynchronousGCOnOOM` parameterto prevent nondeterministic behavior when the JVM runs out of memory.

- Adjust threads for garbage collection.
  WebSphere eXtreme Scale creates a lot of temporary objects associated with each transaction and Remote Procedure Call (RPC) threads. Garbage collection has performance benefits if your computer has enough processor cycles. The default number of threads is `1`. You can change the number of threads with the `-Xgcthreads n` argument. The suggested value of this argument is the number of cores that are available with consideration of the number of Java virtual machines per computer.

- Adjust the performance for short-running applications with WebSphere eXtreme Scale.
  WebSphere Real Time is tuned for long running applications. Usually you need to run WebSphere eXtreme Scale continuous transactions for two hours to get reliable performance data. You can use the `-Xquickstart` parameter to make your short-running applications perform better. This parameter tells just-in-time (JIT) compiler to use lower level of optimization.

- Minimize WebSphere eXtreme Scale client queue and WebSphere eXtreme Scale client relay.
  The main advantage of using WebSphere eXtreme Scale with WebSphere Real Time is to have highly reliable transaction response time, which usually has several times of order magnitude improvements on the deviation of transaction response time. Any queued client requests and client request relay through other software impacts the response time that is beyond the control of WebSphere Real Time and WebSphere eXtreme Scale. You should change your threads and sockets parameters to maintain steady and smooth load without any significant delay and decrease your queue depth.

- Write WebSphere eXtreme Scale applications to use WebSphere Real Time threading.
  Without modifying your application, you can get highly reliable WebSphere eXtreme Scale transaction response time with several order magnitude improvements on the deviation of response time. You can further exploit threading advantage of your transactional applications from regular Java thread to RealtimeThread which provides better control on thread priority and scheduling control.

  Your application currently includes the following code.

  ```
  public class WXSCacheAppImpl extends Thread implements WXSCacheAppIF
  ```

  You can optionally replace this code with the following.

  ```
  public class WXSCacheAppImpl extends RealtimeThread implements WXSCacheAppIF
  ```

WebSphere Real Time in WebSphere Application Server

You can use WebSphere® Real Time with eXtreme Scale in a WebSphere® Application Server Network Deployment environment version 7.0. By enabling WebSphere Real Time, you can get more predictable garbage collection along with a stable, consistent response time and throughput of transactions.

Advantages

Using WebSphere eXtreme Scale with WebSphere Real Time increases consistency and predictability at a cost of performance throughput in comparison to the default garbage collection policy employed in the standard IBM® Java™ SE Runtime Environment (JRE). The cost versus benefit proposition can vary based on several criteria. The following are some of the major criteria:

- Server capabilities - Available memory, CPU speed and size, network speed and use
- Server loads – Sustained CPU load, peak CPU load
- Java™ configuration – Heap sizes, target use, garbage-collection threads
- WebSphere eXtreme Scale copy mode configuration – byte array vs. POJO storage
- Application specifics – Thread usage, response requirements and tolerance, object size, and so on.

In addition to this metronome garbage collection policy available in WebSphere Real Time, there are optional garbage collection policies available in standard IBM Java™ SE Runtime Environment (JRE). These policies, optthruput (default), gencon (default), optavgpause and subpool are specifically designed to solve differing application requirements and environments. For more information on these policies, see Tuning Java virtual machines. Depending upon application and environment requirements, resources and restrictions, prototyping one or more of these garbage collection policies can ensure that you meet your requirements and determine an optimal policy.

Capabilities with WebSphere Application Server Network Deployment

1. The following are some supported versions.
   - WebSphere Application Server Network Deployment version 7.0.0.5 and above.
   - WebSphere Real Time V2 SR2 for Linux and above. See IBM WebSphere Real Time V2 for Linux for more information.
   - WebSphere eXtreme Scale version 7.0.0.0 and above.
   - Linux 32 and 64 bit operating systems.
2. WebSphere eXtreme Scale servers cannot be collocated with a WebSphere Application Server DMgr.
3. Real Time does not support DMgr.
4. Real Time does not support WebSphere Node Agents.

Enabling WebSphere Real Time

Install WebSphere Real Time and WebSphere eXtreme Scale onto the computers on which you plan to run eXtreme Scale. Update the WebSphere Real Time Java to SR2.

You can specify the JVM settings for each server through the WebSphere Application Server version 7.0 console as follows.

Choose Servers > Server types > WebSphere application servers > <required installed server>

On the resulting page, choose "Process definition."

On the next page, click Java Virtual Machine at the top of the column on the right. (Here you can set heap sizes, garbage collection and other flags for each server.)

Set the following flags in the "Generic JVM arguments" field:

```
-Xrealtime -Xgcpolicy:metronome  -Xnocompressedrefs -Xgc:targetUtilization=80
```

Apply and save changes.

To use Real Time in WebSphere Application Server 7.0 to with eXtreme Scale servers including the JVM flags above, you must create a JAVA_HOME environment variable.

Set JAVA_HOME as follows.

1. Expand "Environment".
2. Select "WebSphere variables".
3. Ensure that "All scopes" is checked under “Show scope".
4. Select the required server from the drop-down list. (Do not select DMgr or node agent servers.)
5. If the JAVA_HOME environment variable is not listed, select "New," and specify JAVA_HOME for the variable name. In the "Value" field, enter the fully qualified path name to Real Time.
6. Apply and then save your changes.

Best practices

For a set of best practices see the best practices section in Tuning garbage collection with WebSphere Real Time. There are some important modifications to note in this list of best practices for a stand-alone WebSphere eXtreme Scale environment when deploying into a WebSphere Application Server Network Deployment environment.

You must place any additional JVM command line parameters in the same location as the garbage collection policy parameters specified in the previous section.

An acceptable initial target for sustained processor loads is 50% with short duration peek loads hitting up to 75%. Beyond this, you must add additional capacity before you see measurable degradation in predictability and consistency. You can increase performance slightly if you can tolerate longer response times. Exceeding an 80% threshold often leads to significant degradation in consistency and predictability.

Tuning the dynamic cache provider

The WebSphere® eXtreme Scale dynamic cache provider supports the following configuration parameters for performance tuning.

About this task

- **com.ibm.websphere.xs.dynacache.ignore_value_in_change_event**: When you register a change event listener with the dynamic cache provider and generate a ChangeEvent instance, there is overhead associated with deserializing the cache entry so the value can be put inside the ChangeEvent. Setting this optional parameter on the cache instance to `true` skips the deserialization of the cache entry when generating ChangeEvents. The value returned is either null for a remove operation or a byte array containing the serialized form of the object. InvalidationEvent instances carry a similar performance penalty, which you can avoid by setting com.ibm.ws.cache.CacheConfig.ignoreValueInInvalidationEvent to `true`.
- **com.ibm.websphere.xs.dynacache.enable_compression**: By default, the eXtreme Scale dynamic cache provider compresses the cache entries in memory to increase cache density, which can save a significant amount of memory for applications like servlet caching. If you know that most of your cache data is not be compressible, consider setting this value to `false`.

Tuning the cache sizing agent for accurate memory consumption estimates

WebSphere® eXtreme Scale supports sizing the memory consumption of BackingMap instances in distributed data grids. Memory consumption sizing is not supported for local data grid instances. The value that is reported by WebSphere eXtreme Scale for a given map is very close to the value that is reported by heap dump analysis. If map object is complex, the sizings might be less accurate. The CWOBJ4543 message is displayed in the log for any cache entry object that cannot be accurately sized because it is overly complex. You can get a more accurate measurement by avoiding unnecessary map complexity.

Procedure

- Enable the sizing agent.
  If you are using a Java™ 5 or higher Java virtual machine (JVM), use the sizing agent. With the sizing agent, WebSphere eXtreme Scale can obtain additional information from the JVM to improve its estimates. The agent can be loaded by adding the following argument to the JVM command line:

  ```
  -javaagent:WXS lib directory/wxssizeagent.jar
  ```

  For an embedded topology, add the argument to the command line of the WebSphere Application Server process.

  For a distributed topology, add the argument to command line of the eXtreme Scale processes (containers) and the WebSphere Application Server process.

  When loaded correctly, the following message is written to the SystemOut.log file.

  ```
  CWOBJ4541I: Enhanced BackingMap memory sizing is enabled.
  ```

- Prefer Java data types over custom data types, where possible.
  WebSphere eXtreme Scale can accurately size the memory cost of the following types:
  - java.lang.String and arrays where String is the component class (String[])
  - All primitive wrapper types (Byte, Short, Character, Boolean, Long, Double, Float, Integer) and arrays where primitive wrappers are the component type (for example, Integer[], Character[])
  - java.math.BigDecimal and java.math.BigInteger, and arrays where these two classes are the component type (BigInteger[] and BigDecimal[])
  - Temporal types (java.util.Date, java.sql.Date, java.util.Time, java.sql.Timestamp)
  - java.util.Calendar and java.util.GregorianCalendar
- Avoid object internment, when possible.
  When an object is inserted into a map, WebSphere eXtreme Scale assumes that it holds the only reference to the object and all the objects to which the object directly refers. If you insert 1000 custom Objects into a map, and each one has a reference to the same string instance, then WebSphere eXtreme Scale sizes that string instance 1000 times, overestimating the actual size of the map on the heap. However, WebSphere eXtreme Scale correctly compensates for the following common internment scenarios:
  - References to Java 5 Enums
  - References to Classes that follow the Typesafe Enum Pattern. Classes following this pattern only have only private constructors defined, have at least one private static final field of its own type, and if they implement Serializable, the class implements the readResolve() method.
  - Java 5 Primitive wrapper internment. For example, using Integer.valueOf(1) instead of new Integer(1)

  If you must use internment, use one of the preceding techniques to get more accurate estimates.

- Use custom types thoughtfully.
  When using custom types, prefer primitive data types for fields vs Object types.

  Also, prefer the Object types listed in entry 2 over your own custom implementations.

  When using custom types, keep the Object tree to one level. When inserting a custom Object into a map, WebSphere eXtreme Scale will only calculate the cost of the inserted Object, which includes any primitive fields, and all the Objects it directly references. WebSphere eXtreme Scale will not follow references further down into the Object tree. If you insert an Object into the map, and WebSphere eXtreme Scale detects references that were not followed during the sizing process, a message coded CWOBJ4543 that includes the name of the Class that could not be fully sized results. When this error occurs, treat the size statistics on the map as trend data, rather than relying on the size statistics as an accurate total.

- Use the CopyMode.COPY_TO_BYTES copy mode if possible.
  Use the CopyMode.COPY_TO_BYTES copy mode to remove any uncertainty from sizing the value Objects being inserted into the map, even when an Object tree has too many levels to be sized normally (resulting in the CWOBJ4543 message).

- Cache memory consumption sizing
  WebSphere eXtreme Scale can accurately estimate the Java heap memory usage of a given BackingMap in bytes. Use this capability to help correctly size your Java virtual machine heap settings and eviction policies. The behavior of this feature varies with the complexity of the Objects being placed in the backing map and how the map is configured. Currently, this feature is supported only for distributed data grids. Local data grid instances do not support used bytes sizing.

Related concepts:
Cache memory consumption sizing

Cache memory consumption sizing

WebSphere® eXtreme Scale can accurately estimate the Java™ heap memory usage of a given BackingMap in bytes. Use this capability to help correctly size your Java virtual machine heap settings and eviction policies. The behavior of this feature varies with the complexity of the Objects being placed in the backing map and how the map is configured. Currently, this feature is supported only for distributed data grids. Local data grid instances do not support used bytes sizing.

Heap consumption considerations

eXtreme Scale stores all of its data inside the heap space of the JVM processes that make up the data grid. For a given map, the heap space it consumes can be broken down into the following components:

- The size all the key objects currently in the map
- The size of all the value objects currently in the map
- The size of all the EvictorData objects that are in use by the Evictor plug-ins on the map
- The overhead of the underlying data structure

The number of used bytes that is reported by the sizing statistics is the sum of these four components. These values are calculated on a per entry basis on the insert, update, and remove map operations, meaning that eXtreme Scale always has a current value for the number of bytes that a given backing map is consuming.

When data grids are partitioned, each partition contains a piece of the backing map. Because the sizing statistics are calculated at the lowest level of the eXtreme Scale code, each partition of a backing map tracks its own size. You can use the eXtreme Scale Statistics APIs to track the cumulative size of the map, as well as the size of its individual partitions.

In general, use the sizing data as a measure of the trends of data over time, not as an accurate measurement of the heap space that is being used by the map. For example, if the reported size of a map doubles from 5 MB to 10 MB, then view the memory consumption of the map as having doubled. The actual measurement of 10 MB might be inaccurate for a number of reasons. If you take the reasons into account and follow the best practices, then the accuracy of the size measurements approaches that of post-processing a Java heap dump.

The main issue with accuracy is that the Java Memory Model is not restrictive enough to allow for memory measurements that are certain to be accurate. The fundamental problem is that an object can be live on the heap due to multiple references. For example, if the same 5 KB object instance is inserted into three separate maps, then any of those three maps prevent the object from being garbage collected. In this situation, any of the following measurements would be justifiable:

- The size of each map is increased by 5 KB.
- The size of the first map the Object is placed into is increased by 5 KB.
- The other two maps are not increased in size. The size of each map is increased by a fraction of the size of the object.

This ambiguity is why these measurements should be considered trend data, unless you have removed the ambiguity through design choices, best practices, and understanding of the implementation choices that can provide more accurate statistics.

eXtreme Scale assumes that a given map holds the only long-lived reference to the key and value Objects that it contains. If the same 5 KB object is put into three maps, then the size of each map is increased by 5 KB. The increase usually is not a problem, because the feature is supported only for distributed data grids. If you insert the same Object into three different maps on a remote client, each map receives its own copy of the Object. The default transactional COPY MODE settings also usually guarantee that each map has its own copy of a given Object.

Object interning

Object interning can cause a challenge with estimating heap memory usage. When you implement object interning, your application code purposely ensures that all references to a given object value actually point to the same object instance on the heap, and therefore the same location in memory. An example of this might be the following class:

```
public class ShippingOrder implements Serializeable,Cloneable{

    public static final STATE_NEW = "new";
    public static final STATE_PROCESSING = "processing";
    public static final STATE_SHIPPED = "shipped";

    private String state;
    private int orderNumber;
       private int customerNumber;

       public Object clone(){
       ShippingOrder toReturn = new ShippingOrder();
       toReturn.state = this.state;
       toReturn.orderNumber = this.orderNumber;
       toReturn.customerNumber = this.customerNumber;
       return toReturn;
    }

    private void readResolve(){
        if (this.state.equalsIgnoreCase("new")
            this.state = STATE_NEW;
        else if (this.state.equalsIgnoreCase("processing")
            this.state = STATE_PROCESSING;
        else if (this.state.equalsIgnoreCase("shipped")
            this.state = STATE_SHIPPED:
    }
}
```

Object interning causes overestimation by the sizing statistics because eXtreme Scale assumes that the objects are using different memory locations. If a million ShippingOrder objects exist, the sizing statistics display the cost of a million Strings holding the state information. In reality, only three Strings exist that are static class members. The memory cost for the static class members never should be added to any eXtreme Scale map. However, this situation cannot be detected at runtime. There are dozens of ways that similar object interning can be implemented, which is why it is so hard to detect. It is not practical for eXtreme Scale to protect against all possible implementations. However, eXtreme Scale does protect against the most commonly used types of object interning. To optimize memory usage with Object interning, implement interning only on custom objects that fall into the following two categories to enhance the accuracy of the memory consumption statistics:

- eXtreme Scale automatically adjusts for Java 5 enums and the Typesafe Enum pattern, as described at Java 2 Platform Standard Edition 5.0 Overview: Enums.
- eXtreme Scale automatically accounts for the automatic interning of primitive wrapper types, such as Integer. Automatic interning for primitive wrapper types was introduced in Java 5 through the use of static valueOf methods.

Memory consumption statistics

Use one of the following methods to access the memory consumption statistics.

Statistics API
: Use the MapStatsModule.getUsedBytes() method, which provides statistics for a single map, including the number of entries and hit rate.

  For details, see Statistics modules.

Managed Beans (MBeans)
: Use the MapUsedBytes managed MBean statistic. You can use several different types of Java Management Extensions (JMX) MBeans to administer and monitor deployments. Each MBean refers to a specific entity, such as a map, eXtreme Scale, server, replication group, or replication group member.

  For details, see Administering with Managed Beans (MBeans).

Performance monitoring infrastructure (PMI) modules
: You can monitor the performance of your applications with the PMI modules. Specifically, use the map PMI module for containers embedded in WebSphere Application Server.

  For details, see PMI modules.

WebSphere eXtreme Scale console
: With the console, you can view the memory consumption statistics. See Monitoring with the web console.

All of these methods access the same underlying measurement of the memory consumption of a given BaseMap instance. The WebSphere eXtreme Scale runtime attempts with a best effort to calculate the number of bytes of heap memory that is consumed by the key and value objects that are stored in the map, as well as the overhead of the map itself. You can see how much heap memory each map is consuming across the whole distributed data grid.

In most cases the value reported by WebSphere eXtreme Scale for a given map is very close to the value reported by heap dump analysis. WebSphere eXtreme Scale accurately sizes its own overhead, but cannot account for every possible object that might be put into a map. Following the best practices described in Tuning the cache sizing agent for accurate memory consumption estimates can enhance the accuracy of the size in bytes measurements provided by WebSphere eXtreme Scale.

Related tasks:
Tuning the cache sizing agent for accurate memory consumption estimates

Tuning and performance for application development

To improve performance for your in-memory data grid or database processing space, you can investigate several considerations such using the best practices for product features such as locking, serialization, and query performance.

- Tuning the copy mode
  WebSphere® eXtreme Scale makes a copy of the value based on the available CopyMode settings. Determine which setting works best for your deployment requirements.
- Tuning evictors
  If you use plug-in evictors, they are not active until you create them and associate them with a backing map. The following best practices increase performance for least frequently used (LFU) and least recently used (LRU) evictors.
- Tuning locking performance
  Locking strategies and transaction isolation settings affect the performance of your applications.
- Tuning serialization performance
  WebSphere eXtreme Scale uses multiple Java™ processes to hold data. These processes serialize the data: That is, they convert the data (which is in the form of Java object instances) to bytes and back to objects again as needed to move the data between client and server processes. Marshalling the data is the most expensive operation and must be addressed by the application developer when designing the schema, configuring the data grid and interacting with the data-access APIs.
- Tuning query performance
  To tune the performance of your queries, use the following techniques and tips.
- Tuning EntityManager interface performance
  The EntityManager interface separates applications from the state held in its server grid data store.

Tuning the copy mode

WebSphere® eXtreme Scale makes a copy of the value based on the available CopyMode settings. Determine which setting works best for your deployment requirements.

You can use the BackingMap API setCopyMode(CopyMode, valueInterfaceClass) method to set the copy mode to one of the following final static fields that are defined in the com.ibm.websphere.objectgrid.CopyMode class.

When an application uses the ObjectMap interface to obtain a reference to a map entry, use that reference only within the data grid transaction that obtained the reference. Using the reference in a different transaction can lead to errors. For example, if you use the pessimistic locking strategy for the BackingMap, a get or getForUpdate method call acquires an S (shared) or U (update) lock, depending on the transaction. The get method returns the reference to the value and the lock that is obtained is released when the transaction completes. The transaction must call the get or getForUpdate method to lock the map entry in a different transaction. Each transaction must obtain its own reference to the value by calling the get or getForUpdate method instead of reusing the same value reference in multiple transactions.

CopyMode for entity maps

When using a map associated with an EntityManager API entity, the map always returns the entity Tuple objects directly without making a copy unless you are using COPY_TO_BYTES copy mode. It is important that the CopyMode is updated or the Tuple is copied appropriately when making changes.

COPY_ON_READ_AND_COMMIT

The COPY_ON_READ_AND_COMMIT mode is the default mode. The valueInterfaceClass argument is ignored when this mode is used. This mode ensures that an application does not contain a reference to the value object that is in the BackingMap. Instead, the application is always working with a copy of the value that is in the BackingMap. The COPY_ON_READ_AND_COMMIT mode ensures that the application can never inadvertently corrupt the data that is cached in the BackingMap. When an application transaction calls an ObjectMap.get method for a given key, and it is the first access of the ObjectMap entry for that key, a copy of the value is returned. When the transaction is committed, any changes that are committed by the application are copied to the BackingMap to ensure that the application does not have a reference to the committed value in the BackingMap.

COPY_ON_READ

The COPY_ON_READ mode improves performance over the COPY_ON_READ_AND_COMMIT mode by eliminating the copy that occurs when a transaction is committed. The valueInterfaceClass argument is ignored when this mode is used. To preserve the integrity of the BackingMap data, the application ensures that every reference that it has for an entry is destroyed after the transaction is committed. With this mode, the ObjectMap.get method returns a copy of the value instead of a reference to the value to ensure that changes that are made by the application to the value does not affect the BackingMap value until the transaction is committed. However, when the transaction does commit, a copy of changes is not made. Instead, the reference to the copy that was returned by the ObjectMap.get method is stored in the BackingMap. The application destroys all map entry references after the transaction is committed. If application does not destroy the map entry references, the application might cause the data cached in BackingMap to become corrupted. If an application is using this mode and is having problems, switch to COPY_ON_READ_AND_COMMIT mode to see if the problem still exists. If the problem goes away, then the application is failing to destroy all of its references after the transaction has committed.

COPY_ON_WRITE

The COPY_ON_WRITE mode improves performance over the COPY_ON_READ_AND_COMMIT mode by eliminating the copy that occurs when the ObjectMap.get method is called for the first time by a transaction for a given key. 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 that is specified by the valueInterfaceClass argument. The proxy provides a copy on write implementation. When a transaction commits, the BackingMap examines the proxy to determine if any copy was made as a result of a set method being called. If a copy was made, then the reference to that copy is stored in the BackingMap. The main advantage of using this mode is that a value is never copied on a read operation or during a commit operation, when the transaction never calls a set method to change the value.

The COPY_ON_READ_AND_COMMIT and COPY_ON_READ modes both make a deep copy when a value is retrieved from the ObjectMap. If an application only updates some of the values that are retrieved in a transaction then this mode is not optimal. The COPY_ON_WRITE mode supports this behavior efficiently but requires that the application uses a simple pattern. The value objects are required to support an interface. The application must use the methods on this interface when it is interacting with the value in a session. If this is the case, then proxies are created for the values that are returned to the application. The proxy has a reference to the real value. If the application performs read operations only, the read operations always run against the real copy. If the application modifies an attribute on the object, the proxy makes a copy of the real object and then modifies the copy. The proxy then uses the copy from that point on. Using the copy allows the copy operation to be avoided completely for objects that are only read by the application. All modify operations must start with the set prefix. Enterprise JavaBeans normally are coded to use this style of method naming for methods that modify the objects attributes. This convention must be followed. Any objects that are modified are copied at the time that they are modified by the application. This read and write scenario is the most efficient scenario supported by eXtreme Scale. To configure a map to use COPY_ON_WRITE mode, use the following example. In this example, the application stores Person objects that are keyed by using the name in the Map. The person object is represented in the following code snippet.

```
class Person {
    String name;
    int age;
    public Person() {
    }
    public void setName(String n)        {
        name = n;
    }
    public String getName() {
        return name;
    }
    public void setAge(int a) {
        age = a;
    }
    public int getAge() {
        return age;
    }
}
```

The application uses the IPerson interface only when it interacts with values that are retrieved from a ObjectMap. Modify the object to use an interface as in the following example.

```
interface IPerson
{
    void setName(String n);
    String getName();
    void setAge(int a);
    int getAge();
}
// Modify Person to implement IPerson interface
class Person implements IPerson {
```

```
    ...
}
```

The application then needs to configure the BackingMap to use COPY_ON_WRITE mode, like in the following example:

```
ObjectGrid dg = ...;
BackingMap bm = dg.defineMap("PERSON");
// use COPY_ON_WRITE for this Map with
// IPerson as the valueProxyInfo Class
bm.setCopyMode(CopyMode.COPY_ON_WRITE,IPerson.class);
// The application should then use the following
// pattern when using the PERSON Map.
Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
...
sess.begin();
// the application casts the returned value to IPerson and not Person
IPerson p = (IPerson)person.get("Billy");
p.setAge(p.getAge()+1);
...
// make a new Person and add to Map
Person p1 = new Person();
p1.setName("Bobby");
p1.setAge(12);
person.insert(p1.getName(), p1);
sess.commit();
// the following snippet WON'T WORK. Will result in ClassCastException
sess.begin();
// the mistake here is that Person is used rather than
// IPerson
Person a = (Person)person.get("Bobby");
sess.commit();
```

The first section of the application retrieves a value that was named `Billy` in the map. The application casts the returned value to the IPerson object, not the Person object because the proxy that is returned implements two interfaces:

- The interface specified in the BackingMap.setCopyMode method call
- The com.ibm.websphere.objectgrid.ValueProxyInfo interface

You can cast the proxy to two types. The last part of the preceding code snippet demonstrates what is not allowed in COPY_ON_WRITE mode. The application retrieves the Bobby record and tries to cast the record to a Person object. This action fails with a class cast exception because the proxy that is returned is not a Person object. The returned proxy implements the IPerson object and ValueProxyInfo.

ValueProxyInfo interface and partial update support: This interface allows an application to retrieve either the committed read-only value referenced by the proxy or the set of attributes that have been modified during this transaction.

```
public interface ValueProxyInfo {
    List /**/ ibmGetDirtyAttributes();
    Object ibmGetRealValue();
}
```

The ibmGetRealValue method returns a read-only copy of the object. The application must not modify this value. The ibmGetDirtyAttributes method returns a list of strings that represent the attributes that were modified by the application during this transaction. The main use case for the ibmGetDirtyAttributes method is in a Java™ database connectivity (JDBC) or CMP-based loader. Only the attributes that are named in the list need be updated on either the SQL statement or object mapped to the table. This practice leads to more efficient SQL statements that are generated by the Loader. When a copy on write transaction is committed and if a loader is plugged in, the loader can cast the values of the modified objects to the ValueProxyInfo interface to obtain this information.

Handling the equals method when using COPY_ON_WRITE or proxies: For example, the following code constructs a Person object and then inserts it to an ObjectMap. Next, it retrieves the same object using the ObjectMap.get method. The value is cast to the interface. If the value is cast to the Person interface, a ClassCastException exception results because the returned value is a proxy that implements the IPerson interface and is not a Person object. The equality check fails when using the == operation because they are not the same object.

```
session.begin();
// new the Person object
Person p = new Person(...);
personMap.insert(p.getName, p);
// retrieve it again, remember to use the interface for the cast
IPerson p2 = personMap.get(p.getName());
if(p2 == p) {
    // they are the same
} else {
    // they are not
}
```

Another consideration is when you must override the equals method. The equals method must verify that the argument is an object that implements the IPerson interface and cast the argument to be an IPerson object. Because the argument might be a proxy that implements the IPerson interface, you must use the getAge and getName methods when comparing instance variables for equality. See the following example:

```
{
    if ( obj == null ) return false;
    if ( obj instanceof IPerson ) {
        IPerson x = (IPerson) obj;
        return ( age.equals( x.getAge() ) && name.equals( x.getName() ) )
    }
    return false;
}
```

ObjectQuery and HashIndex configuration requirements: When you are using COPY_ON_WRITE with ObjectQuery or a HashIndex plug-ins, you must configure the ObjectQuery schema and HashIndex plug-in to access the objects using property methods, which is the default. If you configured field access, the query engine and index attempts to access the fields in the proxy object, which always returns null or 0 because the object instance is a proxy.

NO_COPY

The NO_COPY mode allows an application to obtain performance improvements, but requires that application to never modify a value object that is obtained using an ObjectMap.get method. The valueInterfaceClass argument is ignored when this mode is used. If this mode is used, no copy of the value is ever made. If the application modifies any value object instances that are retrieved from or added to the ObjectMap, then the data in the BackingMap is corrupted. The NO_COPY mode is primarily useful for read-only maps where data is never modified by the application. If the application is using this mode and it is having problems, then switch to the COPY_ON_READ_AND_COMMIT mode to see if the problem still exists. If the problem goes away, then the application is modifying the value returned by ObjectMap.get method, either during transaction or after transaction has committed. All maps associated with EntityManager API entities automatically use this mode regardless of what is specified in the eXtreme Scale configuration.

All maps associated with EntityManager API entities automatically use this mode regardless of what is specified in the eXtreme Scale configuration.

COPY_TO_BYTES

You can store objects in a serialized format instead of POJO format. By using the COPY_TO_BYTES setting, you can reduce the memory footprint that a large graph of objects can consume. For more information, see Improving performance with byte array maps.

COPY_TO_BYTES_RAW

With COPY_TO_BYTES_RAW, you can directly access the serialized form of your data. This copy mode offers an efficient way for you to interact with serialized bytes, which allows you to bypass the deserialization process to access objects in memory.

In the ObjectGrid descriptor XML file, you can set the copy mode to COPY_TO_BYTES, and programmatically set the copy mode to COPY_TO_BYTES_RAW in the instances where you want to access the raw, serialized data. Set the copy mode to COPY_TO_BYTES _RAW in the ObjectGrid descriptor XML file only when your application uses the raw data as a part of a main application process.

Incorrect use of CopyMode

Errors occur when an application attempts to improve performance by using the COPY_ON_READ, COPY_ON_WRITE, or NO_COPY copy mode, as described above. The intermittent errors do not occur when you change the copy mode to the COPY_ON_READ_AND_COMMIT mode.

Problem

The problem might be due to corrupted data in the ObjectGrid map, which is a result of the application violating the programming contract of the copy mode that is being used. Data corruption can cause unpredictable errors to occur intermittently or in an unexplained or unexpected fashion.

Solution

The application must comply with the programming contract that is stated for the copy mode being used. For the COPY_ON_READ and COPY_ON_WRITE copy modes, the application uses a reference to a value object outside of the transaction scope from which the value reference was obtained. To use these modes, the application must delete the reference to the value object after the transaction completes, and obtain a new reference to the value object in each transaction that accesses the value object. For the NO_COPY copy mode, the application must never change the value object. In this case, either write the application so that it does not change the value object, or set the application to use a different copy mode.

- Improving performance with byte array maps
  You can store values in your maps in a byte array instead of POJO form, which reduces the memory footprint that a large graph of objects can consume.
- Tuning copy operations with the ObjectTransformer interface
  The ObjectTransformer interface uses callbacks to the application to provide custom implementations of common and expensive operations such as object serialization and deep copies on objects.

Related concepts:
CopyMode attribute
Improving performance with byte array maps
Related reference:
Server properties file
ObjectGrid descriptor XML file

Improving performance with byte array maps

You can store values in your maps in a byte array instead of POJO form, which reduces the memory footprint that a large graph of objects can consume.

Advantages

The amount of memory that is consumed increases with the number of objects in a graph of objects. By reducing a complicated graph of objects to a byte array, only one object is maintained in the heap instead of several objects. With this reduction of the number of objects in the heap, the Java™ run time has fewer objects to search for during garbage collection.

The default copy mechanism used by WebSphere® eXtreme Scale is serialization, which is expensive. For instance, if using the default copy mode of `COPY_ON_READ_AND_COMMIT`, a copy is made both at read time and at get time. Instead of making a copy at read time, with byte arrays, the value is inflated from bytes, and instead of making a copy at commit time, the value is serialized to bytes. Using byte arrays results in equivalent data consistency to the default setting with a reduction of memory used.

When using byte arrays, note that having an optimized serialization mechanism is critical to seeing a reduction of memory consumption. For more information, see Tuning serialization performance.

Configuring byte array maps

You can enable byte array maps with the ObjectGrid XML file by modifying the CopyMode attribute that is used by a map to the setting COPY_TO_BYTES, shown in the following example:

```
<backingMap name="byteMap" copyMode="COPY_TO_BYTES" />
```

Considerations

You must consider whether or not to use byte array maps in a given scenario. Although you can reduce your memory use, processor use can increase when you use byte arrays.

The following list outlines several factors that should be considered before choosing to use the byte array map function.

Object type

Comparatively, memory reduction may not be possible when using byte array maps for some object types. Consequently, several types of objects exist for which you should not use byte array maps. If you are using any of the Java primitive wrappers as values, or a POJO that does not contain references to other objects (only storing primitive fields), the number of Java Objects is already as low as possible–there is only one. Since the amount of memory used by the object is already optimized, using a byte array map for these types of objects is not recommended. Byte array maps are more suitable to object types that contain other objects or collections of objects where the total number of POJO objects is greater than one.

For example, if you have a Customer object that had a business Address and a home Address, as well as a collection of Orders, the number of objects in the heap and the number of bytes used by those objects can be reduced by using byte array maps.

Local access

When using other copy modes, applications can be optimized when copies are made if objects are Cloneable with the default ObjectTransformer or when a custom ObjectTransformer is provided with an optimized copyValue method. Compared to the other copy modes, copying on reads, writes, or commit operations will have additional cost when accessing objects locally. For example, if you have a near cache in a distributed topology or are directly accessing a local or server ObjectGrid instance, the access and commit time will increase when using byte array maps due to the cost of serialization. You will see a similar cost in a distributed topology if you use data grid agents or you access the server primary when using the ObjectGridEventGroup.ShardEvents plug-in.

Plug-in interactions

With byte array maps, objects are not inflated when communicating from a client to a server unless the server needs the POJO form. Plug-ins that interact with the map value will experience a reduction in performance due to the requirement to inflate the value.

Any plug-in that uses LogElement.getCacheEntry or LogElement.getCurrentValue will see this additional cost. If you want to get the key, you can use LogElement.getKey, which avoids the additional overhead associated with the LogElement.getCacheEntry().getKey method. The following sections discuss plug-ins in light of the usage of byte arrays.

Indexes and queries

To avoid the overhead of serialization, use byte arrays instead of Java objects. Byte arrays are much cheaper to store in memory since the JDK has less objects to search for during garbage collection, and they can be inflated only when needed. Previously, you only used byte arrays if you did not need to access the objects using queries or indexes, since the data is stored as bytes and could only be accessed through its key. However, beginning in V7.1.1, indexes and queries are more doable because serializer plug-ins lift copy-to-bytes restrictions that previously existed.

Optimistic locking

When using the optimistic locking strategy, you will have the additional cost during updates and invalidate operations. This comes from having to inflate the value on the server to get the version value to do optimistic collision checking. If you are just using optimistic locking to guarantee fetch operations and do not need optimistic collision checking, you can use the com.ibm.websphere.objectgrid.plugins.builtins.NoVersioningOptimisticCallback to disable version checking.

Loader

With a Loader, you will also have the cost in the eXtreme Scale run time from inflating and reserializing the value when it is used by the Loader. You can still use byte array maps with Loaders, but consider the cost of making changes to the value in such a scenario. For example, you can use the byte array feature in the context of a read mostly cache. In this case, the benefit of having less objects in the heap and less memory used will outweigh the cost incurred from using byte arrays on insert and update operations.

ObjectGridEventListener

When using the transactionEnd method in the ObjectGridEventListener plug-in, you will have an additional cost on the server side for remote requests when accessing a LogElement's CacheEntry or current value. If the implementation of the method does not access these fields, then you will not have the additional cost.

Related concepts:
Tuning the copy mode
CopyMode attribute

Related reference:
ObjectGrid descriptor XML file

Tuning copy operations with the ObjectTransformer interface

The ObjectTransformer interface uses callbacks to the application to provide custom implementations of common and expensive operations such as object serialization and deep copies on objects.

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.

Overview

Copies of values are always made except when the NO_COPY mode is used. The default copying mechanism that is employed in eXtreme Scale is serialization, which is known as an expensive operation. The ObjectTransformer interface is used in this situation. The ObjectTransformer interface uses callbacks to the application to provide a custom implementation of common and expensive operations, such as object serialization and deep copies on objects.

An application can provide an implementation of the ObjectTransformer interface to a map, and eXtreme Scale then delegates to the methods on this object and relies on the application to provide an optimized version of each method in the interface. The ObjectTransformer interface follows:

```
public interface ObjectTransformer {
    void serializeKey(Object key, ObjectOutputStream stream) throws IOException;
    void serializeValue(Object value, ObjectOutputStream stream) throws IOException;
    Object inflateKey(ObjectInputStream stream) throws IOException, ClassNotFoundException;
    Object inflateValue(ObjectInputStream stream) throws IOException, ClassNotFoundException;
    Object copyValue(Object value);
    Object copyKey(Object key);
}
```

You can associate an ObjectTransformer interface with a BackingMap by using the following example code:

```
ObjectGrid g = ...;
BackingMap bm = g.defineMap("PERSON");
MyObjectTransformer ot = new MyObjectTransformer();
bm.setObjectTransformer(ot);
```

Tune deep copy operations

After an application receives an object from an ObjectMap, eXtreme Scale performs a deep copy on the object value to ensure that the copy in the BaseMap map maintains data integrity. The application can then modify the object value safely. When the transaction commits, the copy of the object value in the BaseMap map is updated to the new modified value and the application stops using the value from that point on. You could have copied the object again at the commit phase to make a private copy. However, in this case the performance cost of this action was traded off against requiring the application programmer not to use the value after the transaction commits. The default ObjectTransformer attempts to use either a clone or a serialize and inflate pair to generate a copy. The serialize and inflate pair is the worst case performance scenario. If profiling reveals that serialize and inflate is a problem for your application, write an appropriate clone method to create a deep copy. If you cannot alter the class, then create a custom ObjectTransformer plug-in and implement more efficient copyValue and copyKey methods.
Related concepts:
Tuning serialization
ObjectTransformer plug-in
Serialization using the DataSerializer plug-ins

Tuning evictors

If you use plug-in evictors, they are not active until you create them and associate them with a backing map. The following best practices increase performance for least frequently used (LFU) and least recently used (LRU) evictors.

Least frequently used (LFU) evictor

The concept of a LFU evictor is to remove entries from the map that are used infrequently. The entries of the map are spread over a set amount of binary heaps. As the usage of a particular cache entry grows, it becomes ordered higher in the heap. When the evictor attempts a set of evictions it removes only the cache entries that are located lower than a specific point on the binary heap. As a result, the least frequently used entries are evicted.

Least recently used (LRU) evictor

The LRU Evictor follows the same concepts of the LFU Evictor with a few differences. The main difference is that the LRU uses a first in, first out queue (FIFO) instead of a set of binary heaps. Every time a cache entry is accessed, it moves to the head of the queue. Consequently, the front of the queue contains the most recently used map entries and the end becomes the least recently used map entries. For example, the A cache entry is used 50 times, and the B cache entry is used only once right after the A cache entry. In this situation, the B cache entry is at the front of the queue because it was used most recently, and the A cache entry is at the end of the queue. The LRU evictor evicts the cache entries that are at the tail of the queue, which are the least recently used map entries.

LFU and LRU properties and best practices to improve performance

Number of heaps

When using the LFU evictor, all of the cache entries for a particular map are ordered over the number of heaps that you specify, improving performance drastically and preventing all of the evictions from synchronizing on one binary heap that contains all of the ordering for the map. More heaps also speeds up the time that is required for reordering the heaps because each heap has fewer entries. Set the number of heaps to 10% of the number of entries in your BaseMap.

Number of queues

When using the LRU evictor, all of the cache entries for a particular map are ordered over the number of LRU queues that you specify, improving performance drastically and preventing all of the evictions from synchronizing on one queue that contains all of the ordering for the map. Set the number of queues to 10% of the number of entries in your BaseMap.

MaxSize property

When an LFU or LRU evictor begins evicting entries, it uses the MaxSize evictor property to determine how many binary heaps or LRU queue elements to evict. For example, assume that you set the number of heaps or queues to have about 10 map entries in each map queue. If your MaxSize property is set to 7, the evictor evicts 3 entries from each heap or queue object to bring the size of each heap or queue back down to 7. The evictor only evicts map entries from a heap or queue when that heap or queue has more than the MaxSize property value of elements in it. Set the MaxSize to 70% of your heap or queue size. For this example, the value is set to 7. You can get an approximate size of each heap or queue by dividing the number of BaseMap entries by the number of heaps or queues that are used.

SleepTime property

An evictor does not constantly remove entries from your map. Instead it is idle for a set amount of time, only checking the map every n number of seconds, where n refers to the SleepTime property. This property also positively affects performance: running an eviction sweep too often lowers performance because of the resources that are needed for processing them. However, not using the evictor often can result in a map that has entries that are not needed. A map full of entries that are not needed can negatively affect both the memory requirements and processing resources that are required for your map. Setting the eviction sweep interval to fifteen seconds is a good practice for most maps. If the map is written to frequently and is used at a high transaction rate, consider setting the value to a lower time. If the map is accessed infrequently, you can set the time to a higher value.

Example

The following example defines a map, creates a new LFU evictor, sets the evictor properties, and sets the map to use the evictor:

```
//Use ObjectGridManager to create/get the ObjectGrid. Refer to
// the ObjectGridManger section
ObjectGrid objGrid = ObjectGridManager.create.....
BackingMap bMap = objGrid.defineMap("SomeMap");

//Set properties assuming 50,000 map entries
LFUEvictor someEvictor = new LFUEvictor();
someEvictor.setNumberOfHeaps(5000);
someEvictor.setMaxSize(7);
someEvictor.setSleepTime(15);
bMap.setEvictor(someEvictor);
```

Using the LRU evictor is very similar to using an LFU evictor. An example follows:

```
ObjectGrid objGrid = new ObjectGrid;
BackingMap bMap = objGrid.defineMap("SomeMap");

//Set properties assuming 50,000 map entries
LRUEvictor someEvictor = new LRUEvictor();
someEvictor.setNumberOfLRUQueues(5000);
someEvictor.setMaxSize(7);
someEvictor.setSleepTime(15);
bMap.setEvictor(someEvictor);
```

Notice that only two lines are different from the LFUEvictor example.

Related concepts:
Evictors
Plug-ins for evicting cache objects
Custom evictors
Related tasks:
Enabling evictors programmatically
Configuring evictors with XML files
Related reference:
ObjectGrid descriptor XML file

Tuning locking performance

Locking strategies and transaction isolation settings affect the performance of your applications.

Pessimistic locking strategy

Use the pessimistic locking strategy for read and write map operations where keys often collide. The pessimistic locking strategy has the greatest impact on performance.

Read committed and read uncommitted transaction isolation

When you are using pessimistic locking strategy, set the transaction isolation level with the Session.setTransactionIsolation method. For read committed or read uncommitted isolation, use the Session.TRANSACTION_READ_COMMITTED or Session.TRANSACTION_READ_UNCOMMITTED arguments depending on the isolation. To reset the transaction isolation level to the default pessimistic locking behavior, use the Session.setTransactionIsolation method with the Session.REPEATABLE_READ argument.

Read committed isolation reduces the duration of shared locks, which can improve concurrency and reduce the chance for deadlocks. This isolation level should be used when a transaction does not need assurances that read values remain unchanged for the duration of the transaction.

Use an uncommitted read when the transaction does not need to see the committed data.

Optimistic locking strategy

Optimistic locking is the default configuration. This strategy improves both performance and scalability compared to the pessimistic strategy. Use this strategy when your applications can tolerate some optimistic update failures, while still performing better than the pessimistic strategy. This strategy is excellent for read operations and infrequent update applications.

OptimisticCallback plug-in

The optimistic locking strategy makes a copy of the cache entries and compares them as needed. This operation can be expensive because copying the entry might involve cloning or serialization. To implement the fastest possible performance, implement the custom plug-in for non-entity maps.

Use version fields for entities

When you are using optimistic locking with entities, use the @Version annotation or the equivalent attribute in the Entity metadata descriptor file. The version annotation gives the ObjectGrid a very efficient way of tracking the version of an object. If the entity does not have a version field and optimistic locking is used for the entity, then the entire entity must be copied and compared.

None locking strategy

Use the none locking strategy for applications that are read only. The none locking strategy does not obtain any locks or use a lock manager. Therefore, this strategy offers the most concurrency, performance and scalability.

Tuning serialization performance

WebSphere® eXtreme Scale uses multiple Java™ processes to hold data. These processes serialize the data: That is, they convert the data (which is in the form of Java object instances) to bytes and back to objects again as needed to move the data between client and server processes. Marshalling the data is the most expensive operation and must be addressed by the application developer when designing the schema, configuring the data grid and interacting with the data-access APIs.

The default Java serialization and copy routines are relatively slow and can consume 60 to 70 percent of the processor in a typical setup. The following sections are choices for improving the performance of the serialization.


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.

Write an ObjectTransformer for each BackingMap

An ObjectTransformer can be associated with a BackingMap. Your application can have a class that implements the ObjectTransformer interface and provides implementations for the following operations:

- Copying values
- Serializing and inflating keys to and from streams
- Serializing and inflating values to and from streams

The application does not need to copy keys because keys are considered immutable.

Note: The ObjectTransformer is only invoked when the ObjectGrid knows about the data that is being transformed. For example, when DataGrid API agents are used, the agents themselves as well as the agent instance data or data returned from the agent must be optimized using custom serialization techniques. The ObjectTransformer is not invoked for DataGrid API agents.

Using entities

When using the EntityManager API with entities, the ObjectGrid does not store the entity objects directly into the BackingMaps. 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 entity ObjectTransformer is invoked.

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.

Byte arrays

When using the ObjectMap or DataGrid APIs, the key and value objects are serialized whenever the client interacts with the data grid and when the objects are replicated. To avoid the overhead of serialization, use byte arrays instead of Java objects. Byte arrays are much cheaper to store in memory since the JDK has less objects to search for during garbage collection and they can be inflated only when needed. Previously, you only used byte arrays if you did not need to access the objects using queries or indexes, since the data is stored as bytes and could only be accessed through its key. However, beginning in V7.1.1, indexes and queries are more doable because serializer plug-ins lift copy-to-bytes restrictions that previously existed.

WebSphere eXtreme Scale can automatically store data as byte arrays using the CopyMode.COPY_TO_BYTES map configuration option, or it can be handled manually by the client. This option will store the data efficiently in memory and can also automatically inflate the objects within the byte array for use by queries and indexes on demand.

A MapSerializerPlugin plug-in can be associated with a BackingMap plug-in when you use the COPY_TO_BYTES or COPY_TO_BYTES_RAW copy modes. This association allows data to be stored in serialized form in memory, rather than the native Java object form. Storing serialized data conserves memory and improves replication and performance on the client and server. You can use a DataSerializer plug-in to develop high-performance serialization streams that can be compressed, encrypted, evolved, and queried.

- Tuning serialization
  The DataSerializer plug-ins expose metadata that tells WebSphere eXtreme Scale which attributes it can and cannot directly use during serialization, the path to the data that will be serialized, and the type of data that is stored in memory. You can optimize object serialization and inflation performance so that you can efficiently interact with the byte array.

Related concepts:
ObjectTransformer plug-in
Tuning serialization
Using a loader with entity maps and tuples
Serialization using the DataSerializer plug-ins

Tuning serialization

The DataSerializer plug-ins expose metadata that tells WebSphere® eXtreme Scale which attributes it can and cannot directly use during serialization, the path to the data that will be serialized, and the type of data that is stored in memory. You can optimize object serialization and inflation performance so that you can efficiently interact with the byte array.

Overview

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.

Copies of values are always made except when the NO_COPY mode is used. The default copying mechanism that is employed in eXtreme Scale is serialization, which is known as an expensive operation. The ObjectTransformer interface is used in this situation. The ObjectTransformer interface uses callbacks to the application to provide a custom implementation of common and expensive operations, such as object serialization and deep copies on objects. However, for improved performance in most cases, you can use the DataSerializer plug-ins to serialize objects. You must use either the COPY_TO_BYTES or COPY_TO_BYTES_RAW copy modes to use the DataSerializer plug-ins. For more information, see Serialization using the DataSerializer plug-ins.

An application can provide an implementation of the ObjectTransformer interface to a map, and eXtreme Scale then delegates to the methods on this object and relies on the application to provide an optimized version of each method in the interface. The ObjectTransformer interface follows:

```
public interface ObjectTransformer {
    void serializeKey(Object key, ObjectOutputStream stream) throws IOException;
    void serializeValue(Object value, ObjectOutputStream stream) throws IOException;
    Object inflateKey(ObjectInputStream stream) throws IOException, ClassNotFoundException;
    Object inflateValue(ObjectInputStream stream) throws IOException, ClassNotFoundException;
    Object copyValue(Object value);
    Object copyKey(Object key);
}
```

You can associate an ObjectTransformer interface with a BackingMap by using the following example code:

```
ObjectGrid g = ...;
BackingMap bm = g.defineMap("PERSON");
MyObjectTransformer ot = new MyObjectTransformer();
bm.setObjectTransformer(ot);
```

Tune object serialization and inflation

Object serialization is typically the most important performance consideration with eXtreme Scale, which uses the default serializable mechanism if an ObjectTransformer plug-in is not supplied by the application. An application can provide implementations of either the Serializable readObject and writeObject, or it can have the objects implement the Externalizable interface, which is approximately ten times faster. If the objects in the map cannot be modified, then an application can associate an ObjectTransformer interface with the ObjectMap. The serialize and inflate methods are provided to allow the application to provide custom code to optimize these operations, given their large performance impact on the system. The serialize method serializes the object to the provided stream. The inflate method provides the input stream and expects the application to create the object, inflate it using data in the stream and return the object. Implementations of the serialize and inflate methods must mirror each other.

The DataSerializer plug-ins replace the ObjectTransformer plug-ins, which are deprecated. To serialize your data in the most efficient way, use the DataSerializer plug-ins to improve performance in most cases. For example, if you intend to use functions, such as query and indexing, then you can immediately take advantage of the performance improvement that the DataSerializer plug-ins yield without making configuration or programmatic changes to your application code.

Related concepts:
Tuning serialization performance
ObjectTransformer plug-in
Using a loader with entity maps and tuples
Serialization using the DataSerializer plug-ins
Tuning copy operations with the ObjectTransformer interface

Tuning query performance

To tune the performance of your queries, use the following techniques and tips.

Using parameters

When a query runs, the query string must be parsed and a plan developed to run the query, both of which can be costly.WebSphere® eXtreme Scale caches query plans by the query string. Since the cache is a finite size, it is important to reuse query strings whenever possible. Using named or positional parameters also helps performance by fostering query plan reuse.

```
Positional Parameter Example Query q = em.createQuery("select c from Customer c where c.surname=?1"); q.setParameter(1,
"Claus");
```

Using indexes

Proper indexing on a map might have a significant impact on query performance, even though indexing has some overhead on overall map performance. Without indexing on object attributes involved in queries, the query engine performs a table scan for each attribute. The table scan is the most expensive operation during a query run. Indexing on object attributes that are involved in queries allow the query engine to avoid an unnecessary table scan, improving the overall query performance. If the application is designed to use query intensively on a read-most map, configure indexes for object attributes that are involved in the query. If the map is mostly updated, then you must balance between query performance improvement and indexing overhead on the map.

When plain old Java™ objects (POJO) are stored in a map, proper indexing can avoid a Java reflection. In the following example, query replaces the WHERE clause with range index search, if the budget field has an index built over it. Otherwise, query scans the entire map and evaluates the WHERE clause by first getting the budget using Java reflection and then comparing the budget with the value 50000:

```
SELECT d FROM DeptBean d WHERE d.budget=50000
```

See Query plan for details on how to best tune individual queries and how different syntax, object models and indexes can affect query performance.

Using pagination

In client-server environments, the query engine transports the entire result map to the client. The data that is returned should be divided into reasonable chunks. The EntityManager Query and ObjectMap ObjectQuery interfaces both support the setFirstResult and setMaxResults methods that allow the query to return a subset of the results.

Return primitive values instead of entities

With the EntityManager Query API, entities are returned as query parameters. The query engine currently returns the keys for these entities to the client. When the client iterates over these entities using the Iterator from the getResultIterator method, each entity is automatically inflated and managed as if it were created with the find method on the EntityManager interface. The entire entity graph is built from the entity ObjectMap on the client. The entity value attributes and any related entities are eagerly resolved.

To avoid building the costly graph, modify the query to return the individual attributes with path navigation.

For example:

```
// Returns an entity
SELECT p FROM Person p
// Returns attributes SELECT p.name, p.address.street, p.address.city, p.gender FROM Person p
```

- Query plan
  All eXtreme Scale queries have a query plan. The plan describes how the query engine interacts with ObjectMaps and indexes. Display the query plan to determine if the query string or indexes are being used appropriately. The query plan can also be used to explore the differences that subtle changes in a query string make in the way eXtreme Scale runs a query.
- Query optimization using indexes
  Defining and using indexes properly can significantly improve query performance.

Related concepts:
Indexing
Related tasks:
Configuring the HashIndex plug-in
Accessing data with indexes (Index API)
Related reference:
HashIndex plug-in attributes

Query plan

All eXtreme Scale queries have a query plan. The plan describes how the query engine interacts with ObjectMaps and indexes. Display the query plan to determine if the query string or indexes are being used appropriately. The query plan can also be used to explore the differences that subtle changes in a query string make in the way eXtreme Scale runs a query.

The query plan can be viewed one of two ways:

- EntityManager Query or ObjectQuery getPlan API methods
- ObjectGrid diagnostic trace

getPlan method

The getPlan method on the ObjectQuery and Query interfaces return a String that describes the query plan. This string can be displayed to standard output or a log to display a query plan.
Note: In a distributed environment, the getPlan method does not run against the server and does not reflect any defined indexes. To view the plan, use an agent to view the plan on the server.

Query plan trace

The query plan can be displayed using ObjectGrid trace. To enable query plan trace, use the following trace specification:

```
QueryEnginePlan=debug=enabled
```

See Collecting trace for details on how to enable trace and locate the trace log files.

Query plan examples

Query plan uses the word for to indicate that the query is iterating through an ObjectMap collection or through a derived collection such as: q2.getEmps(), q2.dept, or a temporary collection returned by an inner loop. If the collection is from an ObjectMap, the query plan shows whether a sequential scan (denoted by `INDEX SCAN`), unique or non-unique index is used. Query plan uses a filter string to list the condition expressions applied to a collection.

A Cartesian product is not commonly used in object query. The following query scans the entire EmpBean map in the outer loop and scans the entire DeptBean map in the inner loop:

```
SELECT e, d FROM EmpBean e, DeptBean d

Plan trace:

for q2 in EmpBean ObjectMap using INDEX SCAN
     for q3 in DeptBean ObjectMap using INDEX SCAN
   returning new Tuple( q2,  q3  )
```

The following query retrieves all employee names from a particular department by sequentially scanning the EmpBean map to get an employee object. From the employee object, the query navigates to its department object and applies the d.no=1 filter. In this example, each employee has only one department object reference, so the inner loop runs one time:

```
SELECT e.name FROM EmpBean e JOIN e.dept d WHERE d.no=1

Plan trace:

for q2 in EmpBean ObjectMap using INDEX SCAN
     for q3 in  q2.dept
      filter ( q3.getNo() = 1 )
   returning new Tuple( q2.name  )
```

The following query is equivalent to the previous query. However, the following query performs better because it first narrows the result down to one department object by using the unique index that is defined over the DeptBean primary key field number. From the department object, the query navigates to its employee objects to get their names:

```
SELECT e.name FROM DeptBean d JOIN d.emps e WHERE d.no=1
```

```
Plan trace:

for q2 in DeptBean ObjectMap using UNIQUE INDEX key=(1)
     for q3 in  q2.getEmps()
   returning new Tuple( q3.name  )
```

The following query finds all the employees that work for development or sales. The query scans the entire EmpBean map and performs additional filtering by evaluating the expressions: d.name = 'Sales' or d.name='Dev'

```
SELECT e FROM EmpBean e, in (e.dept) d WHERE d.name = 'Sales'
        or d.name='Dev'

Plan trace:

for q2 in EmpBean ObjectMap using INDEX SCAN
     for q3 in  q2.dept
      filter (( q3.getName() = Sales ) OR ( q3.getName() = Dev ) )
   returning new Tuple( q2  )
```

The following query is equivalent to the previous query, but this query runs a different query plan and uses the range index built over the field name. In general, this query performs better because the index over the name field is used for narrowing down the department objects, which run quickly if only a few departments are development or sales.

```
SELECT e FROM DeptBean d, in(d.emps) e WHERE d.name='Dev' or d.name='Sales'

Plan trace:

IteratorUnionIndex of

     for q2 in DeptBean ObjectMap using INDEX on name = (Dev)
       for q3 in  q2.getEmps()


     for q2 in DeptBean ObjectMap using INDEX on name = (Sales)
       for q3 in  q2.getEmps()
```

The following query finds departments that do not have any employees:

```
SELECT d FROM DeptBean d WHERE NOT EXISTS(select e from d.emps e)

Plan trace:

for q2 in DeptBean ObjectMap using INDEX SCAN
    filter ( NOT  EXISTS (    correlated collection defined as

       for q3 in  q2.getEmps()
       returning new Tuple( q3      )

   returning new Tuple( q2  )
```

The following query is equivalent to the previous query but uses the SIZE scalar function. This query has similar performance but is easier to write.

```
SELECT d FROM DeptBean d WHERE SIZE(d.emps)=0
for q2 in DeptBean ObjectMap using INDEX SCAN
    filter (SIZE( q2.getEmps()) = 0 )
   returning new Tuple( q2  )
```

The following example is another way of writing the same query as the previous query with similar performance, but this query is easier to write as well:

```
SELECT d FROM DeptBean d WHERE d.emps is EMPTY

Plan trace:

for q2 in DeptBean ObjectMap using INDEX SCAN
    filter ( q2.getEmps() IS EMPTY  )
   returning new Tuple( q2  )
```

The following query finds any employees with a home address matching at least one of the addresses of the employee whose name equals the value of the parameter. The inner loop has no dependency on the outer loop. The query runs the inner loop one time.

```
SELECT e FROM EmpBean e WHERE e.home =  any (SELECT e1.home FROM EmpBean e1
        WHERE e1.name=?1)
for q2 in EmpBean ObjectMap using INDEX SCAN
    filter ( q2.home =ANY     temp collection defined as

       for q3 in EmpBean ObjectMap using INDEX on name = ( ?1)
       returning new Tuple( q3.home      )
 )
   returning new Tuple( q2  )
```

The following query is equivalent to the previous query, but has a correlated subquery; also, the inner loop runs repeatedly.

```
SELECT e FROM EmpBean e WHERE EXISTS(SELECT e1 FROM EmpBean e1 WHERE
        e.home=e1.home and e1.name=?1)

Plan trace:

for q2 in EmpBean ObjectMap using INDEX SCAN
    filter ( EXISTS (    correlated collection defined as
```

```
      for q3 in EmpBean ObjectMap using INDEX on name = (?1)
       filter ( q2.home =  q3.home )
      returning new Tuple( q3      )

   returning new Tuple( q2  )
```

Query optimization using indexes

Defining and using indexes properly can significantly improve query performance.

WebSphere® eXtreme Scale queries can use built-in HashIndex plug-ins to improve performance of queries. Indexes can be defined on entity or object attributes. The query engine will automatically use the defined indexes if its WHERE clause uses one of the following strings:

- A comparison expression with the following operators: =, <, >, <= or >= (any comparison expressions except not equals <>)
- A BETWEEN expression
- Operands of the expressions are constants or simple terms

Requirements

Indexes have the following requirements when used by Query:

- All indexes must use the built-in HashIndex plug-in.
- All indexes must be statically defined. Dynamic indexes are not supported.
- The @Index annotation may be used to automatically create static HashIndex plug-ins.
- All single-attribute indexes must have the RangeIndex property set to true.
- All composite indexes must have the RangeIndex property set to false.
- All association (relationship) indexes must have the RangeIndex property set to false.

For information about configuring the HashIndex, refer to Plug-ins for indexing data.

For information regarding indexing, see Indexing.

For a more efficient way to search for cached objects, see Using a composite index

Using hints to choose an index

An index can be manually selected using the setHint method on the Query and ObjectQuery interfaces with the HINT_USEINDEX constant. This can be helpful when optimizing a query to use the best performing index.

Query examples that use attribute indexes

The following examples use simple terms: e.empid, e.name, e.salary, d.name, d.budget and e.isManager. The examples assume that indexes are defined over the name, salary and budget fields of an entity or value object. The empid field is a primary key and isManager has no index defined.

The following query uses both indexes over the fields of name and salary. It returns all employees with names that equal the value of the first parameter or a salary equal to the value of the second parameter:

```
SELECT e FROM EmpBean e where e.name=?1 or e.salary=?2
```

The following query uses both indexes over the fields of name and budget. The query returns all departments named 'DEV' with a budget that is greater than 2000.

```
SELECT d FROM DeptBean dwhere d.name='DEV' and d.budget>2000
```

The following query returns all employees with a salary greater than 3000 and with an isManager flag value that equals the value of the parameter. The query uses the index that is defined over the salary field and performs additional filtering by evaluating the comparison expression: `e.isManager=?1`.

```
SELECT e FROM EmpBean e where e.salary>3000 and e.isManager=?1
```

The following query finds all employees who earn more than the first parameter, or any employee that is a manager. Although the salary field has an index defined, query scans the built-in index that is built over the primary keys of the EmpBean field and evaluates the expression: `e.salary>?1` or `e.isManager=TRUE`.

```
SELECT e FROM EmpBean e WHERE e.salary>?1 or e.isManager=TRUE
```

The following query returns employees with a name that contains the letter a. Although the name field has an index defined, query does not use the index because the name field is used in the LIKE expression.

```
SELECT e FROM EmpBean e WHERE e.name LIKE '%a%'
```

The following query finds all employees with a name that is not "Smith". Although the name field has an index defined, query does not use the index because the query uses the not equals (<>) comparison operator.

```
SELECT e FROM EmpBean e where e.name<>'Smith'
```

The following query finds all departments with a budget less than the value of the parameter, and with an employee salary greater than 3000. The query uses an index for the salary, but it does not use an index for the budget because dept.budget is not a simple term. The dept objects are derived from collection e. You do not need to use the budget index to look for dept objects.

```
SELECT dept from EmpBean e, in (e.dept) dept where e.salary>3000 and dept.budget<?
```

The following query finds all employees with a salary greater than the salary of the employees that have the empid of 1, 2, and 3. The index salary is not used because the comparison involves a subquery. The empid is a primary key, however, and is used for a unique index search because all the primary keys have a built-in index defined.

```
SELECT e FROM EmpBean e WHERE e.salary > ALL (SELECT e1.salary FROM EmpBean e1 WHERE e1.empid=1 or e1.empid =2 or
e1.empid=99)
```

To check if the index is being used by the query, you can view the Query plan. Here is an example query plan for the previous query:

```
for q2 in EmpBean ObjectMap using INDEX SCAN
    filter ( q2.salary >ALL     temp collection defined as
       IteratorUnionIndex of

         for q3 in EmpBean ObjectMap using UNIQUE INDEX key=(1)
        )

         for q3 in EmpBean ObjectMap using UNIQUE INDEX key=(2)
        )

         for q3 in EmpBean ObjectMap using UNIQUE INDEX key=(99)
        )
       returning new Tuple( q3.salary )
   returning new Tuple( q2  )

for q2 in EmpBean ObjectMap using RANGE INDEX on salary with range(3000,)
     for q3 in  q2.dept
      filter ( q3.budget <  ?1 )
   returning new Tuple( q3  )
```

Indexing attributes

Indexes can be defined over any single attribute type with the constraints previously defined.

Defining entity indexes using @Index

To define an index on an entity, simply define an annotation:

```
Entities using annotations

        @Entity
        public class Employee {
        @Id int empid;
        @Index String name
        @Index double salary
        @ManyToOne Department dept;
}
@Entity
        public class Department {
        @Id int deptid;
        @Index String name;
        @Index double budget;
        boolean isManager;
        @OneToMany Collection<Employee> employees;
        }
```

With XML

Indexes can also be defined using XML:

```
        Entities without annotations

        public class Employee {
        int empid;
        String name
        double salary
        Department dept;
        }

        public class Department {
        int deptid;
        String name;
        double budget;
        boolean isManager;
        Collection employees;
        }

ObjectGrid XML with attribute indexes

<?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="DepartmentGrid" entityMetadataXMLFile="entity.xml>
        <backingMap name="Employee" pluginCollectionRef="Emp"/>
        <backingMap name="Department" pluginCollectionRef="Dept"/>
        </objectGrid>
```

```
        </objectGrids>
        <backingMapPluginCollections>
        <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxsqryoptmiz_Emp">
        <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxsqryoptmiz_MapIndexPlugin"
className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
        <property name="Name" type="java.lang.String" value="Employee.name"/>
        <property name="AttributeName" type="java.lang.String" value="name"/>
        <property name="RangeIndex" type="boolean" value="true"
        description="Ranges are must be set to true for attributes." />
        </bean>
        <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxsqryoptmiz_MapIndexPlugin"
className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
        <property name="Name" type="java.lang.String" value="Employee.salary"/>
        <property name="AttributeName" type="java.lang.String" value="salary"/>
        <property name="RangeIndex" type="boolean" value="true"
        description="Ranges are must be set to true for attributes." />
        </bean>
        </backingMapPluginCollection>
        <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxsqryoptmiz_Dept">
        <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxsqryoptmiz_MapIndexPlugin"
className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
        <property name="Name" type="java.lang.String" value="Department.name"/>
        <property name="AttributeName" type="java.lang.String" value="name"/>
        <property name="RangeIndex" type="boolean" value="true"
        description="Ranges are must be set to true for attributes." />
        </bean>
        <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxsqryoptmiz_MapIndexPlugin"
className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
        <property name="Name" type="java.lang.String" value="Department.budget"/>
        <property name="AttributeName" type="java.lang.String" value="budget"/>
        <property name="RangeIndex" type="boolean" value="true"
        description="Ranges are must be set to true for attributes." />
        </bean>
        </backingMapPluginCollection>
        </backingMapPluginCollections>
        </objectGridConfig>
```

```
Entity XML

<?xml version="1.0" encoding="UTF-8"?>
        <entity-mappings xmlns="http://ibm.com/ws/projector/config/emd"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://ibm.com/ws/projector/config/emd ./emd.xsd">

        <description>Department entities</description>
        <entity class-name="acme.Employee" name="Employee" access="FIELD">
        <attributes>
        <id name="empid" />
        <basic name="name" />
        <basic name="salary" />
        <many-to-one name="department"
        target-entity="acme.Department"
        fetch="EAGER">
        <cascade><cascade-persist/></cascade>
        </many-to-one>
        </attributes>
        </entity>
        <entity class-name="acme.Department" name="Department" access="FIELD">
        <attributes>
        <id name="deptid" />
        <basic name="name" />
        <basic name="budget" />
        <basic name="isManager" />
        <one-to-many name="employees"
        target-entity="acme.Employee"
        fetch="LAZY" mapped-by="parentNode">
        <cascade><cascade-persist/></cascade>
        </one-to-many>
        </attributes>
        </entity>
        </entity-mappings>
```

Defining indexes for non-entities using XML

Indexes for non-entity types are defined in XML. There is no difference when creating the MapIndexPlugin for entity maps and non-entity maps.

```
Java bean
public class Employee {
        int empid;
        String name
        double salary
        Department dept;

        public class Department {
        int deptid;
```

```
        String name;
        double budget;
        boolean isManager;
        Collection employees;
        }
```

```
ObjectGrid XML with attribute indexes
```

```
<?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="DepartmentGrid">
        <backingMap name="Employee" pluginCollectionRef="Emp"/>
        <backingMap name="Department" pluginCollectionRef="Dept"/>
        <querySchema>
        <mapSchemas>
        <mapSchema mapName="Employee" valueClass="acme.Employee"
        primaryKeyField="empid" />
        <mapSchema mapName="Department" valueClass="acme.Department"
        primaryKeyField="deptid" />
        </mapSchemas>
        <relationships>
        <relationship source="acme.Employee"
        target="acme.Department"
        relationField="dept" invRelationField="employees" />
        </relationships>
        </querySchema>
        </objectGrid>
        </objectGrids>
        <backingMapPluginCollections>
        <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxsqryoptmiz_Emp">
        <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxsqryoptmiz_MapIndexPlugin"
className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
        <property name="Name" type="java.lang.String" value="Employee.name"/>
        <property name="AttributeName" type="java.lang.String" value="name"/>
        <property name="RangeIndex" type="boolean" value="true"
        description="Ranges are must be set to true for attributes." />
        </bean>
        <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxsqryoptmiz_MapIndexPlugin"
className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
        <property name="Name" type="java.lang.String" value="Employee.salary"/>
        <property name="AttributeName" type="java.lang.String" value="salary"/>
        <property name="RangeIndex" type="boolean" value="true"
        description="Ranges are must be set to true for attributes." />
        </bean>
        </backingMapPluginCollection>
        <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxsqryoptmiz_Dept">
        <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxsqryoptmiz_MapIndexPlugin"
className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
        <property name="Name" type="java.lang.String" value="Department.name"/>
        <property name="AttributeName" type="java.lang.String" value="name"/>
        <property name="RangeIndex" type="boolean" value="true"
        description="Ranges are must be set to true for attributes." />
        </bean>
        <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxsqryoptmiz_MapIndexPlugin"
className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
        <property name="Name" type="java.lang.String" value="Department.budget"/>
        <property name="AttributeName" type="java.lang.String" value="budget"/>
        <property name="RangeIndex" type="boolean" value="true"
        description="Ranges are must be set to true for attributes." />
        </bean>
        </backingMapPluginCollection>
        </backingMapPluginCollections>
        </objectGridConfig>
```

Indexing relationships

WebSphere eXtreme Scale stores the foreign keys for related entities within the parent object. For entities, the keys are stored in the underlying tuple. For non-entity objects, the keys are explicitly stored in the parent object.

Adding an index on a relationship attribute can speed up queries that use cyclical references or use the IS NULL, IS EMPTY, SIZE and MEMBER OF query filters. Both single- and multi-valued associations may have the @Index annotation or a HashIndex plug-in configuration in an ObjectGrid descriptor XML file.

Defining entity relationship indexes using @Index

The following example defines entities with @Index annotations:

```
Entity with annotation
```

```
@Entity
public class Node {
```

```
    @ManyToOne @Index
    Node parentNode;

    @OneToMany @Index
    List<Node> childrenNodes = new ArrayList();

    @OneToMany @Index
    List<BusinessUnitType> businessUnitTypes = new ArrayList();
}
```

Defining entity relationship indexes using XML

The following example defines the same entities and indexes using XML with HashIndex plug-ins:

```
        Entity without annotations

        public class Node {
        int nodeId;
        Node parentNode;
        List<Node> childrenNodes = new ArrayList();
        List<BusinessUnitType> businessUnitTypes = new ArrayList();
        }

ObjectGrid 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="ObjectGrid_Entity" entityMetadataXMLFile="entity.xml>
        <backingMap name="Node" pluginCollectionRef="Node"/>
        <backingMap name="BusinessUnitType" pluginCollectionRef="BusinessUnitType"/>
        </objectGrid>
        </objectGrids>
        <backingMapPluginCollections>
        <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxsqryoptmiz_Node">
        <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxsqryoptmiz_MapIndexPlugin"
className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
        <property name="Name" type="java.lang.String" value="parentNode"/>
        <property name="AttributeName" type="java.lang.String" value="parentNode"/>
<property name="RangeIndex" type="boolean" value="false"
        description="Ranges are not supported for association indexes." />      </bean>
        <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxsqryoptmiz_MapIndexPlugin"
className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
        <property name="Name" type="java.lang.String" value="businessUnitType"/>
        <property name="AttributeName" type="java.lang.String" value="businessUnitTypes"/>

<property name="RangeIndex" type="boolean" value="false"
        description="Ranges are not supported for association indexes." />
        </bean>
        <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxsqryoptmiz_MapIndexPlugin"
className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
        <property name="Name" type="java.lang.String" value="childrenNodes"/>
        <property name="AttributeName" type="java.lang.String" value="childrenNodes"/>
<property name="RangeIndex" type="boolean" value="false"
        description="Ranges are not supported for association indexes." />
        </bean>
        </backingMapPluginCollection>
        </backingMapPluginCollections>
        </objectGridConfig>

 Entity XML

        <?xml version="1.0" encoding="UTF-8"?>
        <entity-mappings xmlns="http://ibm.com/ws/projector/config/emd"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://ibm.com/ws/projector/config/emd ./emd.xsd">

        <description>My entities</description>
        <entity class-name="acme.Node" name="Account" access="FIELD">
        <attributes>
        <id name="nodeId" />
        <one-to-many name="childrenNodes"
        target-entity="acme.Node"
        fetch="EAGER" mapped-by="parentNode">
        <cascade><cascade-all/></cascade>
        </one-to-many>
        <many-to-one name="parentNodes"
        target-entity="acme.Node"
        fetch="LAZY" mapped-by="childrenNodes">
        <cascade><cascade-none/></cascade>
        </one-to-many>
        <many-to-one name="businessUnitTypes"
        target-entity="acme.BusinessUnitType"
        fetch="EAGER">
        <cascade><cascade-persist/></cascade>
```

```
        </many-to-one>
</attributes>
        </entity>
        <entity class-name="acme.BusinessUnitType" name="BusinessUnitType" access="FIELD">
        <attributes>
        <id name="buId" />
        <basic name="TypeDescription" />
        </attributes>
        </entity>
        </entity-mappings>
```

Using the previously defined indexes, the following entity query examples are optimized:

```
SELECT n FROM Node n WHERE n.parentNode is null
SELECT n FROM Node n WHERE n.businessUnitTypes is EMPTY
        SELECT n FROM Node n WHERE size(n.businessUnitTypes)>=10
        SELECT n FROM BusinessUnitType b, Node n WHERE b member of n.businessUnitTypes and b.name='TELECOM'
```

Defining non-entity relationship indexes

The following example defines a HashIndex plug-in for non-entity maps in an ObjectGrid descriptor 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="ObjectGrid_POJO">
      <backingMap name="Node" pluginCollectionRef="Node"/>
      <backingMap name="BusinessUnitType" pluginCollectionRef="BusinessUnitType"/>
      <querySchema>
        <mapSchemas>
          <mapSchema mapName="Node"
                                        valueClass="com.ibm.websphere.objectgrid.samples.entity.Node"
            primaryKeyField="id" />
          <mapSchema mapName="BusinessUnitType"
            valueClass="com.ibm.websphere.objectgrid.samples.entity.BusinessUnitType"
            primaryKeyField="id" />
        </mapSchemas>
        <relationships>
          <relationship source="com.ibm.websphere.objectgrid.samples.entity.Node"
            target="com.ibm.websphere.objectgrid.samples.entity.Node"
            relationField="parentNodeId" invRelationField="childrenNodeIds" />
          <relationship source="com.ibm.websphere.objectgrid.samples.entity.Node"
            target="com.ibm.websphere.objectgrid.samples.entity.BusinessUnitType"
            relationField="businessUnitTypeKeys" invRelationField="" />
        </relationships>
      </querySchema>
    </objectGrid>
  </objectGrids>
  <backingMapPluginCollections>
    <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxsqryoptmiz_Node">
      <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxsqryoptmiz_MapIndexPlugin"
className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
        <property name="Name" type="java.lang.String" value="parentNode"/>
<property name="Name" type="java.lang.String" value="parentNodeId"/>
<property name="AttributeName" type="java.lang.String" value="parentNodeId"/>
<property name="RangeIndex" type="boolean" value="false"
        description="Ranges are not supported for association indexes." />
      </bean>
      <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxsqryoptmiz_MapIndexPlugin"
className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
        <property name="Name" type="java.lang.String" value="businessUnitType"/>
        <property name="AttributeName" type="java.lang.String" value="businessUnitTypeKeys"/>

<property name="RangeIndex" type="boolean" value="false"
        description="Ranges are not supported for association indexes." />
   </bean>
      <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxsqryoptmiz_MapIndexPlugin"
className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
<property name="Name" type="java.lang.String" value="childrenNodeIds"/>
        <property name="AttributeName" type="java.lang.String" value="childrenNodeIds"/>
  <property name="RangeIndex" type="boolean" value="false"
                                        description="Ranges are not supported for association indexes." />
</bean>
    </backingMapPluginCollection>
  </backingMapPluginCollections>
</objectGridConfig>
```

Given the above index configurations, the following object query examples are optimized:

```
SELECT n FROM Node n WHERE n.parentNodeId is null
SELECT n FROM Node n WHERE n.businessUnitTypeKeys is EMPTY
SELECT n FROM Node n WHERE size(n.businessUnitTypeKeys)>=10
SELECT n FROM BusinessUnitType b, Node n WHERE
        b member of n.businessUnitTypeKeys and b.name='TELECOM'
```

Tuning EntityManager interface performance

The EntityManager interface separates applications from the state held in its server grid data store.

The cost of using the EntityManager interface is not high and depends on the type of work being performed. Always use the EntityManager interface and optimize the crucial business logic after the application is complete. You can rework any code that uses EntityManager interfaces to use maps and tuples. Generally, this code rework might be necessary for 10 percent of the code.

If you use relationships between objects, then the performance impact is lower because an application that is using maps needs to manage those relationships similarly to the EntityManager interface.

Applications that use the EntityManager interface do not need to provide an ObjectTransformer implementation. The applications are optimized automatically.

Reworking EntityManager code for maps

A sample entity follows:

```
@Entity
public class Person
{
        @Id
        String ssn;
        String firstName;
        @Index
        String middleName;
        String surname;
}
```

Some code to find the entity and update the entity follows:

```
Person p = null;
s.begin();
p = (Person)em.find(Person.class, "1234567890");
p.middleName = String.valueOf(inner);
s.commit();
```

The same code using Maps and Tuples follows:

```
Tuple key = null;
key = map.getEntityMetadata().getKeyMetadata().createTuple();
key.setAttribute(0, "1234567890");

// The Copy Mode is always NO_COPY for entity maps if not using COPY_TO_BYTES.
// Either we need to copy the tuple or we can ask the ObjectGrid to do it for us:
map.setCopyMode(CopyMode.COPY_ON_READ);
s.begin();
Tuple value = (Tuple)map.get(key);
value.setAttribute(1, String.valueOf(inner));
map.update(key, value);
value = null;
s.commit();
```

Both of these code snippets have the same result, and an application can use either or both snippets.
The second code snippet shows how to use maps directly and how to work with the tuples (the key and value pairs). The value tuple has three attributes: firstName, middlename, and surname, indexed at 0, 1, and 2. The key tuple has a single attribute the ID number is indexed at zero. You can see how Tuples are created by using the EntityMetadata#getKeyMetaData or EntityMetadata#getValueMetaData methods. You must use these methods to create Tuples for an Entity. You cannot implement the Tuple interface and pass an instance of your Tuple implementation.

- Entity performance instrumentation agent
  You can improve the performance of field-access entities by enabling the WebSphere® eXtreme Scale instrumentation agent when using Java™ Development Kit (JDK) Version 5 or later.

Related concepts:
Caching objects and their relationships (EntityManager API)
Entity manager in a distributed environment
Interacting with EntityManager
EntityManager fetch plan support
Entity query queues
Routing cache objects to the same partition
Related tasks:
Tutorial: Storing order information in entities
Collocating multiple cache objects in the same partition
Related reference:
Entity performance instrumentation agent
Defining an entity schema
Entity listeners and callback methods
Entity listener examples
EntityTransaction interface
Related information:
Sample: Running Queries in Parallel using a ReduceGridAgent

Entity performance instrumentation agent

You can improve the performance of field-access entities by enabling the WebSphere® eXtreme Scale instrumentation agent when using Java™ Development Kit (JDK) Version 5 or later.

Enabling eXtreme Scale agent on JDK Version 5 or later

The ObjectGrid agent can be enabled with a Java command line option with the following syntax:

```
-javaagent:jarpath[=options]
```

The *jarpath* value is the path to an eXtreme Scale runtime Java archive (JAR) file that contains eXtreme Scale agent class and supporting classes such as the objectgrid.jar, wsobjectgrid.jar, ogclient.jar, wsogclient.jar, and ogagent.jar files. Typically, in a stand-alone Java program or in a Java Platform, Enterprise Edition environment that is not running WebSphere Application Server, use the objectgrid.jar or ogclient.jar file. In a WebSphere Application Server or a multi-classloaders environment, you must use the ogagent.jar file in the Java command line agent option. Provide the ogagent.config file in the class path or use agent options to specify additional information.

eXtreme Scale agent options

config
: Overrides the configuration file name.

include
: Specifies or overrides transformation domain definition that is the first part of the configuration file.

exclude
: Specifies or overrides the @Exclude definition.

fieldAccessEntity
: Specifies or overrides the @FieldAccessEntity definition.

trace
: Specifies a trace level. Levels can be ALL, CONFIG, FINE, FINER, FINEST, SEVERE, WARNING, INFO, and OFF.

trace.file
: Specifies the location of the trace file.

The semicolon (`;`) is used as a delimiter to separate each option. The comma (`,`) is used as a delimiter to separate each element within an option. The following example demonstrates the eXtreme Scale agent option for a Java program:

```
-javaagent:objectgridRoot/lib/objectgrid.jar=config=myConfigFile;
include=includedPackage;exclude=excludedPackage;
fieldAccessEntity=package1,package2
```

ogagent.config file

The ogagent.config file is the designated eXtreme Scale agent configuration file name. If the file name is in the class path, the eXtreme Scale agent finds and parses the file. You can override the designated file name through the config option of eXtreme Scale agent. The following example shows how to specify the configuration file:

```
-javaagent:objectgridRoot/lib/objectgrid.jar=config=myOverrideConfigFile
```

An eXtreme Scale agent configuration file has the following parts:

- **Transformation domain:** The transformation domain part is first in the configuration file. The transformation domain is a list of packages and classes that are included in the class transformation process. This transformation domain must include all classes that are field-access entity classes, and other classes that refer to these field-access entity classes. Field-access entity classes and those classes that refer to these field-access entity classes construct the transformation domain. If you plan to specify field-access entity classes in the @FieldAccessEntity part, then you do not need to include field-access entity classes here. The transformation domain must be complete. Otherwise, you might see a FieldAccessEntityNotInstrumentedException exception.
- **@Exclude:** The @Exclude token indicates that packages and classes listed after this token are excluded from the transformation domain.
- **@FieldAccessEntity:** The @FieldAccessEntity token indicates that packages and classes listed after this token are field-access Entity packages and classes. If no line exists after the @FieldAccessEntity token, then its equivalent is "No @FieldAccessEntity specified". The eXtreme Scale agent determines that there are no field-access Entity packages and classes defined. If there are lines after the @FieldAccessEntity token, then they represent the user-specified field-access Entity packages and classes. For example, "field-access entity domain". The field-access entity domain is a sub-domain of the transformation domain. Packages and classes that are listed in the field-access entity domain are a part of the transformation domain, even when they are not listed in the transformation domain. The @Exclude token, which lists packages and classes that are excluded from transformation, has no impact on the field-access Entity domain. When @FieldAccessEntity token is specified, all field-access entities must be in this field-access Entity domain. Otherwise, a FieldAccessEntityNotInstrumentedException exception might occur.

Example agent configuration file (ogagent.config)

```
################################
# The # indicates comment line
################################
# This is an ObjectGrid agent config file (the designated file name is ogagent.config) that can be found and parsed by
the ObjectGrid agent
# if it is in classpath.
# If the file name is "ogagent.config" and in classpath, Java program runs with -javaagent:objectgridRoot/ogagent.jar
will have
```

```
# ObjectGrid agent enabled.
# If the file name is not "ogagent.config" but in classpath, you can specify the file name in config option of
ObjectGrid agent
#     -javaagent:objectgridRoot/lib/objectgrid.jar=config=myOverrideConfigFile
# See comments below for more info regarding instrumentation setting override.

# The first part of the configuration is the list of packages and classes that should be included in transformation
domain.
# The includes (packages/classes, construct the instrumentation doamin) should be in the beginning of the file.
com.testpackage
com.testClass

# Transformation domain: The above lines are packages/classes that construct the transformation domain.
# The system will process classes with name starting with above packages/classes for transformation.
#
# @Exclude token : Exclude from transformation domain.
# The @Exclude token indicates packages/classes after that line should be excluded from transformation domain.
# It is used when user want to exclude some packages/classes from above specified included packages
#
# @FieldAccessEntity token: Field-access Entity domain.
# The @FieldAccessEntity token indicates packages/classes after that line are field-access Entity packages/classes.
# If there is no lilne after the @FieldAccessEntity token, it is equivalent to "No @FieldAccessEntity specified".
# The runtime will consider the user does not specify any field-access Entity packages/classes.
# The "field-acces Entity domain" is a sub-domain of transformation domain.
#
# Packages/classes listed in the "field-access Entity domain" will always be part of transformation domain,
# even they are not listed in transformation domain.
# The @Exclude, which lists packages/classes excluded from transformation, has no impact on the "field-acces Entity
domain".
# Note: When @FieldAccessEntity is specified, all field-access entities must be in this field-acces Entity domain,
#       otherwise, FieldAccessEntityNotInstrumentedException may occur.
#
# The default ObjectGrid agent config file name is ogagent.config
# The runtime will look for this file as a resource in classpath and process it.
# Users can override this designated ObjectGrid agent config file name via config option of agent.
#
# e.g.
#       javaagent:objectgridRoot/lib/objectgrid.jar=config=myOverrideConfigFile
#
# The instrumentation definition, including transformation domain, @Exclude, and  @FieldAccessEntity can be overriden
individually
# by corresponding designated agent options.
# Designated agent options include:
#    include              -> used to override instrumentation domain definition that is the first part of the config
file
#    exclude              -> used to override @Exclude definition
#    fieldAccessEntity    -> used to override @FieldAccessEntity definition
#
# Each agent option should be separated by ";"
# Within the agent option, the package or class should be seperated by ","
#
# The following is an example that does not override the config file name:
#    -
javaagent:objectgridRoot/lib/objectgrid.jar=include=includedPackage;exclude=excludedPackage;fieldAccessEntity=package1,p
ackage2
#
################################

@Exclude
com.excludedPackage
com.excludedClass

@FieldAccessEntity
```

Performance consideration

For better performance, specify the transformation domain and field-access entity domain.

Related concepts:

Tuning EntityManager interface performance

Caching objects and their relationships (EntityManager API)

Entity manager in a distributed environment

Interacting with EntityManager

EntityManager fetch plan support

Entity query queues

Routing cache objects to the same partition

Related tasks:

Tutorial: Storing order information in entities

Collocating multiple cache objects in the same partition

Related information:

Sample: Running Queries in Parallel using a ReduceGridAgent

Security

WebSphere® eXtreme Scale can secure data access, including allowing for integration with external security providers. Aspects of security include authentication, authorization, transport security, data grid security, local security, and JMX (MBean) security.

- Data grid authentication
  You can use the secure token manager plug-in to enable server-to-server authentication, which requires you to implement the SecureTokenManager interface.
- Data grid security
  Data grid security ensures that a joining server has the right credentials, so a malicious server cannot join the data grid. Data grid security uses a shared secret string mechanism.
- Authenticating and authorizing clients
  You can enable security and credential authentication to authenticate clients. In addition, you can authorize administrative clients to access the data grid.
- Configuring secure transport types
  Transport layer security (TLS) provides secure communication between the client and server. The communication mechanism that is used depends on the value of the transportType parameter that is specified in the client and server configuration files.
- Java Management Extensions (JMX) security
  You can secure managed beans (MBean) invocations in a distributed environment.
- Security integration with external providers
  To protect your data, the product can integrate with several security providers.
- Securing the REST data service
  Secure multiple aspects of the REST data service. Access to the eXtreme Scale REST data service can be secured through authentication and authorization. Access can also be controlled by service-scoped configuration rules, known as access rules. Transport security is the third consideration.
- Security integration with WebSphere Application Server
  When WebSphere eXtreme Scale is deployed in a WebSphere Application Server environment, you can simplify the authentication flow and transport layer security configuration from WebSphere Application Server.
- Enabling data grid authorization
  WebSphere eXtreme Scale provides several security endpoints to integrate custom mechanisms. In the local programming model, the main security function is authorization, and has no authentication support. You must authenticate independently from the already existing WebSphere Application Server authentication. However, you can use the provided plug-ins to obtain and validate Subject objects.
- Starting and stopping secure servers
  Security is enabled by specifying security-specific configurations when you start and stop servers.
- Configuring security profiles for the xscmd utility
  By creating a security profile, you can use saved security parameters to use the **xscmd** utility with secure environments.
- Securing J2C client connections
  Use the Java™ 2 Connector (J2C) architecture to secure connections between WebSphere eXtreme Scale clients and your applications.
- Programming for security
  Use programming interfaces to handle various aspects of security in a WebSphere eXtreme Scale environment.

Data grid authentication

You can use the secure token manager plug-in to enable server-to-server authentication, which requires you to implement the SecureTokenManager interface.

The generateToken(Object) method takes an object protect, and then generates a token that cannot be understood by others. The verifyTokens(byte[]) method does the reverse process: it converts the token back to the original object.
A simple SecureTokenManager implementation uses a simple encoding algorithm, such as a XOR algorithm, to encode the object in serialized form and then use corresponding decoding algorithm to decode the token. This implementation is not secure and is easy to break.
WebSphere® eXtreme Scale default implementation
WebSphere eXtreme Scale provides an immediately available implementation for this interface. This default implementation uses a key pair to sign and verify the signature, and uses a secret key to encrypt the content. Every server has a JCEKS type keystore to store the key pair, a private key and public key, and a secret key. The keystore has to be the JCEKS type to store secret keys. These keys are used to encrypt and sign or verify the secret string on the sending end. Also, the token is associated with an expiration time. On the receiving end, the data is verified, decrypted, and compared to the receiver secret string. Secure Sockets Layer (SSL) communication protocols are not required between a pair of servers for authentication because the private keys and public keys serve the same purpose. However, if server communication is not encrypted, the data can be stolen by looking at the communication. Because the token expires soon, the replay attack threat is minimized. This possibility is significantly decreased if all servers are deployed behind a firewall.
The disadvantage of this approach is that the WebSphere eXtreme Scale administrators have to generate keys and transport them to all servers, which can cause security breach during transportation.
Related concepts:
Data grid security
Related tasks:
Authenticating and authorizing clients
Authenticating application clients
Authorizing application clients
Related reference:
Client properties file
Class ClientSecurityConfigurationFactory

Data grid security

Data grid security ensures that a joining server has the right credentials, so a malicious server cannot join the data grid. Data grid security uses a shared secret string mechanism.

All WebSphere® eXtreme Scale servers, including catalog servers, agree on a shared secret string. When a server joins the data grid, it is challenged to present the secret string. If the secret string of the joining server matches the string in the president server or catalog server, the joining server is accepted. If the string does not match, the join request is rejected.

Sending a clear text secret is not secure. The WebSphere eXtreme Scale security infrastructure provides a secure token manager plug-in to allow the server to secure this secret before sending. You must decide how to implement the secure operation. WebSphere eXtreme Scale provides an out-of-the-box implementation, in which the secure operation is implemented to encrypt and sign the secret.

The secret string is set in the server.properties file. See Server properties file for more information about the authenticationSecret property.

SecureTokenManager plug-in

A secure token manager plug-in is represented by the com.ibm.websphere.objectgrid.security.plugins.SecureTokenManager interface.

For more information about the SecureTokenManager plug-in, see SecureTokenManager API documentation.

The generateToken(Object) method takes an object, and then generates a token that cannot be understood by others. The verifyTokens(byte[]) method does the reverse process: the method converts the token back to the original object.

A simple SecureTokenManager implementation uses a simple encoding algorithm, such as an exclusive or (XOR) algorithm, to encode the object in serialized form and then use the corresponding decoding algorithm to decode the token. This implementation is not secure.

WebSphere eXtreme Scale provides an immediately available implementation for this interface.

The default implementation uses a key pair to sign and verify the signature, and uses a secret key to encrypt the content. Every server has a JCEKS type keystore to store the key pair, a private key and public key, and a secret key. The keystore has to be the JCEKS type to store secret keys.

These keys are used to encrypt and sign or verify the secret string on the sending end. Also, the token is associated with an expiration time. On the receiving end, the data is verified, decrypted, and compared to the receiver secret string. Secure Sockets Layer (SSL) communication protocols are not required between a pair of servers for authentication because the private keys and public keys serve the same purpose. However, if server communication is not encrypted, the data can be stolen by looking at the communication. Because the token expires soon, the replay attack threat is minimized. This possibility is significantly decreased if all servers are deployed behind a firewall.

The disadvantage of this approach is that the WebSphere eXtreme Scale administrators have to generate keys and transport them to all servers, which can cause security breach during transportation.

Sample scripts to create default secure token manager properties

As noted in the previous section, you can create a keystore that contains a key pair to sign and verify the signature and a secret key to encrypt the content.

For example, you can use the JDK 6 keytool command to create the keys as follows:

```
keytool -genkeypair -alias keypair1 -keystore key1.jck -storetype JCEKS -keyalg
rsa -dname "CN=sample.ibm.com, OU=WebSphere eXtreme Scale" -storepass key111 -keypass
keypair1 -validity 10000
```

```
keytool -genseckey -alias seckey1 -keystore key1.jck -storetype JCEKS -keyalg
DES  -storepass key111 -keypass seckey1 -validity 1000
```

These two commands create a key pair "keypair1" and a secret key "seckey1". You can then configure the following in the server property file:

```
secureTokenKeyStore=key1.jck
secureTokenKeyStorePassword=key111
secureTokenKeyStoreType=JCEKS
secureTokenKeyPairAlias=keypair1
secureTokenKeyPairPassword=keypair1
secureTokenSecretKeyAlias=seckey1
secureTokenSecretKeyPassword=seckey1
secureTokenCipherAlgorithm=DES
secureTokenSignAlgorithm=RSA
```

Configuration

See Server properties for more information about the properties that you use to configure the secure token manager.
Related concepts:
Data grid authentication
Related tasks:
Authenticating and authorizing clients
Authenticating application clients
Authorizing application clients
Related reference:
Client properties file
Class ClientSecurityConfigurationFactory

Transport layer security and secure sockets layer

WebSphere® eXtreme Scale supports both TCP/IP and Transport Layer Security/Secure Sockets Layer (TLS/SSL) for secure communication between clients and servers.

TLS and SSL encryption for clients and servers

TLS/SSL is sometimes enabled in one direction. For example, the server public certificate is imported in the client truststore, but the client public certificate is not imported into the server truststore. However, WebSphere eXtreme Scale extensively uses data grid agents. A characteristic of a data grid agent, when ORB transport is used, is that when the server sends responses back to the client, it creates a new connection. The eXtreme Scale server then acts as a client. Therefore, you must import the client public certificate into the server truststore.

Transport layer security for the Oracle JDK

Use the Oracle JRE for SSL with the following limitations.

- When you use the ORB transport, the eXtreme Scale client can run SSL or TLS using the Oracle JRE. When you use the Oracle JRE, specify the Sun JSSE provider (instead of the IBMJSSE2 provider) in the contextProvider property of the eXtreme Scale client properties file (for clients) or the eXtreme Scale server properties file (for servers).

Configuring secure transport types

Transport layer security (TLS) provides secure communication between the client and server. The communication mechanism that is used depends on the value of the transportType parameter that is specified in the client and server configuration files.

About this task

When Secure Sockets Layer (SSL) is used, the SSL configuration parameters must be provided on both the client and server side. In a Java™ SE environment, the SSL configuration is configured in the client or server property files. If the client or server is in WebSphere® Application Server, then you can use the existing WebSphere Application Server CSIV2 transport settings for your container servers and clients. See Security integration with WebSphere Application Server for more information.

Table 1. Transport protocol to use under client transport and server transport settings.
If the transportType settings are different between the client and server, the resulting protocol can vary or result in an error.

Client transportType property	Server transportType property	Resulting protocol
TCP/IP	TCP/IP	TCP/IP
TCP/IP	SSL-supported	TCP/IP
TCP/IP	SSL-required	Error
SSL-supported	TCP/IP	TCP/IP
SSL-supported	SSL-supported	SSL (if SSL fails, then TCP/IP)
SSL-supported	SSL-required	SSL
SSL-required	TCP/IP	Error
SSL-required	SSL-supported	SSL
SSL-required	SSL-required	SSL

Procedure

1. To set the transportType property in the client security configuration, see Client properties file.
2. To set the transportType property in the container and catalog server security configuration, see Server properties file.

- Transport layer security and secure sockets layer
  WebSphere eXtreme Scale supports both TCP/IP and Transport Layer Security/Secure Sockets Layer (TLS/SSL) for secure communication between clients and servers.
- Configuring Secure Sockets Layer (SSL) parameters for clients or servers
  How you configure SSL parameters varies between clients and servers.

Configuring Secure Sockets Layer (SSL) parameters for clients or servers

How you configure SSL parameters varies between clients and servers.

About this task

TLS/SSL is sometimes enabled in one direction. For example, the server public certificate is imported in the client truststore, but the client public certificate is not imported to the server truststore. However, WebSphere® eXtreme Scale extensively uses data grid agents. A characteristic of a data grid agent is when the server sends responds back to the client, it creates a connection. The eXtreme Scale server then acts as a client. Therefore, you must import the client public certificate into the server truststore.

Procedure

- Configure client SSL parameters.
  Use one of the following options to configure SSL parameters on the client:
  - Create a com.ibm.websphere.objectgrid.security.config.SSLConfiguration object by using the com.ibm.websphere.objectgrid.security.config.ClientSecurityConfigurationFactory factory class.
  - Configure the parameters in the client.properties file. You can then either set the property file as a JVM client property or you can use the WebSphere eXtreme Scale APIs. Pass the properties file into the ClientSecurityConfigurationFactory.getClientSecurityConfiguration(String) method for the client and use the returned object as a parameter to the ObjectGridManager.connect(String, ClientSecurityConfiguration, URL) method.
- Configure server SSL parameters.
  SSL parameters are configured for servers using the server.properties file. To start a container or catalog server with a specific property file, use the -serverProps parameter on the **startOgServer** script. For more information about the SSL parameters you can set for eXtreme Scale servers, see Security server properties.

Related reference:
Client properties file

Java Management Extensions (JMX) security

You can secure managed beans (MBean) invocations in a distributed environment.

For more information about the MBeans that are available, see Administering with Managed Beans (MBeans).

In the distributed deployment topology, MBeans are directly hosted in the catalog servers and container servers. In general, JMX security in a distributed topology follows the JMX security specification as specified in the Java Management Extensions (JMX) Specification. It consists of the following three parts:

1. Authentication: The remote client needs to be authenticated in the connector server.
2. Access control: MBean access control limits who can access the MBean information and who can perform the MBean operations.
3. Secure transport: The transport between the JMX client and server can be secured with TLS/SSL.

Authentication

JMX provides methods for the connector servers to authenticate the remote clients. For the RMI connector, authentication is completed by supplying an object that implements the JMXAuthenticator interface when the connector server is created. So eXtreme Scale implements this JMXAuthenticator interface to use the ObjectGrid Authenticator plug-in to authenticate the remote clients. See Java SE security tutorial - Step 2 for details on how eXtreme Scale authenticates a client.

The JMX client follows the JMX APIs to provide credentials to connect to the connector server. The JMX framework passes the credential to the connector server, and then calls the JMXAuthenticator implementation for authentication. As described previously, the JMXAuthenticator implementation then delegates the authentication to the ObjectGrid Authenticator implementation.

Review the following example that describes how to connect to a connector server with a credential:

```
javax.management.remote.JMXServiceURL jmxUrl = new JMXServiceURL(
        "service:jmx:rmi:///jndi/rmi://localhost:1099/objectgrid/MBeanServer");

    environment.put(JMXConnector.CREDENTIALS, new UserPasswordCredential("admin", "xxxxxx"));

    // Create the JMXCconnectorServer
    JMXConnector cntor = JMXConnectorFactory.newJMXConnector(jmxUrl, null);

    // Connect and invoke an operation on the remote MBeanServer
    cntor.connect(environment);
```

In the preceding example, a UserPasswordCredential object is provided with the user ID set to `admin` and the password set to `xxxxx`. This UserPasswordCredential object is set in the environment map, which is used in the JMXConnector.connect(Map) method. This UserPasswordCredential object is then passed to the server by the JMX framework, and finally passed to the ObjectGrid authentication framework for authentication.

The client programming model strictly follows the JMX specification.

Access control

A JMX MBean server might have access to sensitive information and might be able to perform sensitive operations. JMX provides necessary access control that identifies which clients can access that information and who can perform those operations. The access control is built on the standard Java security model by defining permissions that control access to the MBean server and its operations.

For JMX operation access control or authorization, eXtreme Scale relies on the JAAS support provided by the JMX implementation. At any point in the execution of a program, there is a current set of permissions that a thread of execution holds. When such a thread calls a JMX specification operation, these permissions

are known as the held permissions. When a JMX operation is performed, a security check is done to check whether the needed permission is implied by the held permission.

The MBean policy definition follows the Java policy format. For example, the following policy grants all signers and all code bases with the right to retrieve the server JMX address for the PlacementServiceMBean. However, the signers and code bases are restricted to the com.ibm.websphere.objectgrid domain.

```
grant {
    permission javax.management.MBeanPermission
        "com.ibm.websphere.objectgrid.management.PlacementServiceMBean#retrieveServerJMXAddress
                                        [com.ibm.websphere.objectgrid:*,type=PlacementService]",
        "invoke";
}
```

You can use the following policy example to complete authorization based on remote client identity. The policy grants the same MBean permission as shown in the preceding example, except only to users with X500Principal name as: `CN=Administrator,OU=software,O=IBM,L=Rochester,ST=MN,C=US`.

```
grant principal javax.security.auth.x500.X500Principal "CN=Administrator,OU=software,O=IBM,
                                L=Rochester,ST=MN,C=US" {permission javax.management.MBeanPermission
        "com.ibm.websphere.objectgrid.management.PlacementServiceMBean#retrieveServerJMXAddress
                                [com.ibm.websphere.objectgrid:*,type=PlacementService]",
        "invoke";
}
```

Java policies are checked only if the security manager is turned on. Start catalog servers and container servers with the -Djava.security.manager JVM argument to enforce the MBean operation access control.

Secure transport

The transport between the JMX client and server can be secured with TLS/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.

To use SSL, you need to configure the truststore, truststore type, and truststore password on the MBean client with -D system properties:

1. `-Djavax.net.ssl.trustStore=TRUST_STORE_LOCATION`
2. `-Djavax.net.ssl.trustStorePassword=TRUST_STORE_PASSWORD`
3. `-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:

1. `-Dcom.ibm.ssl.trustStore=TRUST_STORE_LOCATION`
2. `-Dcom.ibm.ssl.trustStorePassword=TRUST_STORE_PASSWORD`
3. `-Dcom.ibm.ssl.trustStoreType=TRUST_STORE_TYPE`

To obtain this information when Transport Layer Security/Secure Sockets Layer (TLS/SSL) is enabled in stand-alone configurations, you must start the catalog and container servers with the JMX service port set. Use one of the following methods to set the JMX service port:

- Use the -JMXServicePort option on the **startOgServer** script.
- If you are using an embedded server, call the setJMXServicePort method in the ServerProperties interface to set the JMX service port.

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.
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. Use one of the following methods to set the JMX connector port.

- Use the -JMXConnectorPort option on the **startOgServer** script.
- If you are using an embedded server, call the setJMVConnectorPort method in the ServerProperties interface.

Security integration with external providers

To protect your data, the product can integrate with several security providers.

WebSphere® eXtreme Scale can integrate with an external security implementation. This external implementation must provide authentication and authorization services for WebSphere eXtreme Scale. WebSphere eXtreme Scale has plug-in points to integrate with a security implementation.WebSphere eXtreme Scale has been successfully integrated with the following components:

- Lightweight Directory Access Protocol (LDAP)
- Kerberos
- ObjectGrid security
- Tivoli® Access Manager
- Java™ Authentication and Authorization Service (JAAS)

eXtreme Scale uses the security provider for the following tasks:

- Authenticating clients to servers.
- Authorizing clients to access certain eXtreme Scale artifacts or to specify what can be done with eXtreme Scale artifacts.

eXtreme Scale has the following types of authorizations:

Map authorization
: Clients or groups can be authorized to perform insert, read, update, evict or delete operations on maps.

ObjectGrid authorization
: Clients or groups can be authorized to perform object or entity queries on objectGrids.

DataGrid agent authorization
: Clients or groups can be authorized to allow DataGrid agents to be deployed to an ObjectGrid.

Server-side map authorization
: Clients or groups can be authorized to replicate a server map to client side or create a dynamic index to the server map.

Administration authorization
: Clients or groups can be authorized to perform administration tasks.

Note: If you had security already enabled for your back end , remember that these security settings are no longer sufficient to protect your data. Security settings from your database or other datastore does not in any way transfer to your cache. You must separately protect the data that is now cached using the eXtreme Scale security mechanism, including authentication, authorization, and transport level security.
Important: Use a Development Kit or Runtime Environment at Version 1.6 and later to support SSL Transport security with WebSphere eXtreme Scale Version 7.1.1 and later.

Securing the REST data service

Secure multiple aspects of the REST data service. Access to the eXtreme Scale REST data service can be secured through authentication and authorization. Access can also be controlled by service-scoped configuration rules, known as access rules. Transport security is the third consideration.

About this task

Access to the eXtreme Scale REST data service can be secured through authentication and authorization. Authentication and authorization is accomplished by integrating with eXtreme Scale security.

Access can also be controlled by service-scoped configuration rules, known as access rules Two types of access rules exist, service operation rights which control the CRUD operations that are allowed by the service and entity access rights which control the CRUD operations that are allowed for a particular entity type.

Transport security is provided by the hosting container configuration for connections between the web client and the REST service. And transport security is provided by eXtreme Scale client configuration (for REST service to eXtreme Scale data grid connections).

Procedure

- Control authentication and authorization.
  Access to the eXtreme Scale REST data service can be secured through authentication and authorization. Authentication and authorization are accomplished by integrating with eXtreme Scale security.

  The eXtreme Scale REST data service uses eXtreme Scale security, for authentication and authorization, to control which users can access the service and the operations a user is allowed to perform through the service. The eXtreme Scale REST data service uses either a configured global credential, with user and password, or a credential derived from an HTTP BASIC challenge that is sent with each transaction to the eXtreme Scale data grid where authentication and authorization is performed.

  1. Configure eXtreme Scale client authentication and authorization on the grid See Security integration with external providers for details about how to configure eXtreme Scale client authentication and authorization.
  2. Configure the eXtreme Scale client, which is used by the REST service, for security.
     The eXtreme Scale REST data service invokes the eXtreme Scale client library when communicating with the eXtreme Scale grid. Therefore, the eXtreme Scale client must be configured for eXtreme Scale security.

     eXtreme Scale client authentication is enabled via properties in objectgrid client properties file. At a minimum, the following attributes must be enabled when using client security with the REST service:

     ```
     securityEnabled=true
     credentialAuthentication=Supported [-or-] Required
     credentialGeneratorProps=user:pass [-or-] {xor encoded user:pass}
     ```

     Remember: The user and password specified in the credentialGeneratorProps property must map to an ID in the authentication registry and have sufficient ObjectGrid policy rights to connect to and create ObjectGrids.
     A sample objectgrid client policy file is located in *restservice_home*/security/security.ogclient.properties. See also Client properties file.

  3. Configure the eXtreme Scale REST data service for security.
     The eXtreme Scale REST data service configuration properties file needs to contain the following entries to integrate with eXtreme Scale security:

     ```
     ogClientPropertyFile=file_name
     ```

     The ogClientPropertyFile is the location of the propery file that contains ObjectGrid client properties mentioned in the preceding step. The REST service uses this file to initialize the eXtreme Scale client to talk to the grid when security is enabled.

     ```
     loginType=basic [-or-] none
     ```

     The loginType property configures the REST service for the login type. If a value of `none` is specified, the “global” user id and password defined by the credentialGeneratorProps will be sent to the grid for each transaction. If a value of `basic` is specified, the REST service will present an HTTP

BASIC challenge to the client asking for credentials that it will send in each transaction when communicating with the grid.

For more information about the ogClientPropertyFile and loginType properties, refer to REST data service properties file.

- Apply access rules.
  Access can also be controlled by service scoped configuration rules, known as access rules Two types of access rules exist, service operation rights which control the CRUD operations that are allowed by the service and entity access rights which control the CRUD operations that are allowed for a particular entity type.

  The eXtreme Scale REST data service optionally allows access rules that can be configured to restrict access to the service and entities in the service. These access rules are specified in the REST service access rights property file. The name of this file is specified in the REST data service properties file by the wxsRestAccessRightsFile property. For more information about this property, see REST data service properties file. This file is a typical Java™ property file with key and value pairs. Two types of access rules exist, service operation rights which control the CRUD operations that are allowed by the service and entity access rights which control the CRUD operations that are allowed for a particular entity type.

  1. Configure service operation rights.
     Service Operations rights specify access rights that apply to all the ObjectGrids exposed via the REST service or to all entities of an individual ObjectGrid as specified.

     Use the following syntax.

     ```
     serviceOperationRights=service_operation_right
     serviceOperationRights.grid_name -OR- *=service_operation_right
     ```

     where
     - serviceOperationRights can be one of the following [NONE, READSINGLE, READMULTIPLE, ALLREAD, ALL]
     - serviceOperationRights.*grid_name -OR- ** implies that the access right applies to all the ObjectGrids, else name of a specific ObjectGrid can be provided.

     For example:

     ```
     serviceOperationsRights=ALL
     serviceOperationsRights.*=NONE
     serviceOperationsRights.EMPLOYEEGRID=READSINGLE
     ```

     The first example specifies that all service operations are allowed for all the ObjectGrids exposed by this REST Service. The second example is similar to the first example as it also applies to all the ObjectGrids exposed by the REST service, however it specifies the access right as NONE, which means none of the service operations are allowed on the ObjectGrids. The last example specifies how to control the service operations for a specific grid, here only Reads which results in a single record are allowed for all entities of the EMPLOYEEGRID.

     The default assumed by the REST service is `serviceOperationsRights=ALL` which means that all operations are allowed for all the ObjectGrids exposed by this service. This is different from the Microsoft implementation, for which the default is `NONE`, so no operations are allowed on the REST Service.

     Important: The service operations rights are evaluated in the order they are specified in this file, so the last specified right will override the rights preceding it.
  2. Configure entity access rights.
     Entity set rights specify access rights that apply to specific ObjectGrid entities exposed via the REST service. These rights provide a way to impose tighter and more finer-grained access control on individual ObjectGrid entities than compared to Service Operation rights.

     Use the following syntax.

     ```
     entitySetRights.grid_name.entity_name=entity_set_right
     ```

     where
     - *entity_set_right* can be one of the following rights.

       Table 1. Entity access rights. Supported values.

Access right	Description
NONE	Denies all rights to access data
READSINGLE	Allows to read single data items
READMULTIPLE	Allows reading sets of data
ALLREAD	Allows reading single or multiple sets of data
WRITEAPPEND	Allows creating new data items in data sets
WRITEREPLACE	Allows replacing data
WRITEDELETE	Allows deleting data items from data sets
WRITEMERGE	Allows merging data
ALLWRITE	Allows to write (i.e. create, replace, merge or delete) data
ALL	Allows creating, reading, updating, and deleting data

     - *entity_name* is the name of a specific ObjectGrid within the REST service.
     - *grid_name* is the name of a specific entity within the specified ObjectGrid.

     Note: If both service operation rights and entity set rights are specified for a respective ObjectGrid and its entities, then the more restrictive of those rights will be enforced, as illustrated in the following examples. Note also that the entity set rights are evaluated in the order they are specified in the file. The last specified right will override the rights preceding it.

Example 1: If serviceOperationsRights.NorthwindGrid=READSINGLE and entitySetRights.NorthwindGrid.Customer=ALL are specified. READSINGLE will be enforced for the Customer entity.

Example 2: If serviceOperationsRights.NorthwindGrid=ALLREAD is specified and entitySetRights.NorthwindGrid.Customer=ALLWRITE is specified then only Reads will be allowed for all entities of NorthwindGrid. However for Customer its entity set rights will prevent any Reads (since it specified ALLWRITE) and hence effectively the Customer entity will have access right as NONE.

- Secure transports.
  Transport security is provided by the hosting container configuration for connections between the web client and REST service. Transport security is provided by the eXtreme Scale client configuration for connections between the REST service and the eXtreme Scale grid.

  1. Secure the connection from the client and REST service. Transport security for this connection is provided by the hosting container environment, not in eXtreme Scale.
  2. Secure the connection from the REST service and the eXtreme Scale grid. Transport security for this connection is configured in eXtreme Scale. See Transport layer security and secure sockets layer.

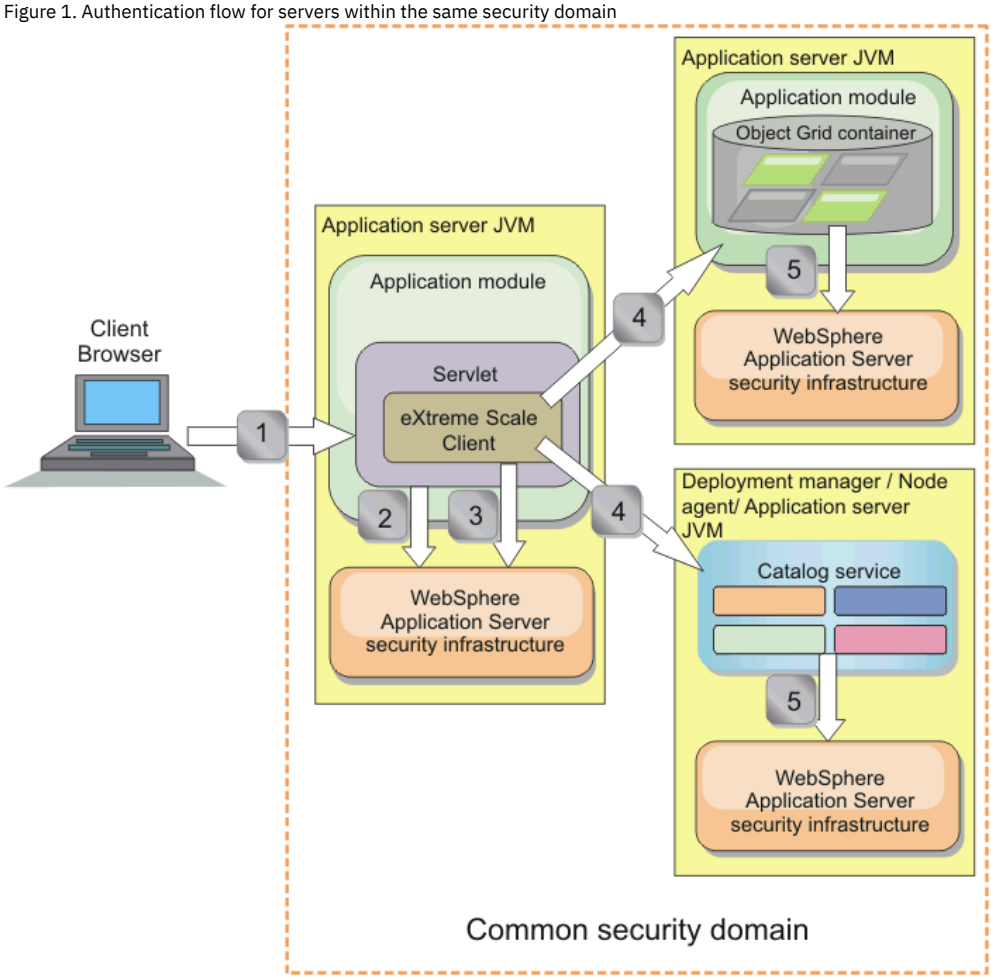
Security integration with WebSphere Application Server

When WebSphere® eXtreme Scale is deployed in a WebSphere Application Server environment, you can simplify the authentication flow and transport layer security configuration from WebSphere Application Server.

Simplified authentication flow

When eXtreme Scale clients and servers are running in WebSphere Application Server and in the same security domain, you can use the WebSphere Application Server security infrastructure to propagate the client authentication credentials to the eXtreme Scale server. For example, if a servlet acts as an eXtreme Scale client to connect to an eXtreme Scale server in the same security domain, and the servlet is already authenticated, it is possible to propagate the authentication token from the client (servlet) to the server, and then use the WebSphere Application Server security infrastructure to convert the authentication token back to the client credentials.

Figure 1. Authentication flow for servers within the same security domain



In the previous diagram, the application servers are in the same security domain. One application server hosts the web application, which is also an eXtreme Scale client. The other application server hosts the container server. The deployment manager or node agent Java virtual machine (JVM) hosts the catalog service.
Note: Use this type of configuration in development environments. However, for production environments run the catalog servers in separate processes, and if possible, run catalog servers on a different system from where the container servers are running.
The arrows in the diagram indicate how the authentication process flows:

1. An enterprise application user uses a Web browser to log in to the first application server with a user name and password.
2. The first application server sends the client user name and password to the WebSphere Application Server security infrastructure to authenticate with the user registry. For example, this user registry might be an LDAP server. As a result, the security information is stored in the application server thread.
3. The JavaServer Pages (JSP) file acts as an eXtreme Scale client to retrieve the security information from the server thread. The JSP file calls the WebSphere Application Server security infrastructure to get the security tokens that represent the enterprise application user.
4. The eXtreme Scale client, or JSP file, sends the security tokens with the request to the container server and catalog service that is hosted in the other JVMs. The catalog server and container server use the WebSphere Application Server security tokens as an eXtreme Scale client credential.
5. The catalog and container servers send the security tokens to the WebSphere Application Server security infrastructure to convert the security tokens into user security information. This user security information is represented by a Subject object, which contains the principals, public credentials, and private credentials. This conversion can occur because the application servers that are hosting the eXtreme Scale client, catalog server, and container server are sharing the same WebSphere Application Server Lightweight Third-Party Authentication (LTPA) tokens.

Authentication integration

Distributed security integration with WebSphere Application Server:

For the distributed model, use the following classes:

- com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredentialGenerator
- com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenAuthenticator
- com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredential

For examples on how to use these classes, see Tutorial: Integrate WebSphere eXtreme Scale security with WebSphere Application Server.

On the server side, use the WSTokenAuthentication authenticator to authenticate the WSTokenCredential object.

Local security integration with WebSphere Application Server:

For the local ObjectGrid model, use the following classes:

- com.ibm.websphere.objectgrid.security.plugins.builtins.WSSubjectSourceImpl
- com.ibm.websphere.objectgrid.security.plugins.builtins.WSSubjectValidationImpl

For more information about these classes, see Local security programming. You can configure the WSSubjectSourceImpl class as the SubjectSource plug-in, and the WSSubjectValidationImpl class as the SubjectValidation plug-in.

Transport layer security support in WebSphere Application Server

When an eXtreme Scale client, container server, or catalog server is running in a WebSphere Application Server process, eXtreme Scale transport security is managed by the WebSphere Application Server CSIV2 transport settings. For the eXtreme Scale client or container server, you should not use eXtreme Scale client or server properties to configure the SSL settings. All the SSL settings should be specified in the WebSphere Application Server configuration.

However, the catalog server is a little different. The catalog server has its own proprietary transport paths which cannot be managed by the WebSphere Application Server CSIV2 transport settings. Therefore, the SSL properties still need to be configured in the server properties file for the catalog server. See Tutorial: Integrate WebSphere eXtreme Scale security with WebSphere Application Server for more information.

- Configuring client security on a catalog service domain
  By configuring client security on a catalog service domain, you can define default client authentication configuration properties. These properties are used when a client properties file is not located in the Java virtual machine (JVM) that is hosting the client or when the client does not programmatically specify security properties. If a client properties file exists, the properties that you specify in the console override the values in the file. You can override these properties by specifying a splicer.properties file with the com.ibm.websphere.xs.sessionFilterProps custom property or by splicing the application EAR file.

Configuring client security on a catalog service domain

By configuring client security on a catalog service domain, you can define default client authentication configuration properties. These properties are used when a client properties file is not located in the Java virtual machine (JVM) that is hosting the client or when the client does not programmatically specify security properties. If a client properties file exists, the properties that you specify in the console override the values in the file. You can override these properties by specifying a splicer.properties file with the com.ibm.websphere.xs.sessionFilterProps custom property or by splicing the application EAR file.

Before you begin

- You must know the CredentialGenerator implementation that you are using to authenticate clients with the remote data grid. You can use one of the implementations that are provided by WebSphere® eXtreme Scale: UserPasswordCredentialGenerator or WSTokenCredentialGenerator.
  You can also use a custom implementation of the CredentialGenerator interface. The custom implementation must be in the class path of the runtime client and the server. If you are configuring an HTTP session scenario with WebSphere Application Server, you must put the implementation in the class path of the deployment manager and the class path of the application server in which the client is running.
- You must have a catalog service domain defined. See Creating catalog service domains in WebSphere Application Server for more information.

About this task

You must configure client security on the catalog service domain when you have enabled credential authentication on the server side, by configuring one of the following scenarios:

- The server-side security policy has the credentialAuthentication property set to `Required`.
- The server-side security policy has the credentialAuthentication property set to `Supported` AND an authorizationMechanism has been specified in the ObjectGrid XML file.

In these scenarios, a credential must be passed from the client. The credential that is passed from the client is retrieved from the getCredential method on a class that implements the CredentialGenerator interface. In an HTTP session configuration scenario, the run time must know the CredentialGenerator implementation to use to generate a credential that is passed to a remote data grid. If you do not specify the CredentialGenerator implementation class to use, the remote data grid would reject requests from the client because the client cannot be authenticated.

Procedure

Define client security properties. In the WebSphere Application Server administrative console, click System administration > WebSphere eXtreme Scale > Catalog service domains > *catalog_service_domain_name* > Client security properties. Specify client security properties on the page and save your changes. See Client security properties for a list of the properties you can set.

Results

The client security properties that you configured on the catalog service domain are used as default values. The values you specify override any properties that are defined in the client.properties files.

What to do next

Configure your applications to use WebSphere eXtreme Scale for session management. See Configuring WebSphere Application Server HTTP session persistence to a data grid for more information.

Enabling data grid authorization

WebSphere® eXtreme Scale provides several security endpoints to integrate custom mechanisms. In the local programming model, the main security function is authorization, and has no authentication support. You must authenticate independently from the already existing WebSphere Application Server authentication. However, you can use the provided plug-ins to obtain and validate Subject objects.

About this task

You can enable local security with the ObjectGrid XML descriptor file or programmatically.

Procedure

- Enable local security with the ObjectGrid XML descriptor XML file.
  The secure-objectgrid-definition.xml file that is used in the ObjectGridSample enterprise application sample is shown in the following example. Set the securityEnabled attribute to `true` to enable security.

```
<objectGrids>
    <objectGrid name="secureClusterObjectGrid" securityEnabled="true"
        authorizationMechanism="AUTHORIZATION_MECHANISM_JAAS">
    ...
</objectGrids>
```

- Enable local security programmatically.
  To create an ObjectGrid using the ObjectGrid.setSecurityEnabled method, call the following method on the ObjectGrid interface:

```
/**
 * Enable the ObjectGrid security
 */
void setSecurityEnabled();
```

What to do next

Start the container and catalog servers with security enabled.
Related reference:
Deployment policy descriptor XML file

Starting and stopping secure servers

Security is enabled by specifying security-specific configurations when you start and stop servers.

- Starting secure servers in a stand-alone environment
  To start secure stand-alone servers, you pass the proper configuration files by specifying parameters on the **startOgServer** or command.
- Starting secure servers in WebSphere Application Server
  To start secure servers in WebSphere® Application Server, you must specify the security configuration files in the generic Java™ virtual machine (JVM)

arguments.
- Stopping secure servers
  Stopping secure catalog servers or container servers requires one security configuration file.

Related tasks:
Configuring WebSphere Application Server applications to automatically start container servers

Starting secure servers in a stand-alone environment

To start secure stand-alone servers, you pass the proper configuration files by specifying parameters on the **startOgServer** or command.

Procedure

- Start secure container servers.
  Starting a secure container server requires the following security configuration file:
  - **Server property file:** The server property file configures the security properties specific to the server. Refer to the Server properties file for more details.

  Specify the location of this configuration file by providing the following argument to the **startOgServer** script:

  -serverProps
  Specifies the location of the server property file, which contains the server-specific security properties. The file name specified for this property is in plain file path format, such as ../security/server.properties.

  Enter the following lines when you run the **startOgServer** command 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
```

  Windows

```
startOgServer.bat <arguments> -jvmargs -Djava.security.auth.login.config=jaas.config
-Djava.security.manager -Djava.security.policy="auth.policy" -Dobjectgrid.home=%OBJECTGRID_HOME%
```

- Start secure catalog servers.
  To start a secure catalog service, you must have the following configuration files:
  - **Security descriptor XML file:** The security descriptor XML file describes the security properties common to all servers, including catalog servers and container servers. One property example is the authenticator configuration which represents the user registry and authentication mechanism.
  - **Server property file:** The server property file configures the security properties that are specific to the server.

  Specify the location of these configuration files by providing the following arguments to the **startOgServer** script:

  -clusterSecurityFile and -clusterSecurityUrl
  These arguments specify the location of the Security descriptor XML file. Use the -clusterSecurityFile parameter to specify a local file, or the -clusterSecurityUrl parameter to specify the URL of the objectGridSecurity.xml file.
  -serverProps
  Specifies the location of the server property file, which contains the server-specific security properties. The file name specified for this property is in plain file path format, such as c:/tmp/og/catalogserver.props.

Starting secure servers in WebSphere Application Server

To start secure servers in WebSphere® Application Server, you must specify the security configuration files in the generic Java™ virtual machine (JVM) arguments.

Procedure

- Associate WebSphere eXtreme Scale catalog servers with WebSphere application servers using the administrative console. In the administrative console, click System Administration > WebSphere eXtreme Scale > Catalog Service Domains.
- Associate WebSphere eXtreme Scale container servers with particular WebSphere application servers by deploying an enterprise archive (EAR) file that contains the required XML descriptors for the data grid. For more information about this procedure, see Tutorial: Integrate WebSphere eXtreme Scale security with WebSphere Application Server.
- Specify Java virtual machine (JVM) arguments that point to configuration files to make the catalog and container servers secure. In addition, specify `securityEnabled="true"` in the objectgrid xml file for each data grid. After you specify the JVM arguments and enable security in your data grids, you can start the servers or clusters that act as eXtreme Scale catalog servers or container servers.
- Start catalog and containers servers with the WebSphere Application Server administrative console, or use the WebSphere Application Server command line.

What to do next

Stopping secure servers

Stopping secure servers

Stopping secure catalog servers or container servers requires one security configuration file.

Procedure

- Stop a secure catalog server or container server in stand-alone deployments. In stand-alone environments, stop WebSphere® eXtreme Scale catalog and container servers using the teardown function of the **xscmd** command, or using the **stopXsServer** or **stopOgServer** commands.
- Use the WebSphere Application Server administrative console to stop eXtreme Scale server that run with WebSphere Application Server.

Configuring WebSphere eXtreme Scale to use FIPS 140-2

Federal Information Processing Standard (FIPS) 140-2 specifies required levels of encryption for Transport Layer Security/Secure Sockets Layer (TLS/SSL). This standard ensures high protection of data as it is sent over the wire.

Before you begin

- You must be using an IBM® Runtime Environment. For more information, see Java SE considerations.
- Configure transport layer security and secure sockets layer in both directions. Your catalog server truststore file must contain the self-signed certificates for the container servers. The container servers must contain the self-signed certificates for the catalog server. For more information, see Transport layer security and secure sockets layer.

About this task

You can use the following steps to configure the catalog servers and container servers in your WebSphere eXtreme Scale stand-alone installation to use FIPS.

Procedure

1. Edit the java.security file. The location of the java.security depends on your Java virtual machine (JVM) configuration:
   - If you are using the default JVM that ships with the product, the file is in the *wxs_install_root*/java/jre/lib/security directory.
   - If you are using a different JVM, edit the file in the *java_home*/jre/lib/security directory.
   - If you installed WebSphere eXtreme Scale on an HP-UX or Solaris operating system, WebSphere eXtreme Scale requires the IBM hybrid JDK. A hybrid platform is a platform where you get 32-bit and 64-bit instance support in the same installation. You must update the java.security file in the jre/lib/security directory to include the IBMJCEFIPS provider first, such as:
     `security.provider.1=com.ibm.crypto.fips.provider.IBMJCEFIPS`.

   The file must contain the following text:

   ```
   security.provider.1=com.ibm.crypto.fips.provider.IBMJCEFIPS
   security.provider.2=com.ibm.jsse2.IBMJSSEProvider2
   security.provider.3=com.ibm.crypto.provider.IBMJCE
   security.provider.4=com.ibm.security.jgss.IBMJGSSProvider
   security.provider.5=com.ibm.security.cert.IBMCertPath
   security.provider.6=com.ibm.security.sasl.IBMSASL
   security.provider.7=com.ibm.xml.crypto.IBMXMLCryptoProvider
   security.provider.8=com.ibm.xml.enc.IBMXMLEncProvider
   security.provider.9=org.apache.harmony.security.provider.PolicyProvider
   security.provider.10=com.ibm.security.jgss.mech.spnego.IBMSPNEGO
   ```

   Important: If you are running Solaris or HP-UX you need to include the following line:

   ```
   security.provider.11=sun.security.provider.Sun
   ```

2. Edit the Secure Sockets Layer (SSL) configuration in the server properties files for the catalog server and container servers to use the TLSv1 protocol. You must also configure any clients that access the catalog and container servers to use the TLSv1 protocol.
   These files must contain the following properties and values:

   ```
   contextProvider=IBMJSSE2
   transportType=SSL-Required
   protocol=TLSv1
   fips=true
   ```

   For more information about SSL properties, see Server properties file and Client properties file.
3. Create a keystore with a FIPS 140-2 compatible certificate, for use by the catalog and container servers. The certificate should not use an MD5 signature algorithm. FIPS does not dictate RSA or key length.
4. Restart your catalog and container servers.
   When you start the catalog servers, you must specify Java virtual machine (JVM) arguments. The arguments that you use depend on which version of Java SE you are using. If the FIPS property is set and -Dcom.ibm.jsse2.usefipsprovider=true argument is set when you start the server, the JVM setting overrides the FIPS option that is configured in the security server properties.
   - For Java 6 SR 10 and later, or Java 7, specify the -Dcom.ibm.jsse2.usefipsprovider=true argument when you start the server.

   Restriction: When WebSphere eXtreme Scale is configured to run with the ORB transport, you cannot configure SSL to use both FIPS encryption and SP800-131a data protection. Running with both security standards is only allowed when eXtreme Scale is configured to run with the eXtremeIO (XIO) transport.

For more information, see Starting and stopping secure servers.

Configuring security profiles for the xscmd utility

By creating a security profile, you can use saved security parameters to use the **xscmd** utility with secure environments.

Before you begin

For more information about setting up the **xscmd** utility, see Administering with the xscmd utility.

About this task

You can use the -ssp *profile_name* or --saveSecProfile *profile_name* parameter with the rest of your **xscmd** command to save a security profile. The profile can contain settings for user names and passwords, credential generators, keystores, truststores, and transport types.

The ProfileManagement command group in the **xscmd** utility contains commands for managing your security profiles.

Procedure

- Save a security profile.
  To save a security profile, use the -ssp *profile_name* or --saveSecProfile *profile_name* parameter with the rest of your command. Adding this parameter to your command saves the following parameters:

  ```
  -al,--alias <alias>
  -arc,--authRetryCount <integer>
  -ca,--credAuth <support>
  -cgc,--credGenClass <className>
  -cgp,--credGenProps <property>
  -cxpv,--contextProvider <provider>
  -ks,--keyStore <filePath>
  -ksp,--keyStorePassword <password>
  -kst,--keyStoreType <type>
  -prot,--protocol <protocol>
  -pwd,--password <password>
  -ts,--trustStore <filePath>
  -tsp,--trustStorePassword <password>
  -tst,--trustStoreType <type>
  -tt,--transportType <type>
  -user,--username <username>
  ```

  Security profiles are saved in the *user_home*\.xscmd\profiles\security\<*profile_name*>.properties file.
  Important: Do not include the .properties file name extension on the *profile_name* parameter. This extension is automatically added to the file name.
- Use a saved security profile.
  To use a saved security profile, add the **-sp** *profile_name* or **--securityProfile** *profile_name* parameter to the command you are running.

  Command example: `xscmd -c listHosts -cep myhost.mycompany.com -sp myprofile`
- List the commands in the ProfileManagement command group.
  Run the following command: **xscmd -lc ProfileManagement**.
- List the existing security profiles.
  Run the following command: **xscmd -c listProfiles -v**.
- Display the settings that are saved in a security profile.
  Run the following command: **xscmd -c showProfile -pn** *profile_name*.
- Remove an existing security profile.
  Run the following command: **xscmd -c RemoveProfile -pn** *profile_name*.

Related tasks:
Java SE security tutorial - Step 4
Administering with the xscmd utility
Monitoring with the xscmd utility
Related reference:
xsadmin tool to xscmd tool migration

Securing J2C client connections

8.5+ 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.
Previous topic: 8.5+ Configuring applications to connect with eXtreme Scale
Next topic: 8.5+ Developing eXtreme Scale client components to use transactions

Programming for security

Use programming interfaces to handle various aspects of security in a WebSphere® eXtreme Scale environment.

- Security API
  WebSphere eXtreme Scale adopts an open security architecture. It provides a basic security framework for authentication, authorization, and transport security, and requires users to implement plug-ins to complete the security infrastructure.
- Client authentication programming
  For authentication, WebSphere eXtreme Scale provides a runtime to send the credential from the client to the server side, and then calls the authenticator plug-in to authenticate the users.

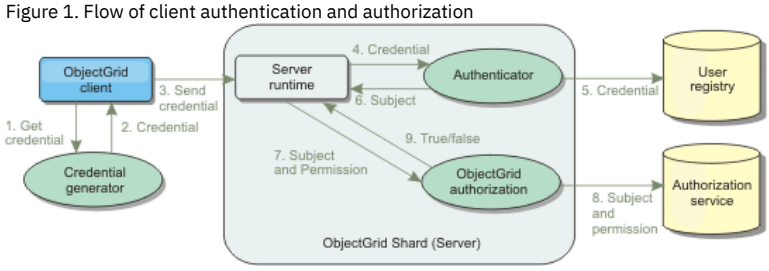
- Client authorization programming
  WebSphere eXtreme Scale supports Java™ Authentication and Authorization Service (JAAS) authorization that is ready to use and also supports custom authorization using the ObjectGridAuthorization interface.
- Data grid authentication
  You can use the secure token manager plug-in to enable server-to-server authentication, which requires you to implement the SecureTokenManager interface.
- Local security programming
  WebSphere eXtreme Scale provides several security endpoints to allow you to integrate custom mechanisms. In the local programming model, the main security function is authorization, and has no authentication support . You must authenticate outside of WebSphere Application Server. However, there are provided plug-ins to obtain and validate Subject objects.

Security API

WebSphere® eXtreme Scale adopts an open security architecture. It provides a basic security framework for authentication, authorization, and transport security, and requires users to implement plug-ins to complete the security infrastructure.

The following image shows the basic flow of client authentication and authorization for an eXtreme Scale server.

Figure 1. Flow of client authentication and authorization



The authentication flow and authorization flow are as follows.

Authentication flow

1. The authentication flow starts with an eXtreme Scale client getting a credential. This is done by the com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator plug-in.
2. A CredentialGenerator object knows how to generate a valid client credential, for example, a user ID and password pair, Kerberos ticket, and so on. This generated credential is sent back to the client.
3. After the client retrieves the Credential object using the CredentialGenerator object, this Credential object is sent along with the eXtreme Scale request to the eXtreme Scale server.
4. The eXtreme Scale server authenticates the Credential object before processing the eXtreme Scale request. Then the server uses the Authenticator plug-in to authenticate the Credential object.
5. The Authenticator plug-in represents an interface to the user registry, for example, a Lightweight Directory Access Protocol (LDAP) server or an operating system user registry. The Authenticator consults the user registry and makes authentication decisions.
6. If the authentication is successful, a Subject object is returned to represent this client.
   Authorization flow

   WebSphere eXtreme Scale adopts a permission-based authorization mechanism, and has different permission categories represented by different permission classes. For example, a com.ibm.websphere.objectgrid.security.MapPermission object represents permissions to read, write, insert, invalidate, and remove the data entries in an ObjectMap. Because WebSphere eXtreme Scale supports Java™ Authentication and Authorization Service (JAAS) authorization out-of-box, you can use JAAS to handle authorization by providing authorization policies.

   Also, eXtreme Scale supports custom authorizations. Custom authorizations are plugged in by the plug-in com.ibm.websphere.objectgrid.security.plugins.ObjectGridAuthorization. The flow of the customer authorization is as follows.

7. The server runtime sends the Subject object and the required permission to the authorization plug-in.
8. The authorization plug-in consults the Authorization service and makes an authorization decision. If permission is granted for this Subject object, a value of `true` is returned, otherwise `false`is returned.
9. This authorization decision, true or false, is returned to the server runtime.

Security implementation

The topics in this section discuss how to program a secure WebSphere eXtreme Scale deployment and how to program the plug-in implementations. The section is organized based on the various security features. In each subtopic, you will learn about relevant plug-ins and how to implement the plug-ins. In the authentication section, you will see how to connect to a secure WebSphere eXtreme Scale deployment environment.

Client Authentication: The client authentication topic describes how a WebSphere eXtreme Scale client gets a credential and how a server authenticates the client. It will also discuss how a WebSphere eXtreme Scale client connects to a secure WebSphere eXtreme Scale server.

Authorization: The authorization topic explains how to use the ObjectGridAuthorization to do customer authorization besides JAAS authorization.

Grid Authentication: The data grid authentication topic discusses how you can use SecureTokenManager to securely transport server secrets.

Java Management Extensions (JMX) programming: When the WebSphere eXtreme Scale server is secured, the JMX client might need to send a JMX credential to the server.

Client authentication programming

For authentication, WebSphere® eXtreme Scale provides a runtime to send the credential from the client to the server side, and then calls the authenticator plug-in to authenticate the users.

WebSphere eXtreme Scale requires you to implement the following plug-ins to complete the authentication.

- Credential: A Credential represents a client credential, such as a user ID and password pair.
- CredentialGenerator: A CredentialGenerator represents a credential factory to generate the credential.
- Authenticator: An Authenticator authenticates the client credential and retrieves client information.

Credential and CredentialGenerator plug-ins

When an eXtreme Scale client connects to a server that requires authentication, the client is required to provide a client credential. A client credential is represented by a com.ibm.websphere.objectgrid.security.plugins.Credential interface. A client credential can be a user name and password pair, a Kerberos ticket, a client certificate, or data in any format that the client and server agree upon. This interface explicitly defines the equals(Object) and hashCode methods. These two methods are important because the authenticated Subject objects are cached by using the Credential object as the key on the server side. WebSphere eXtreme Scale also provides a plug-in to generate a credential. This plug-in is represented by the com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator interface and is useful when the credential can expire. In this case, the getCredential method is called to renew a credential.

The Credential interface explicitly defines the equals(Object) and hashCode methods. These two methods are important because the authenticated Subject objects are cached by using the Credential object as the key on the server side.

You may also use the provided plug-in to generate a credential. This plug-in is represented by the com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator interface, and is useful when the credential can expire. In this case, the getCredential method is called to renew a credential. See CredentialGenerator interface for more details.

There are three provided default implementations for the Credential interfaces:

- The com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredential implementation, which contains a user ID and password pair.
- The com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredential implementation, which contains WebSphere Application Server-specific authentication and authorization tokens. These tokens can be used to propagate the security attributes across the application servers in the same security domain.

WebSphere eXtreme Scale also provides a plug-in to generate a credential. This plug-in is represented by the com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator interface.WebSphere eXtreme Scale provides two default built-in implementations:

- The com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredentialGenerator constructor takes a user ID and a password. When the getCredential method is called, it returns a UserPasswordCredential object that contains the user ID and password.
- The com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredentialGenerator represents a credential (security token) generator when running in WebSphere Application Server. When the getCredential method is called, the Subject that is associated with the current thread is retrieved. Then the security information in this Subject object is converted into a WSTokenCredential object. You can specify whether to retrieve a runAs subject or a caller subject from the thread by using the constant WSTokenCredentialGenerator.RUN_AS_SUBJECT or WSTokenCredentialGenerator.CALLER_SUBJECT.

UserPasswordCredential and UserPasswordCredentialGenerator

For testing purposes, WebSphere eXtreme Scale provides the following plug-in implementations:

1. `com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredential`
2. `com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredentialGenerator`

The user password credential stores a user ID and password. The user password credential generator then contains this user ID and password.

The following example code shows how to implement these two plug-ins.

```
UserPasswordCredential.java
// 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-J34 © COPYRIGHT International Business Machines Corp. 2007
package com.ibm.websphere.objectgrid.security.plugins.builtins;

import com.ibm.websphere.objectgrid.security.plugins.Credential;

/**
 * This class represents a credential containing a user ID and password.
 *
 * @ibm-api
 * @since WAS XD 6.0.1
 *
 * @see Credential
 * @see UserPasswordCredentialGenerator#getCredential()
 */
public class UserPasswordCredential implements Credential {
```

```
    private static final long serialVersionUID = 1409044825541007228L;

    private String ivUserName;

    private String ivPassword;

    /**
     * Creates a UserPasswordCredential with the specified user name and
     * password.
     *
     * @param userName the user name for this credential
     * @param password the password for this credential
     *
     * @throws IllegalArgumentException if userName or password is <code>null</code>
     */
    public UserPasswordCredential(String userName, String password) {
        super();
        if (userName == null || password == null) {
            throw new IllegalArgumentException("User name and password cannot be null.");
        }
        this.ivUserName = userName;
        this.ivPassword = password;
    }

    /**
     * Gets the user name for this credential.
     *
     * @return   the user name argument that was passed to the constructor
     *           or the <code>setUserName(String)</code>
     *           method of this class
     *
     * @see #setUserName(String)
     */
    public String getUserName() {
        return ivUserName;
    }

    /**
     * Sets the user name for this credential.
     *
     * @param userName the user name to set.
     *
     * @throws IllegalArgumentException if userName is <code>null</code>
     */
    public void setUserName(String userName) {
        if (userName == null) {
            throw new IllegalArgumentException("User name cannot be null.");
        }
        this.ivUserName = userName;
    }

    /**
     * Gets the password for this credential.
     *
     * @return   the password argument that was passed to the constructor
     *           or the <code>setPassword(String)</code>
     *           method of this class
     *
     * @see #setPassword(String)
     */
    public String getPassword() {
        return ivPassword;
    }

    /**
     * Sets the password for this credential.
     *
     * @param password the password to set.
     *
     * @throws IllegalArgumentException if password is <code>null</code>
     */
    public void setPassword(String password) {
        if (password == null) {
            throw new IllegalArgumentException("Password cannot be null.");
        }
        this.ivPassword = password;
    }

    /**
     * Checks two UserPasswordCredential objects for equality.
     * <p>
     * Two UserPasswordCredential objects are equal if and only if their user names
     * and passwords are equal.
     *
     * @param o the object we are testing for equality with this object.
     *
     * @return <code>true</code> if both UserPasswordCredential objects are equivalent.
     *
     * @see Credential#equals(Object)
     */
```

```
    public boolean equals(Object o) {
        if (this == o) {
            return true;
        }
        if (o instanceof UserPasswordCredential) {
            UserPasswordCredential other = (UserPasswordCredential) o;
            return other.ivPassword.equals(ivPassword) && other.ivUserName.equals(ivUserName);
        }

        return false;
    }

    /**
     * Returns the hashcode of the UserPasswordCredential object.
     *
     * @return the hash code of this object
     *
     * @see Credential#hashCode()
     */
    public int hashCode() {
        return ivUserName.hashCode() + ivPassword.hashCode();
    }
}
```

```
UserPasswordCredentialGenerator.java
// 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-J34 © COPYRIGHT International Business Machines Corp. 2007
package com.ibm.websphere.objectgrid.security.plugins.builtins;

import java.util.StringTokenizer;

import com.ibm.websphere.objectgrid.security.plugins.Credential;
import com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator;

/**
 * This credential generator creates <code>UserPasswordCredential</code> objects.
 * <p>
 * UserPasswordCredentialGenerator has a one to one relationship with
 * UserPasswordCredential because it can only create a UserPasswordCredential
 * representing one identity.
 *
 * @since WAS XD 6.0.1
 * @ibm-api
 *
 * @see CredentialGenerator
 * @see UserPasswordCredential
 */
public class UserPasswordCredentialGenerator implements CredentialGenerator {

    private String ivUser;

    private String ivPwd;

    /**
     * Creates a UserPasswordCredentialGenerator with no user name or password.
     *
     * @see #setProperties(String)
     */
    public UserPasswordCredentialGenerator() {
        super();
    }

    /**
     * Creates a UserPasswordCredentialGenerator with a specified user name and
     * password
     *
     * @param user the user name
     * @param pwd the password
     */
    public UserPasswordCredentialGenerator(String user, String pwd) {
        ivUser = user;
        ivPwd = pwd;
    }

    /**
     * Creates a new <code>UserPasswordCredential</code> object using this
     * object's user name and password.
     *
     * @return a new <code>UserPasswordCredential</code> instance
     *
     * @see CredentialGenerator#getCredential()
     * @see UserPasswordCredential
     */
    public Credential getCredential() {
        return new UserPasswordCredential(ivUser, ivPwd);
```

```
    }

    /**
     * Gets the password for this credential generator.
     *
     * @return   the password argument that was passed to the constructor
     */
    public String getPassword() {
        return ivPwd;
    }

    /**
     * Gets the user name for this credential.
     *
     * @return   the user argument that was passed to the constructor
     *           of this class
     */
    public String getUserName() {
        return ivUser;
    }
    /**
     * Sets additional properties namely a user name and password.
     *
     * @param properties a properties string  with a user name and
     *                   a password separated by a blank.
     *
     * @throws IllegalArgumentException if the format is not valid
     */
    public void setProperties(String properties) {
        StringTokenizer token = new StringTokenizer(properties, " ");
        if (token.countTokens() != 2) {
            throw new IllegalArgumentException(
                "The properties should have a user name and password and separated by a blank.");
        }

        ivUser = token.nextToken();
        ivPwd = token.nextToken();
    }
    /**
     * Checks two UserPasswordCredentialGenerator objects for equality.
     * <p>
     * Two UserPasswordCredentialGenerator objects are equal if and only if
     * their user names and passwords are equal.
     *
     * @param obj the object we are testing for equality with this object.
     *
     * @return <code>true</code> if both UserPasswordCredentialGenerator objects
     *         are equivalent.
     */
    public boolean equals(Object obj) {
        if (obj == this) {
            return true;
        }

        if (obj != null && obj instanceof UserPasswordCredentialGenerator) {
            UserPasswordCredentialGenerator other = (UserPasswordCredentialGenerator) obj;

            boolean bothUserNull = false;
            boolean bothPwdNull = false;

            if (ivUser == null) {
                if (other.ivUser == null) {
                    bothUserNull = true;
                } else {
                    return false;
                }
            }

            if (ivPwd == null) {
                if (other.ivPwd == null) {
                    bothPwdNull = true;
                } else {
                    return false;
                }
            }

            return (bothUserNull || ivUser.equals(other.ivUser)) && (bothPwdNull || ivPwd.equals(other.ivPwd));
        }

        return false;
    }

    /**
     * Returns the hashcode of the UserPasswordCredentialGenerator object.
     *
     * @return the hash code of this object
     */
    public int hashCode() {

        return ivUser.hashCode() + ivPwd.hashCode();
    }
```

```
}
```

The UserPasswordCredential class contains two attributes: user name and password. The UserPasswordCredentialGenerator serves as a factory that contains the UserPasswordCredential objects.

WSTokenCredential and WSTokenCredentialGenerator

When the WebSphere eXtreme Scale clients and servers are all deployed in WebSphere Application Server, the client application can use these two built-in implementations when the following conditions are satisfied:

1. WebSphere Application Server global security is turned on.
2. All WebSphere eXtreme Scale clients and servers are running in WebSphere Application Server Java™ virtual machines.
3. The application servers are in the same security domain.
4. The client is already authenticated in WebSphere Application Server.

In this situation, the client can use the com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredentialGenerator class to generate a credential. The server uses the WSAuthenticator implementation class to authenticate the credential.

This scenario takes advantage of the fact that the eXtreme Scale client has already been authenticated. Because the application servers that have the servers are in the same security domain as the application servers that house the clients, the security tokens can be propagated from the client to the server so that the same user registry does not need to be authenticated again.

Note: Do not assume that a CredentialGenerator always generates the same credential. For an expirable and refreshable credential, the CredentialGenerator should be able to generate the latest valid credential to make sure the authentication succeeds. One example is using the Kerberos ticket as a Credential object. When the Kerberos ticket refreshes, the CredentialGenerator should retrieve the refreshed ticket when CredentialGenerator.getCredential is called.

Authenticator plug-in

After the eXtreme Scale client retrieves the Credential object using the CredentialGenerator object, this client Credential object is sent along with the client request to the eXtreme Scale server. The server authenticates the Credential object before processing the request. If the Credential object is authenticated successfully, a Subject object is returned to represent this client.

This Subject object is then cached, and it expires after its lifetime reaches the session timeout value. The login session timeout value can be set by using the loginSessionExpirationTime property in the cluster XML file. For example, setting `loginSessionExpirationTime="300"` makes the Subject object expire in 300 seconds.

This Subject object is then used for authorizing the request, which is shown later. An eXtreme Scale server uses the Authenticator plug-in to authenticate the Credential object. See Authenticator for more details.

The Authenticator plug-in is where the eXtreme Scale runtime authenticates the Credential object from the client user registry, for example, a Lightweight Directory Access Protocol (LDAP) server.

WebSphere eXtreme Scale does not provide an immediately available user registry configuration. The configuration and management of user registry is left outside of WebSphere eXtreme Scale for simplicity and flexibility. This plug-in implements connecting and authenticating to the user registry. For example, an Authenticator implementation extracts the user ID and password from the credential, uses them to connect and validate to an LDAP server, and creates a Subject object as a result of the authentication. The implementation might use JAAS login modules. A Subject object is returned as a result of authentication.

Notice that this method creates two exceptions: InvalidCredentialException and ExpiredCredentialException. The InvalidCredentialException exception indicates that the credential is not valid. The ExpiredCredentialException exception indicates that the credential expired. If one of these two exceptions result from the authenticate method, the exceptions are sent back to the client. However, the client runtime handles these two exceptions differently:

- If the error is an InvalidCredentialException exception, the client run time displays this exception. Your application must handle the exception. You can correct the CredentialGenerator, for example, and then try the operation again.
- If the error is an ExpiredCredentialException exception, and the retry count is not 0, the client run time calls the CredentialGenerator.getCredential method again, and sends the new Credential object to the server. If the new credential authentication succeeds, the server processes the request. If the new credential authentication fails, the exception is sent back to the client. If the number of authentication retry attempts reaches the supported value and the client still gets an ExpiredCredentialException exception, the ExpiredCredentialException exception results. Your application must handle the error.

The Authenticator interface provides great flexibility. You can implement the Authenticator interface in your own specific way. For example, you can implement this interface to support two different user registries.

WebSphere eXtreme Scale provides sample authenticator plug-in implementations. Except for the WebSphere Application Server authenticator plug-in, the other implementations are only samples for testing purposes.

KeyStoreLoginAuthenticator

This example uses an eXtreme Scale built-in implementation: KeyStoreLoginAuthenticator, which is for testing and sample purposes (a keystore is a simple user registry and should not be used for a production environment). Again, the class is displayed to further demonstrate how to implement an authenticator.

Attention: In the following example, some lines of code are continued on the next line for publication purposes.

```
KeyStoreLoginAuthenticator.java
// 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-J34 © COPYRIGHT International Business Machines Corp. 2007

package com.ibm.websphere.objectgrid.security.plugins.builtins;

import javax.security.auth.Subject;
import javax.security.auth.login.LoginContext;
import javax.security.auth.login.LoginException;

import com.ibm.websphere.objectgrid.security.plugins.Authenticator;
import com.ibm.websphere.objectgrid.security.plugins.Credential;
import com.ibm.websphere.objectgrid.security.plugins.ExpiredCredentialException;
import com.ibm.websphere.objectgrid.security.plugins.InvalidCredentialException;
import com.ibm.ws.objectgrid.Constants;
import com.ibm.ws.objectgrid.ObjectGridManagerImpl;
import com.ibm.ws.objectgrid.security.auth.callback.UserPasswordCallbackHandlerImpl;

/**
 * This class is an implementation of the <code>Authenticator</code> interface
 * when a user name and password are used as a credential.
 * <p>
 * When user ID and password authentication is used, the credential passed to the
 * <code>authenticate(Credential)</code> method is a UserPasswordCredential object.
 * <p>
 * This implementation will use a <code>KeyStoreLoginModule</code> to authenticate
 * the user into the keystore using the JAAS login module "KeyStoreLogin". The key
 * store can be configured as an option to the <code>KeyStoreLoginModule</code>
 * class. Please see the <code>KeyStoreLoginModule</code> class for more details
 * about how to set up the JAAS login configuration file.
 * <p>
 * This class is only for sample and quick testing purpose. Users should
 * write your own Authenticator implementation which can fit better into
 * the environment.
 *
 * @ibm-api
 * @since WAS XD 6.0.1
 *
 * @see Authenticator
 * @see KeyStoreLoginModule
 * @see UserPasswordCredential
 */
public class KeyStoreLoginAuthenticator implements Authenticator {

    /**
     * Creates a new KeyStoreLoginAuthenticator.
     */
    public KeyStoreLoginAuthenticator() {
        super();
    }

    /**
     * Authenticates a <code>UserPasswordCredential</code>.
     * <p>
     * Uses the user name and password from the specified UserPasswordCredential
     * to login to the KeyStoreLoginModule named "KeyStoreLogin".
     *
     * @throws InvalidCredentialException if credential isn't a
     *         UserPasswordCredential or some error occurs during processing
     *         of the supplied UserPasswordCredential
     *
     * @throws ExpiredCredentialException if credential is expired.  This exception
     *         is not used by this implementation
     *
     * @see Authenticator#authenticate(Credential)
     * @see KeyStoreLoginModule
     */
    public Subject authenticate(Credential credential) throws InvalidCredentialException,
                        ExpiredCredentialException {

        if (credential == null) {
            throw new InvalidCredentialException("Supplied credential is null");
        }

        if (! (credential instanceof UserPasswordCredential) ) {
            throw new InvalidCredentialException("Supplied credential is not a UserPasswordCredential");
        }

        UserPasswordCredential cred = (UserPasswordCredential) credential;
        LoginContext lc = null;
        try {
            lc = new LoginContext("KeyStoreLogin",
                    new UserPasswordCallbackHandlerImpl(cred.getUserName(), cred.getPassword().toCharArray()));

            lc.login();

            Subject subject = lc.getSubject();

            return subject;
        }
        catch (LoginException le) {
            throw new InvalidCredentialException(le);
        }
```

```
        catch (IllegalArgumentException ile) {
            throw new InvalidCredentialException(ile);
        }
    }
}
```

Attention: In the following example, some lines of code are continued on the next line for publication purposes.

```
KeyStoreLoginModule.java
// 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-J34 © COPYRIGHT International Business Machines Corp. 2007
package com.ibm.websphere.objectgrid.security.plugins.builtins;

import java.io.File;
import java.io.FileInputStream;
import java.security.KeyStore;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.UnrecoverableKeyException;
import java.security.cert.Certificate;
import java.security.cert.CertificateException;
import java.security.cert.CertificateFactory;
import java.security.cert.X509Certificate;
import java.util.Arrays;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;

import javax.security.auth.Subject;
import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.NameCallback;
import javax.security.auth.callback.PasswordCallback;
import javax.security.auth.login.LoginException;
import javax.security.auth.spi.LoginModule;
import javax.security.auth.x500.X500Principal;
import javax.security.auth.x500.X500PrivateCredential;

import com.ibm.websphere.objectgrid.ObjectGridRuntimeException;
import com.ibm.ws.objectgrid.Constants;
import com.ibm.ws.objectgrid.ObjectGridManagerImpl;
import com.ibm.ws.objectgrid.util.ObjectGridUtil;

/**
 * A KeyStoreLoginModule is keystore authentication login module based on
 * JAAS authentication.
 * <p>
 * A login configuration should provide an option "<code>keyStoreFile</code>" to
 * indicate where the keystore file is located. If the <code>keyStoreFile</code>
 * value contains a system property in the form, <code>${system.property}</code>,
 * it will be expanded to the value of the system property.
 * <p>
 * If an option "<code>keyStoreFile</code>" is not provided, the default keystore
 * file name is <code>"${java.home}${/}.keystore"</code>.
 * <p>
 * Here is a Login module configuration example:
 * <pre><code>
 *    KeyStoreLogin {
 *        com.ibm.websphere.objectgrid.security.plugins.builtins.KeystoreLoginModule required
 *            keyStoreFile="${user.dir}${/}security${/}.keystore";
 *    };
 * </code></pre>
 *
 * @ibm-api
 * @since WAS XD 6.0.1
 *
 * @see LoginModule
 */
public class KeyStoreLoginModule implements LoginModule {

    private static final String CLASS_NAME = KeyStoreLoginModule.class.getName();

    /**
     * keystore file property name
     */
    public static final String KEY_STORE_FILE_PROPERTY_NAME = "keyStoreFile";

    /**
     * keystore type. Only JKS is supported
     */
    public static final String KEYSTORE_TYPE = "JKS";

    /**
     * The default keystore file name
```

```
     */
    public static final String DEFAULT_KEY_STORE_FILE = "${java.home}${/}.keystore";

    private CallbackHandler handler;

    private Subject subject;

    private boolean debug = false;

    private Set principals = new HashSet();

    private Set publicCreds = new HashSet();

    private Set privateCreds = new HashSet();

    protected KeyStore keyStore;

    /**
     * Creates a new KeyStoreLoginModule.
     */
    public KeyStoreLoginModule() {
    }

    /**
     * Initializes the login module.
     *
     * @see LoginModule#initialize(Subject, CallbackHandler, Map, Map)
     */
    public void initialize(Subject sub, CallbackHandler callbackHandler,
            Map mapSharedState, Map mapOptions) {

        // initialize any configured options
        debug = "true".equalsIgnoreCase((String) mapOptions.get("debug"));

        if (sub == null)
            throw new IllegalArgumentException("Subject is not specified");

        if (callbackHandler == null)
            throw new IllegalArgumentException(
            "CallbackHander is not specified");

        // Get the keystore path
        String sKeyStorePath = (String) mapOptions
            .get(KEY_STORE_FILE_PROPERTY_NAME);

        // If there is no keystore path, the default one is the .keystore
        // file in the java home directory
        if (sKeyStorePath == null) {
            sKeyStorePath = DEFAULT_KEY_STORE_FILE;
        }

        // Replace the system enviroment variable
        sKeyStorePath = ObjectGridUtil.replaceVar(sKeyStorePath);

        File fileKeyStore = new File(sKeyStorePath);

        try {
            KeyStore store = KeyStore.getInstance("JKS");
            store.load(new FileInputStream(fileKeyStore), null);

            // Save the keystore
            keyStore = store;

            if (debug) {
                System.out.println("[KeyStoreLoginModule] initialize: Successfully loaded keystore");
            }
        }
        catch (Exception e) {
            ObjectGridRuntimeException re = new ObjectGridRuntimeException(
                    "Failed to load keystore: " + fileKeyStore.getAbsolutePath());
            re.initCause(e);
            if (debug) {
                System.out.println("[KeyStoreLoginModule] initialize: keystore loading failed with exception "
                        + e.getMessage());
            }
        }

        this.subject = sub;
        this.handler = callbackHandler;
    }

    /**
     * Authenticates a user based on the keystore file.
     *
     * @see LoginModule#login()
     */
    public boolean login() throws LoginException {

        if (debug) {
            System.out.println("[KeyStoreLoginModule] login: entry");
        }
```

```
        String name = null;
        char pwd[] = null;

        if (keyStore == null || subject == null || handler == null) {
            throw new LoginException("Module initialization failed");
        }

        NameCallback nameCallback = new NameCallback("Username:");
        PasswordCallback pwdCallback = new PasswordCallback("Password:", false);

        try {
            handler.handle(new Callback[] { nameCallback, pwdCallback });
        }
        catch (Exception e) {
            throw new LoginException("Callback failed: " + e);
        }

        name = nameCallback.getName();
        char[] tempPwd = pwdCallback.getPassword();

        if (tempPwd == null) {
            // treat a NULL password as an empty password
            tempPwd = new char[0];
        }
        pwd = new char[tempPwd.length];
        System.arraycopy(tempPwd, 0, pwd, 0, tempPwd.length);

        pwdCallback.clearPassword();

        if (debug) {
            System.out.println("[KeyStoreLoginModule] login: "
                    + "user entered user name: " + name);
        }

        // Validate the user name and password
        try {
            validate(name, pwd);
        }
        catch (SecurityException se) {
            principals.clear();
            publicCreds.clear();
            privateCreds.clear();
            LoginException le = new LoginException(
            "Exception encountered during login");
            le.initCause(se);

            throw le;
        }

        if (debug) {
            System.out.println("[KeyStoreLoginModule] login: exit");
        }
        return true;
    }

    /**
     * Indicates the user is accepted.
     * <p>
     * This method is called only if the user is authenticated by all modules in
     * the login configuration file. The principal objects will be added to the
     * stored subject.
     *
     * @return false if for some reason the principals cannot be added; true
     *         otherwise
     *
     * @exception LoginException
     *                LoginException is thrown if the subject is readonly or if
     *                any unrecoverable exceptions is encountered.
     *
     * @see LoginModule#commit()
     */
    public boolean commit() throws LoginException {
        if (debug) {
            System.out.println("[KeyStoreLoginModule] commit: entry");
        }

        if (principals.isEmpty()) {
            throw new IllegalStateException("Commit is called out of sequence");
        }

        if (subject.isReadOnly()) {
            throw new LoginException("Subject is Readonly");
        }

        subject.getPrincipals().addAll(principals);
        subject.getPublicCredentials().addAll(publicCreds);
        subject.getPrivateCredentials().addAll(privateCreds);

        principals.clear();
        publicCreds.clear();
```

```
        privateCreds.clear();

        if (debug) {
            System.out.println("[KeyStoreLoginModule] commit: exit");
        }
        return true;
    }

    /**
     * Indicates the user is not accepted
     *
     * @see LoginModule#abort()
     */
    public boolean abort() throws LoginException {
        boolean b = logout();
        return b;
    }

    /**
     * Logs the user out. Clear all the maps.
     *
     * @see LoginModule#logout()
     */
    public boolean logout() throws LoginException {


        // Clear the instance variables
        principals.clear();
        publicCreds.clear();
        privateCreds.clear();

        // clear maps in the subject
        if (!subject.isReadOnly()) {
            if (subject.getPrincipals() != null) {
                subject.getPrincipals().clear();
            }

            if (subject.getPublicCredentials() != null) {
                subject.getPublicCredentials().clear();
            }

            if (subject.getPrivateCredentials() != null) {
                subject.getPrivateCredentials().clear();
            }
        }
        return true;
    }

    /**
     * Validates the user name and password based on the keystore.
     *
     * @param userName user name
     * @param password password
     * @throws SecurityException if any exceptions encountered
     */
    private void validate(String userName, char password[])
        throws SecurityException {

        PrivateKey privateKey = null;

        // Get the private key from the keystore
        try {
            privateKey = (PrivateKey) keyStore.getKey(userName, password);
        }
        catch (NoSuchAlgorithmException nsae) {
            SecurityException se = new SecurityException();
            se.initCause(nsae);
            throw se;
        }
        catch (KeyStoreException kse) {
            SecurityException se = new SecurityException();
            se.initCause(kse);
            throw se;
        }
        catch (UnrecoverableKeyException uke) {
            SecurityException se = new SecurityException();
            se.initCause(uke);
            throw se;
        }

        if (privateKey == null) {
            throw new SecurityException("Invalid name: " + userName);
        }

        // Check the certificats
        Certificate certs[] = null;
        try {
            certs = keyStore.getCertificateChain(userName);
        }
        catch (KeyStoreException kse) {
            SecurityException se = new SecurityException();
```

```
            se.initCause(kse);
            throw se;
        }

        if (debug) {
            System.out.println("  Print out the certificates:");
            for (int i = 0; i < certs.length; i++) {
                System.out.println("  certificate " + i);
                System.out.println("    " + certs[i]);
            }
        }

        if (certs != null && certs.length > 0) {

            // If the first certificate is an X509Certificate
            if (certs[0] instanceof X509Certificate) {
                try {
                    // Get the first certificate which represents the user
                    X509Certificate certX509 = (X509Certificate) certs[0];

                    // Create a principal
                    X500Principal principal = new X500Principal(certX509
                            .getIssuerDN()
                            .getName());
                    principals.add(principal);

                    if (debug) {
                        System.out.println("  Principal added: " + principal);
                    }
                    // Create the certification path object and add it to the
                    // public credential set
                    CertificateFactory factory = CertificateFactory
                        .getInstance("X.509");
                    java.security.cert.CertPath certPath = factory
                        .generateCertPath(Arrays.asList(certs));
                    publicCreds.add(certPath);

                    // Add the private credential to the private credential set
                    privateCreds.add(new X500PrivateCredential(certX509,
                            privateKey, userName));

                }
                catch (CertificateException ce) {
                    SecurityException se = new SecurityException();
                    se.initCause(ce);
                    throw se;
                }
            }
            else {
                // The first certificate is not an X509Certificate
                // We just add the certificate to the public credential set
                // and the private key to the private credential set.
                publicCreds.add(certs[0]);
                privateCreds.add(privateKey);
            }
        }
    }
}
```

Using the LDAP authenticator plug-in

You are provided with the com.ibm.websphere.objectgrid.security.plugins.builtins.LDAPAuthenticator default implementation to handle the user name and password authentication to an LDAP server. This implementation uses the LDAPLogin login module to log the user into a Lightweight Directory Access Protocol (LDAP) server.The following snippet demonstrates how the authenticate method is implemented:

Attention: In the following example, some lines of code are continued on the next line for publication purposes.

```
/**
* @see com.ibm.ws.objectgrid.security.plugins.Authenticator#
* authenticate(LDAPLogin)
*/
public Subject authenticate(Credential credential) throws
InvalidCredentialException, ExpiredCredentialException {

    UserPasswordCredential cred = (UserPasswordCredential) credential;
    LoginContext lc = null;
    try {
        lc = new LoginContext("LDAPLogin",
            new UserPasswordCallbackHandlerImpl(cred.getUserName(),
            cred.getPassword().toCharArray()));

        lc.login();

        Subject subject = lc.getSubject();

        return subject;
    }
    catch (LoginException le) {
        throw new InvalidCredentialException(le);
    }
```

```
    catch (IllegalArgumentException ile) {
        throw new InvalidCredentialException(ile);
    }
}
```

Also, eXtreme Scale ships a login module com.ibm.websphere.objectgrid.security.plugins.builtins.LDAPLoginModule for this purpose. You must provide the following two options in the JAAS login configuration file.

- providerURL: The LDAP server provider URL
- factoryClass: The LDAP context factory implementation class

The LDAPLoginModule module calls the com.ibm.websphere.objectgrid.security.plugins.builtins.LDAPAuthentcationHelper.authenticate method. The following code snippet shows how you can implement the authenticate method of the LDAPAuthenticationHelper.

```
/**
* Authenticate the user to the LDAP directory.
* @param user the user ID, e.g., uid=xxxxxx,c=us,ou=bluepages,o=ibm.com
* @param pwd the password
*
* @throws NamingException
*/
public String[] authenticate(String user, String pwd)
throws NamingException {
    Hashtable env = new Hashtable();
    env.put(Context.INITIAL_CONTEXT_FACTORY, factoryClass);
    env.put(Context.PROVIDER_URL, providerURL);
    env.put(Context.SECURITY_PRINCIPAL, user);
    env.put(Context.SECURITY_CREDENTIALS, pwd);
    env.put(Context.SECURITY_AUTHENTICATION, "simple");

    InitialContext initialContext = new InitialContext(env);

    // Look up for the user
    DirContext dirCtx = (DirContext) initialContext.lookup(user);

    String uid = null;
    int iComma = user.indexOf(",");
    int iEqual = user.indexOf("=");
    if (iComma > 0 && iComma > 0) {
        uid = user.substring(iEqual + 1, iComma);
    }
    else {
        uid = user;
    }

    Attributes attributes = dirCtx.getAttributes("");

    // Check the UID
    String thisUID = (String) (attributes.get(UID).get());

    String thisDept = (String) (attributes.get(HR_DEPT).get());

    if (thisUID.equals(uid)) {
        return new String[] { thisUID, thisDept };
    }
    else {
        return null;
    }
}
```

If authentication succeeds, the ID and password are considered valid. Then the login module gets the ID information and department information from this authenticate method. The login module creates two principals: SimpleUserPrincipal and SimpleDeptPrincipal. You can use the authenticated subject for group authorization (in this case, the department is a group) and individual authorization.

The following example shows a login module configuration that is used to log in to the LDAP server:

```
LDAPLogin { com.ibm.websphere.objectgrid.security.plugins.builtins.LDAPLoginModule required
    providerURL="ldap://directory.acme.com:389/"
    factoryClass="com.sun.jndi.ldap.LdapCtxFactory";
};
```

In the previous configuration, the LDAP server points to the ldap://directory.acme.com:389/server. Change this setting to your LDAP server. This login module uses the provided ID and password to connect to the LDAP server. This implementation is for testing purposes only.

Using the WebSphere Application Server authenticator plug-in

Also, eXtreme Scale provides the com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenAuthenticator built-in implementation to use the WebSphere Application Server security infrastructure. This built-in implementation can be used when the following conditions are true.

1. WebSphere Application Server global security is turned on.
2. All eXtreme Scale clients and servers are launched in WebSphere Application Server JVMs.
3. These application servers are in the same security domain.
4. The eXtreme Scale client is already authenticated in WebSphere Application Server.

The client can use the com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredentialGenerator class to generate a credential. The server uses this Authenticator implementation class to authenticate the credential. If the token is authenticated successfully, a Subject object returns.

This scenario takes advantage of the fact that the client has already been authenticated. Because the application servers that have the servers are in the same security domain as the application servers that house the clients, the security tokens can be propagated from the client to the server so that the same user registry does not need to be authenticated again.

Using the Tivoli® Access Manager authenticator plug-in

Tivoli Access Manager is used widely as a security server. You can also implement Authenticator using the Tivoli Access Manager's provided login modules.

To authenticate a user for Tivoli Access Manager, apply the the com.tivoli.mts.PDLoginModule login module, which requires that the calling application provide the following information:

1. A principal name, specified as either a short name or an X.500 name (DN)
2. A password

The login module authenticates the principal and returns the Tivoli Access Manager credential. The login module expects the calling application to provide the following information:

1. The user name, through a javax.security.auth.callback.NameCallback object.
2. The password, through a javax.security.auth.callback.PasswordCallback object.

When the Tivoli Access Manager credential is successfully retrieved, the JAAS LoginModule creates a Subject and a PDPrincipal. No built-in for Tivoli Access Manager authentication is provided, because it is just with the PDLoginModule module. See the IBM® Tivoli Access Manager Authorization Java Classes Developer Reference for more details.

Connecting to WebSphere eXtreme Scale securely

To connect an eXtreme Scale client to a server securely, you can use any connect method in the ObjectGridManager interface which takes a ClientSecurityConfiguration object. The following is a brief example.

```
public ClientClusterContext connect(String catalogServerEndpoints,
                 ClientSecurityConfiguration securityProps,
     URL overRideObjectGridXml) throws ConnectException;
```

This method takes a parameter of the ClientSecurityConfiguration type, which is an interface representing a client security configuration. You can use com.ibm.websphere.objectgrid.security.config.ClientSecurityConfigurationFactory public API to create an instance with default values, or you can create an instance by passing the WebSphere eXtreme Scale client property file. This file contains the following properties that are related to authentication. The value marked with a plus sign (+) is the default.

- `securityEnabled (true, false+)`: This property indicates if security is enabled. When a client connects to a server, the securityEnabled value on the client and server side must be both `true` or both `false`. For example, if the connected server security is enabled, the client has to set this property to true to connect to the server.
- authenticationRetryCount (an integer value, 0+): This property determines how many retries are attempted for login when a credential is expired. If the value is 0, no retries are attempted. The authentication retry only applies to the case when the credential is expired. If the credential is not valid, there is no retry. Your application is responsible for trying the operation again.

After you create a com.ibm.websphere.objectgrid.security.config.ClientSecurityConfiguration object, set the credentialGenerator object on the client using the following method:

```
/**
* Set the {@link CredentialGenerator} object for this client.
* @param generator the CredentialGenerator object associated with this client
*/
void setCredentialGenerator(CredentialGenerator generator);
```

You can set the CredentialGenerator object in the WebSphere eXtreme Scale client property file too, as follows.

- credentialGeneratorClass: The class implementation name for the CredentialGenerator object. It must have a default constructor.
- credentialGeneratorProps: The properties for the CredentialGenerator class. If the value is not null, it is set to the constructed CredentialGenerator object using the setProperties(String) method.

Here is a sample to instantiate a ClientSecurityConfiguration and then use it to connect to the server.

```
/**
* Get a secure ClientClusterContext
* @return a secure ClientClusterContext object
*/
protected ClientClusterContext connect() throws ConnectException {
ClientSecurityConfiguration csConfig = ClientSecurityConfigurationFactory
.getClientSecurityConfiguration("/properties/security.ogclient.props");

UserPasswordCredentialGenerator gen= new
UserPasswordCredentialGenerator("manager", "manager1");

csConfig.setCredentialGenerator(gen);

return objectGridManager.connect(csConfig, null);
}
```

When the connect is called, the WebSphere eXtreme Scale client calls the CredentialGenerator.getCredential method to get the client credential. This credential is sent along with the connect request to the server for authentication.

Using a different CredentialGenerator instance per session

In some cases, a WebSphere eXtreme Scale client represents just one client identity, but in others, it might represent multiple identities. Here is one scenario for the latter case: An WebSphere eXtreme Scale client is created and shared in a Web server. All servlets in this Web server use this one WebSphere eXtreme Scale client. Because every servlet represents a different Web client, use different credentials when sending requests to WebSphere eXtreme Scale servers.

WebSphere eXtreme Scale provides for changing the credential on the session level. Every session can uses a different CredentialGenerator object. Therefore, the previous scenarios can be implemented by letting the servlet get a session with a different CredentialGenerator object. The following example illustrates the ObjectGrid.getSession(CredentialGenerator) method in the ObjectGridManager interface.

Attention: In the following example, some lines of code are continued on the next line for publication purposes.

```
/**
     * Get a session using a <code>CredentialGenerator</code>.
     * <p>
     * This method can only be called by the ObjectGrid client in an ObjectGrid
     * client server environment. If ObjectGrid is used in a local model, that is,
     * within the same JVM with no client or server existing, <code>getSession(Subject)</code>
     * or the <code>SubjectSource</code> plugin should be used to secure the ObjectGrid.
     *
     * <p>If the <code>initialize()</code> method has not been invoked prior to
     * the first <code>getSession</code> invocation, an implicit initialization
     * will occur.  This ensures that all of the configuration is complete
     * before any runtime usage is required.</p>
     *
     * @param credGen A <code>CredentialGenerator</code> for generating a credential
     *                for the session returned.
     *
     * @return An instance of <code>Session</code>
     *
     * @throws ObjectGridException if an error occurs during processing
     * @throws TransactionCallbackException if the <code>TransactionCallback</code>
     *         throws an exception
     * @throws IllegalStateException if this method is called after the
     *         <code>destroy()</code> method is called.
     *
     * @see #destroy()
     * @see #initialize()
     * @see CredentialGenerator
     * @see Session
     * @since WAS XD 6.0.1
 */
Session getSession(CredentialGenerator credGen) throws
ObjectGridException, TransactionCallbackException;
```

The following is an example:

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

CredentialGenerator credGenManager = new UserPasswordCredentialGenerator("manager", "xxxxxx");
CredentialGenerator credGenEmployee = new UserPasswordCredentialGenerator("employee", "xxxxxx");

ObjectGrid og = ogManager.getObjectGrid(ctx, "accounting");

// Get a session with CredentialGenerator;
Session session = og.getSession(credGenManager );

// Get the employee map
ObjectMap om = session.getMap("employee");

// start a transaction.
session.begin();

Object rec1 = map.get("xxxxxx");

session.commit();

// Get another session with a different CredentialGenerator;
session = og.getSession(credGenEmployee );

// Get the employee map
om = session.getMap("employee");

// start a transaction.
session.begin();

Object rec2 = map.get("xxxxx");

session.commit();
```

If you use the ObjectGird.getSession method to get a Session object, the session uses the CredentialGenerator object set on the ClientConfigurationSecurity object. The ObjectGrid.getSession(CredentialGenerator) method overrides the CredentialGenerator set in the ClientSecurityConfiguration object.

If you can reuse the Session object, a performance gain results. However, calling the ObjectGrid.getSession(CredentialGenerator) method is not very expensive. The major overhead is the increased object garbage collection time. Make sure that you release the references after you are done with the Session objects. Generally, if your Session object can share the identity, try to reuse the Session object. If not, use the ObjectGrid.getSession(CredentialGenerator) method.

Related information:
Credential API

Client authorization programming

WebSphere® eXtreme Scale supports Java™ Authentication and Authorization Service (JAAS) authorization that is ready to use and also supports custom authorization using the ObjectGridAuthorization interface.

The ObjectGridAuthorization plug-in is used to authorize ObjectGrid, ObjectMap, and JavaMap accesses to the Principals represented by a Subject object in a custom way. A typical implementation of this plug-in is to retrieve the Principals from the Subject object, and then check whether the specified permissions are granted to the Principals.

A permission passed to the checkPermission(Subject, Permission) method can be one of the following permissions:

- MapPermission
- ObjectGridPermission
- ServerMapPermission
- AgentPermission

Refer to ObjectGridAuthorization API documentation for more details.

MapPermission

The com.ibm.websphere.objectgrid.security.MapPermission public class represents permissions to the ObjectGrid resources, specifically the methods of ObjectMap or JavaMap interfaces. WebSphere eXtreme Scale defines the following permission strings to access the methods of ObjectMap and JavaMap:

- **read:** Permission to read the data from the map. The integer constant is defined as `MapPermission.READ`.
- **write:** Permission to update the data in the map. The integer constant is defined as `MapPermission.WRITE`.
- **insert:** Permission to insert the data into the map. The integer constant is defined as `MapPermission.INSERT`.
- **remove:** Permission to remove the data from the map. The integer constant is defined as `MapPermission.REMOVE`.
- **invalidate:** Permission to invalidate the data from the map. The integer constant is defined as `MapPermission.INVALIDATE`.
- **all:** All above permissions: read, write, insert, remote, and invalidate. The integer constant is defined as `MapPermission.ALL`.

Refer to MapPermission API documentation for more details.

You can construct a MapPermission object by passing the fully qualified ObjectGrid map name (in format [ObjectGrid_name].[ObjectMap_name]) and the permission string or integer value. A permission string can be a comma-delimited string of the previous permission strings such as read, insert, or it can be all. A permission integer value can be any previously mentioned permission integer constants or a mathematical value of several integer permission constants, such as MapPermission.READ|MapPermission.WRITE.

The authorization occurs when an ObjectMap or JavaMap method is called. The run time checks different permissions for different methods. If the required permissions are not granted to the client, an AccessControlException results.

Table 1. List of methods and the required MapPermission

Permission	ObjectMap/JavaMap
read	Boolean containsKey(Object)
	Boolean equals(Object)
	Object get(Object)
	Object get(Object, Serializable)
	List getAll(List)
	List getAll(List keyList, Serializable)
	List getAllForUpdate(List)
	List getAllForUpdate(List, Serializable)
	Object getForUpdate(Object)
	Object getForUpdate(Object, Serializable)
	public Object getNextKey(long)
write	Object put(Object key, Object value)
	void put(Object, Object, Serializable)
	void putAll(Map)
	void putAll(Map, Serializable)
	void update(Object, Object)
	void update(Object, Object, Serializable)
insert	public void insert (Object, Object)
	void insert(Object, Object, Serializable)
remove	Object remove (Object)
	void removeAll(Collection)

Permission	ObjectMap/JavaMap
	void clear()
invalidate	public void invalidate (Object, Boolean)
	void invalidateAll(Collection, Boolean)
	void invalidateUsingKeyword(Serializable)
	int setTimeToLive(int)

Authorization is based solely on which method is used, rather than what the method really does. For example, a put method can insert or update a record based on whether the record exists. However, the insert or update cases are not distinguished.

An operation type can be achieved by combinations of other types. For example, an update can be achieved by a remove and then an insert. Consider these combinations when designing your authorization policies.

ObjectGridPermission

A com.ibm.websphere.objectgrid.security.ObjectGridPermission represents permissions to the ObjectGrid:

- Query: permission to create an object query or entity query. The integer constant is defined as ObjectGridPermission.QUERY.
- Dynamic map: permission to create a dynamic map based on the map template. The integer constant is defined as ObjectGridPermission.DYNAMIC_MAP.

Refer to ObjectGridPermission API documentation for more details.
The following table summarizes the methods and the required ObjectGridPermission:

Table 2. List of methods and the required ObjectGridPermission

Permission action	Methods
`query`	com.ibm.websphere.objectgrid.Session.createObjectQuery(String)
`query`	com.ibm.websphere.objectgrid.em.EntityManager.createQuery(String)
`dynamicmap`	com.ibm.websphere.objectgrid.Session.getMap(String)

ServerMapPermission

An ServerMapPermission represents permissions to an ObjectMap hosted in a server. The name of the permission is the full name of the ObjectGrid map name. It has the following actions:

- **replicate**: permission to replicate a server map to near cache
- **dynamicIndex**: permission for a client to create or remove a dynamic index on a server

Refer to ServerMapPermission API documentation for more details. The detailed methods, which require different ServerMapPermission, are listed in the following table:

Table 3. Permissions to a server-hosted ObjectMap

Permission action	Methods
`replicate`	com.ibm.websphere.objectgrid.ClientReplicableMap.enableClientReplication(Mode, int[], ReplicationMapListener)
`dynamicIndex`	com.ibm.websphere.objectgrid.BackingMap.createDynamicIndex(String, Boolean, String, DynamicIndexCallback)
`dynamicIndex`	com.ibm.websphere.objectgrid.BackingMap.removeDynamicIndex(String)

AgentPermission

An AgentPermission represents permissions to the datagrid agents. The name of the permission is the full name of the ObjectGrid map, and the action is a comma-delimited string of agent implementation class names or package names.

Refer to AgentPermission API documentation for more information.

The following methods in the class com.ibm.websphere.objectgrid.datagrid.AgentManager require AgentPermission.

`com.ibm.websphere.objectgrid.datagrid.AgentManager#callMapAgent(MapGridAgent, Collection)`

`com.ibm.websphere.objectgrid.datagrid.AgentManager#callMapAgent(MapGridAgent)`

`com.ibm.websphere.objectgrid.datagrid.AgentManager#callReduceAgent(ReduceGridAgent, Collection)`

`com.ibm.websphere.objectgrid.datagrid.AgentManager#callReduceAgent(ReduceGridAgent, Collection)`

Authorization mechanisms

WebSphere eXtreme Scale supports two kinds of authorization mechanisms: Java Authentication and Authorization Service (JAAS) authorization and custom authorization. These mechanisms apply to all authorizations. JAAS authorization augments the Java security policies with user-centric access controls. Permissions can be granted based not just on what code is running, but also on who is running it. JAAS authorization is part of the SDK Version 5 and later.

Additionally, WebSphere eXtreme Scale also supports custom authorization with the following plug-in:

- ObjectGridAuthorization: custom way to authorize access to all artifacts.

You can implement your own authorization mechanism if you do not want to use JAAS authorization. By using a custom authorization mechanism, you can use the policy database, policy server, or Tivoli® Access Manager to manage the authorizations.

You can configure the authorization mechanism in two ways:

- XML configuration
  You can use the ObjectGrid XML file to define an ObjectGrid and set the authorization mechanism to either AUTHORIZATION_MECHANISM_JAAS or AUTHORIZATION_MECHANISM_CUSTOM. Here is the secure-objectgrid-definition.xml file that is used in the enterprise application ObjectGridSample:

  ```
  <objectGrids>
          <objectGrid name="secureClusterObjectGrid" securityEnabled="true"
                  authorizationMechanism="AUTHORIZATION_MECHANISM_JAAS">
                  <bean
  id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxsprgauthor_TransactionCallback
  "
  classname="com.ibm.websphere.samples.objectgrid.HeapTransactionCallback" />
  ...
  </objectGrids>
  ```

- Programmatic configuration
  If you want to create an ObjectGrid using method ObjectGrid.setAuthorizationMechanism(int), you can call the following method to set the authorization mechanism. Calling this method applies only to the local WebSphere eXtreme Scale programming model when you directly instantiate the ObjectGrid instance:

  ```
  /**
   * Set the authorization Mechanism. The default is
   * com.ibm.websphere.objectgrid.security.SecurityConstants.
   * AUTHORIZATION_MECHANISM_JAAS.
   * @param authMechanism the map authorization mechanism
  */
  void setAuthorizationMechanism(int authMechanism);
  ```

JAAS authorization

A javax.security.auth.Subject object represents an authenticated user. A Subject consists of a set of principals, and each Principal represents an identity for that user. For example, a Subject can have a name principal, for example, Joe Smith, and a group principal, for example, manager.

Using the JAAS authorization policy, permissions can be granted to specific Principals. WebSphere eXtreme Scale associates the Subject with the current access control context. For each call to the ObjectMap or Javamap method, the Java runtime automatically determines if the policy grants the required permission only to a specific Principal and if so, the operation is allowed only if the Subject associated with the access control context contains the designated Principal.

You must be familiar with the policy syntax of the policy file. For detailed description of JAAS authorization, refer to the JAAS Reference Guide.

WebSphere eXtreme Scale has a special code base that is used for checking the JAAS authorization to the ObjectMap and JavaMap method calls. This special code base is `http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction`. Use this code base when granting ObjectMap or JavaMap permissions to principals. This special code was created because the Java archive (JAR) file for eXtreme Scale is granted with all permissions.

The template of the policy to grant the MapPermission permission is:

```
grant codeBase "http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction"
   <Principal field(s)>{
    permission com.ibm.websphere.objectgrid.security.MapPermission
               "[ObjectGrid_name].[ObjectMap_name]", "action";
    ....
    permission com.ibm.websphere.objectgrid.security.MapPermission
               "[ObjectGrid_name].[ObjectMap_name]", "action";
  };
```

A Principal field looks like the following example:

```
principal Principal_class "principal_name"
```

In this policy, only insert and read permissions are granted to these four maps to a certain principal. The other policy file, fullAccessAuth.policy, grants all permissions to these maps to a principal. Before running the application, change the principal_name and principal class to appropriate values. The value of the principal_name depends on the user registry. For example, if local OS is used as user registry, the machine name is MACH1, the user ID is user1, and the principal_name is `MACH1/user1`.

The JAAS authorization policy can be put directly into the Java policy file, or it can be put in a separate JAAS authorization file and then set in either of two ways:

- Use the following JVM argument:

  ```
  -Djava.security.policy=file:[JAAS_AUTH_POLICY_FILE]
  ```

- Use the following property in the java.security file:

  ```
  -Dauth.policy.url.x=file:[JAAS_AUTH_POLICY_FILE]
  ```

Custom ObjectGrid authorization

ObjectGridAuthorization plug-in is used to authorize ObjectGrid, ObjectMap, and JavaMap accesses to the Principals represented by a Subject object in a custom way. A typical implementation of this plug-in is to retrieve the Principals from the Subject object, and then check whether or not the specified permissions are granted to the Principals.

A permission passed to the checkPermission(Subject, Permission) method could be one of the following:

- MapPermission
- ObjectGridPermission
- AgentPermission
- ServerMapPermission

Refer to ObjectGridAuthorization API documentation for more details.

The ObjectGridAuthorization plug-in can be configured in the following ways:

- XML configuration
  You can use the ObjectGrid XML file to define an ObjectAuthorization plug-in. Here is an example:

```
<objectGrids>
        <objectGrid name="secureClusterObjectGrid" securityEnabled="true"
                authorizationMechanism="AUTHORIZATION_MECHANISM_CUSTOM">
...
                <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxsprgauthor_ObjectGridAuthoriza
tion"
className="com.acme.ObjectGridAuthorizationImpl" />
</objectGrids>
```

- Programmatic configuration
  If you want to create an ObjectGrid using the API method ObjectGrid.setObjectGridAuthorization(ObjectGridAuthorization), you can call the following method to set the authorization plug-in. This method only applies to the local eXtreme Scale programming model when you directly instantiate the ObjectGrid instance.

  Attention: In the following example, some lines of code are continued on the next line for publication purposes.

```
/**
     * Sets the <code>ObjectGridAuthorization</code> for this ObjectGrid instance.
     * <p>
     * Passing <code>null</code> to this method removes a previously set
     * <code>ObjectGridAuthorization</code> object from an earlier invocation of this method
     * and indicates that this <code>ObjectGrid</code> is not associated with a
     * <code>ObjectGridAuthorization</code> object.
     * <p>
     * This method should only be used when ObjectGrid security is enabled. If
     * the ObjectGrid security is disabled, the provided <code>ObjectGridAuthorization</code> object
     * will not be used.
     * <p>
     * A <code>ObjectGridAuthorization</code> plug-in can be used to authorize
     * access to the ObjectGrid and maps. Please refer to <code>ObjectGridAuthorization</code> for more details.
     *
     * <p>
     * As of XD 6.1, the <code>setMapAuthorization</code> is deprecated and
     * <code>setObjectGridAuthorization</code> is recommended for use. However,
     * if both <code>MapAuthorization</code> plug-in and <code>ObjectGridAuthorization</code> plug-in
     * are used, ObjectGrid will use the provided <code>MapAuthorization</code> to authorize map accesses,
     * even though it is deprecated.
     * <p>
     * Note, to avoid an <code>IllegalStateException</code>, this method must be
     * called prior to the <code>initialize()</code> method.  Also, keep in mind
     * that the <code>getSession</code> methods implicitly call the
     * <code>initialize()</code> method if it has yet to be called by the
     * application.
     *
     * @param ogAuthorization the <code>ObjectGridAuthorization</code> plug-in
     *
     * @throws IllegalStateException if this method is called after the
     *         <code>initialize()</code> method is called.
     *
     * @see #initialize()
     * @see ObjectGridAuthorization
     * @since WAS XD 6.1
     */
    void setObjectGridAuthorization(ObjectGridAuthorization ogAuthorization);
```

Implementing ObjectGridAuthorization

The Boolean checkPermission(Subject subject, Permission permission) method of the ObjectGridAuthorization interface is called by theWebSphere eXtreme Scale run time to check whether the passed-in subject object has the passed-in permission. The implementation of the ObjectGridAuthorization interface returns true if the object has the permission, and false if not.

A typical implementation of this plug-in is to retrieve the principals from the Subject object and check whether the specified permissions are granted to the principals by consulting specific policies. These policies are defined by users. For example, the policies can be defined in a database, a plain file, or a Tivoli Access Manager policy server.

For example, we can use Tivoli Access Manager policy server to manage the authorization policy and use its API to authorize the access. For how to use Tivoli Access Manager Authorization APIs, refer to the IBM® Tivoli Access Manager Authorization Java Classes Developer Reference for more details.

This sample implementation has the following assumptions:

- Check authorization for MapPermission only. For other permissions, always return `true`.
- The Subject object contains a com.tivoli.mts.PDPrincipal principal.

- The Tivoli Access Manager policy server has defined the following permissions for the ObjectMap or JavaMap name object. The object that is defined in the policy server must have the same name as the ObjectMap or JavaMap name in the format of [ObjectGrid_name].[ObjectMap_name]. The permission is the first character of the permission strings that are defined in the MapPermission permission. For example, the permission "r" that is defined in the policy server represents the read permission to the ObjectMap map.

The following code snippet demonstrates how to implement the checkPermission method:

```
/**
* @see com.ibm.websphere.objectgrid.security.plugins.
*               MapAuthorization#checkPermission
* (javax.security.auth.Subject, com.ibm.websphere.objectgrid.security.
*               MapPermission)
*/
public boolean checkPermission(final Subject subject,
        Permission p) {

  // For non-MapPermission, we always authorize.
  if (!(p instanceof MapPermission)){
    return true;
  }

  MapPermission permission = (MapPermission) p;

  String[] str = permission.getParsedNames();

  StringBuffer pdPermissionStr = new StringBuffer(5);
  for (int i=0; i<str.length; i++) {
    pdPermissionStr.append(str[i].substring(0,1));
  }

  PDPermission pdPerm = new PDPermission(permission.getName(),
               pdPermissionStr.toString());

  Set principals = subject.getPrincipals();

  Iterator iter= principals.iterator();
  while(iter.hasNext()) {
    try {
      PDPrincipal principal = (PDPrincipal) iter.next();
      if (principal.implies(pdPerm)) {
        return true;
      }
    }
    catch (ClassCastException cce) {
      // Handle exception
    }
  }
  return false;
}
```

Related information:
Module 4: Use Java Authentication and Authorization Service (JAAS) authorization in WebSphere Application Server

Data grid authentication

You can use the secure token manager plug-in to enable server-to-server authentication, which requires you to implement the SecureTokenManager interface.

The generateToken(Object) method takes an object protect, and then generates a token that cannot be understood by others. The verifyTokens(byte[]) method does the reverse process: it converts the token back to the original object.
A simple SecureTokenManager implementation uses a simple encoding algorithm, such as a XOR algorithm, to encode the object in serialized form and then use corresponding decoding algorithm to decode the token. This implementation is not secure and is easy to break.
WebSphere® eXtreme Scale default implementation
WebSphere eXtreme Scale provides an immediately available implementation for this interface. This default implementation uses a key pair to sign and verify the signature, and uses a secret key to encrypt the content. Every server has a JCEKS type keystore to store the key pair, a private key and public key, and a secret key. The keystore has to be the JCEKS type to store secret keys. These keys are used to encrypt and sign or verify the secret string on the sending end. Also, the token is associated with an expiration time. On the receiving end, the data is verified, decrypted, and compared to the receiver secret string. Secure Sockets Layer (SSL) communication protocols are not required between a pair of servers for authentication because the private keys and public keys serve the same purpose. However, if server communication is not encrypted, the data can be stolen by looking at the communication. Because the token expires soon, the replay attack threat is minimized. This possibility is significantly decreased if all servers are deployed behind a firewall.
The disadvantage of this approach is that the WebSphere eXtreme Scale administrators have to generate keys and transport them to all servers, which can cause security breach during transportation.
Related concepts:
Data grid security
Related tasks:
Authenticating and authorizing clients
Authenticating application clients
Authorizing application clients
Related reference:
Client properties file
Class ClientSecurityConfigurationFactory

Local security programming

WebSphere® eXtreme Scale provides several security endpoints to allow you to integrate custom mechanisms. In the local programming model, the main security function is authorization, and has no authentication support . You must authenticate outside of WebSphere Application Server. However, there are provided plug-ins to obtain and validate Subject objects.

Authentication

In the local programming model, eXtreme Scale does not provide any authentication mechanism, but relies on the environment, either application servers or applications, for authentication. When eXtreme Scale is used in WebSphere Application Server or WebSphere Extended Deployment, applications can use the WebSphere Application Server security authentication mechanism. When eXtreme Scale is running in a Java™ 2 Platform, Standard Edition (J2SE) environment, the application has to manage authentications with Java Authentication and Authorization Service (JAAS) authentication or other authentication mechanisms. For more information about using JAAS authentication, see the JAAS reference guide. The contract between an application and an ObjectGrid instance is the javax.security.auth.Subject object. After the client is authenticated by the application server or the application, the application can retrieve the authenticated javax.security.auth.Subject object and use this Subject object to get a session from the ObjectGrid instance by calling the ObjectGrid.getSession(Subject) method. This Subject object is used to authorize accesses to the map data. This contract is called a subject passing mechanism. The following example illustrates the ObjectGrid.getSession(Subject) API.

```
/**
 * This API allows the cache to use a specific subject rather than the one
 * configured on the ObjectGrid to get a session.
 * @param subject
 * @return An instance of Session
 * @throws ObjectGridException
 * @throws TransactionCallbackException
 * @throws InvalidSubjectException the subject passed in is not valid based
 * on the SubjectValidation mechanism.
 */
public Session getSession(Subject subject)
throws ObjectGridException, TransactionCallbackException, InvalidSubjectException;
```

The ObjectGrid.getSession() method in the ObjectGrid interface can also be used to get a Session object:

```
/**
 * This method returns a Session object that can be used by a single thread at a time.
 * You cannot share this Session object between threads without placing a
 * critical section around it. While the core framework allows the object to move
 * between threads, the TransactionCallback and Loader might prevent this usage,
 * especially in J2EE environments. When security is enabled, this method uses the
 * SubjectSource to get a Subject object.
 *
 * If the initialize method has not been invoked prior to the first
 * getSession invocation, then an implicit initialization occurs.  This
 * initialization ensures that all of the configuration is complete before
 * any runtime usage is required.
 *
 * @see #initialize()
 * @return An instance of Session
 * @throws ObjectGridException
 * @throws TransactionCallbackException
 * @throws IllegalStateException if this method is called after the
 *         destroy() method is called.
 */
public Session getSession()
throws ObjectGridException, TransactionCallbackException;
```

As the API documentation specifies, when security is enabled, this method uses the SubjectSource plug-in to get a Subject object. The SubjectSource plug-in is one of the security plug-ins defined in eXtreme Scale to support propagating Subject objects. See Security-related plug-ins for more information. The getSession(Subject) method can be called on the local ObjectGrid instance only. If you call the getSession(Subject) method on a client side in a distributed eXtreme Scale configuration, an IllegalStateException results.

Security plug-ins

WebSphere eXtreme Scale provides two security plug-ins that are related to the subject passing mechanism: the SubjectSource and SubjectValidation plug-ins.

SubjectSource plug-in

The SubjectSource plug-in, represented by the com.ibm.websphere.objectgrid.security.plugins.SubjectSource interface, is a plug-in that is used to get a Subject object from an eXtreme Scale running environment. This environment can be an application using the ObjectGrid or an application server that hosts the application. Consider the SubjectSource plug-in an alternative to the subject passing mechanism. Using the subject passing mechanism, the application retrieves the Subject object and uses it to get the ObjectGrid session object. With the SubjectSource plug-in, the eXtreme Scale runtime retrieves the Subject object and uses it to get the session object. The subject passing mechanism gives the control of Subject objects to applications, while the SubjectSource plug-in mechanism frees applications from retrieving the Subject object. You can use the SubjectSource plug-in to get a Subject object that represents an eXtreme Scale client that is used for authorization. When the ObjectGrid.getSession method is called, the Subject getSubject throws an ObjectGridSecurityException if security is enabled. WebSphere eXtreme Scale provides a default implementation of this plug-in: com.ibm.websphere.objectgrid.security.plugins.builtins.WSSubjectSourceImpl. This implementation can be used to retrieve a caller subject or a RunAs subject from the thread when an application is running in WebSphere Application Server. You can configure this class in your ObjectGrid descriptor XML file as the SubjectSource implementation class when using eXtreme Scale in WebSphere Application Server. The following code snippet shows the main flow of the WSSubjectSourceImpl.getSubject method.

```
Subject s = null;
try {
  if (finalType == RUN_AS_SUBJECT) {
    // get the RunAs subject
    s = com.ibm.websphere.security.auth.WSSubject.getRunAsSubject();
  }
  else if (finalType == CALLER_SUBJECT) {
    // get the callersubject
    s = com.ibm.websphere.security.auth.WSSubject.getCallerSubject();
  }
}
catch (WSSecurityException wse) {
  throw new ObjectGridSecurityException(wse);
}

return s;
```

For other details, refer to the API documentation for the SubjectSource plug-in and the WSSubjectSourceImpl implementation.

SubjectValidation plug-in

The SubjectValidation plug-in, which is represented by the com.ibm.websphere.objectgrid.security.plugins.SubjectValidation interface, is another security plug-in. The SubjectValidation plug-in can be used to validate that a javax.security.auth.Subject, either passed to the ObjectGrid or retrieved by the SubjectSource plug-in, is a valid Subject that has not been tampered with.

The SubjectValidation.validateSubject(Subject) method in the SubjectValidation interface takes a Subject object and returns a Subject object. Whether a Subject object is considered valid and which Subject object is returned are all up to your implementations. If the Subject object is not valid, an InvalidSubjectException results.

You can use this plug-in if you do not trust the Subject object that is passed to this method. This case is rare considering that you trust the application developers who develop the code to retrieve the Subject object.

An implementation of this plug-in needs support from the Subject object creator because only the creator knows if the Subject object has been tampered with. However, some subject creator might not know if the Subject has been tampered with. In this case, this plug-in is not useful.

WebSphere eXtreme Scale provides a default implementation of SubjectValidation: com.ibm.websphere.objectgrid.security.plugins.builtins.WSSubjectValidationImpl. You can use this implementation to validate the WebSphere Application Server-authenticated subject. You can configure this class as the SubjectValidation implementation class when using eXtreme Scale in WebSphere Application Server. The WSSubjectValidationImpl implementation considers a Subject object valid only if the credential token that is associated with this Subject has not been tampered with. You can change other parts of the Subject object. The WSSubjectValidationImpl implementation asks WebSphere Application Server for the original Subject corresponding to the credential token and returns the original Subject object as the validated Subject object. Therefore, the changes made to the Subject contents other than the credential token have no effects. The following code snippet shows the basic flow of the WSSubjectValidationImpl.validateSubject(Subject).

```
// Create a LoginContext with scheme WSLogin and
// pass a Callback handler.
LoginContext lc = new LoginContext("WSLogin",
new WSCredTokenCallbackHandlerImpl(subject));

// When this method is called, the callback handler methods
// will be called to log the user in.
lc.login();

// Get the subject from the LoginContext
return lc.getSubject();
```

In the previous code snippet, a credential token callback handler object, WSCredTokenCallbackHandlerImpl, is created with the Subject object to validate. Then a LoginContext object is created with the login scheme WSLogin. When the lc.login method is called, WebSphere Application Server security retrieves the credential token from the Subject object and then returns the correspondent Subject as the validated Subject object.
For other details, refer to the Java APIs of SubjectValidation and WSSubjectValidationImpl implementation.

Plug-in configuration

You can configure the SubjectValidation plug-in and SubjectSource plug-in in two ways:

- **XML Configuration**You can use the ObjectGrid XML file to define an ObjectGrid and set these two plug-ins. Here is an example, in which the WSSubjectSourceImpl class is configured as the SubjectSource plug-in and the WSSubjectValidation class is configured as the SubjectValidation plug-in. Attention: In the following example, some lines of code are continued on the next line for publication purposes.

  ```
  <objectGrids>
          <objectGrid name="secureClusterObjectGrid" securityEnabled="true"
                  authorizationMechanism="AUTHORIZATION_MECHANISM_JAAS">
          <bean
  id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxslocalsec_SubjectSource"
  className="com.ibm.websphere.objectgrid.security.plugins.builtins.
          WSSubjectSourceImpl" />
                            <bean
  id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxslocalsec_SubjectValidation"
   className="com.ibm.websphere.objectgrid.security.plugins.builtins.
          WSSubjectValidationImpl" />
                            <bean
  id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_cxslocalsec_TransactionCallback"
  className="com.ibm.websphere.samples.objectgrid.
          HeapTransactionCallback" />
  ...
  </objectGrids>
  ```

- **Programming** If you want to create an ObjectGrid through APIs, you can call the following methods to set the SubjectSource or SubjectValidation plug-ins.

```
**
* Set the SubjectValidation plug-in for this ObjectGrid instance. A
* SubjectValidation plug-in can be used to validate the Subject object
* passed in as a valid Subject. Refer to {@link SubjectValidation}
* for more details.
* @param subjectValidation the SubjectValidation plug-in
*/
void setSubjectValidation(SubjectValidation subjectValidation);


/**
* Set the SubjectSource plug-in. A SubjectSource plug-in can be used
* to get a Subject object from the environment to represent the
* ObjectGrid client.
*
* @param source the SubjectSource plug-in
*/
void setSubjectSource(SubjectSource source);
```

Write your own JAAS authentication code

You can write you own Java Authentication and Authorization Service (JAAS) authentication code to handle the authentication. You need to write your own login modules and then configure the login modules for your authentication module.

The login module receives information about a user and authenticates the user. This information can be anything that can identify the user. For example, the information can be a user ID and password, client certificate, and so on. After receiving the information, the login module verifies that the information represents a valid subject and then creates a Subject object. Currently, several implementations of login modules are available to the public.

After a login module is written, configure this login module for the run time to use. You must configure a JAAS login module. This login module file contains the module and its authentication scheme. For example:

```
FileLogin
{
    com.acme.auth.FileLoginModule required
};
```

The authentication scheme is FileLogin and the login module is com.acme.auth.FileLoginModule. The required token indicates that the FileLoginModule module must validate this login or the entire scheme fails.

Setting the JAAS login module configuration file can be done in one of the following ways:

- Set the JAAS login module configuration file in the login.config.url property in the java.security file, for example:

  ```
  login.config.url.1=file:${java.home}/lib/security/file.login
  ```

- Set the JAAS login module configuration file from the command line by using the **-Djava.security.auth.login.config** Java virtual machine (JVM) arguments , for example, `-Djava.security.auth.login.config ==$JAVA_HOME/lib/security/file.login`

For more information, see Java SE security tutorial - Step 2.For more information, see Java SE security tutorial - Step 2.
If your code is running in WebSphere Application Server, you must configure the JAAS login in the administrative console and store this login configuration in the application server configuration. See Login configuration for Java Authentication and Authorization Service for details.

Troubleshooting

In addition to the logs and trace, messages, and release notes discussed in this section, you can use monitoring tools to figure out issues such as the location of data in the environment, the availability of servers in the data grid, and so on. If you are running in a WebSphere® Application Server environment, you can use Performance Monitoring Infrastructure (PMI). If you are running in a stand-alone environment, you can use a vendor monitoring tool, such as CA Wily Introscope or Hyperic HQ. You can also use and customize the **xscmd** utility to display textual information about your environment.

- Troubleshooting and support for WebSphere eXtreme Scale
  To isolate and resolve problems with your IBM products, you can use the troubleshooting and support information. This information contains instructions for using the problem-determination resources that are provided with your IBM products, including WebSphere eXtreme Scale .
- Enabling logging
  You can use logs to monitor and troubleshoot your environment.
- Collecting trace
  You can use trace to monitor and troubleshoot your environment. You must provide trace for a server when you work with IBM® support.
- Analyzing log and trace data
  You can use the log analysis tools to analyze how your runtime environment is performing and solve problems that occur in the environment.
- 8.5+ Troubleshooting the product installation
  IBM Installation Manager is a common installer for many IBM software products that you use to install this version of WebSphere eXtreme Scale.
- Troubleshooting client connectivity
  There are several common problems specific to clients and client connectivity that you can solve as described in the following sections.
- Troubleshooting cache integration
  Use this information to troubleshoot issues with your cache integration configuration, including HTTP session and dynamic cache configurations.

- Troubleshooting the JPA cache plug-in
  Use this information to troubleshoot issues with your JPA cache plug-in configuration. These problems can occur in both Hibernate and OpenJPA configurations.
- Troubleshooting IBM eXtremeMemory
  Use the following information to troubleshoot eXtremeMemory.
- Troubleshooting administration
  Use the following information to troubleshoot administration, including starting and stopping servers, using the **xscmd** utility, and so on.
- Troubleshooting multiple data center configurations
  Use this information to troubleshoot multiple data center configurations, including linking between catalog service domains.
- Troubleshooting high availability
  Use this information to troubleshoot high availability.
- Troubleshooting loaders
  Use this information to troubleshoot issues with your database loaders.
- Troubleshooting XML configuration
  When you configure eXtreme Scale, you can encounter unexpected behavior with your XML files. The following sections describe problems that can occur and solutions.
- Troubleshooting deadlocks
  The following sections describe some of the most common deadlock scenarios and suggestions on how to avoid them.
- Troubleshooting security
  Use this information to troubleshoot issues with your security configuration.
- Troubleshooting Liberty profile configurations
  Use this information to troubleshoot commonly experienced problems with the Liberty profile.
- IBM Support Assistant for WebSphere eXtreme Scale
  You can use the IBM Support Assistant to collect data, analyze symptoms, and access product information.

Troubleshooting and support for WebSphere eXtreme Scale

To isolate and resolve problems with your IBM products, you can use the troubleshooting and support information. This information contains instructions for using the problem-determination resources that are provided with your IBM products, including WebSphere® eXtreme Scale .

- Techniques for troubleshooting problems
  Troubleshooting is a systematic approach to solving a problem. The goal of troubleshooting is to determine why something does not work as expected and how to resolve the problem. Certain common techniques can help with the task of troubleshooting.
- Searching knowledge bases
  You can often find solutions to problems by searching IBM knowledge bases. You can optimize your results by using available resources, support tools, and search methods.
- Getting fixes
  A product fix might be available to resolve your problem.
- Contacting IBM Support
  IBM Support provides assistance with product defects, answers FAQs, and helps users resolve problems with the product.
- Exchanging information with IBM
  To diagnose or identify a problem, you might need to provide IBM Support with data and information from your system. In other cases, IBM Support might provide you with tools or utilities to use for problem determination.
- Subscribing to Support updates
  To stay informed of important information about the IBM products that you use, you can subscribe to updates.

Techniques for troubleshooting problems

Troubleshooting is a systematic approach to solving a problem. The goal of troubleshooting is to determine why something does not work as expected and how to resolve the problem. Certain common techniques can help with the task of troubleshooting.
The first step in the troubleshooting process is to describe the problem completely. Problem descriptions help you and the IBM technical-support representative know where to start to find the cause of the problem. This step includes asking yourself basic questions:

- What are the symptoms of the problem?
- Where does the problem occur?
- When does the problem occur?
- Under which conditions does the problem occur?
- Can the problem be reproduced?

The answers to these questions typically lead to a good description of the problem, which can then lead you to a problem resolution.

What are the symptoms of the problem?

When starting to describe a problem, the most obvious question is "What is the problem?" This question might seem straightforward; however, you can break it down into several more-focused questions that create a more descriptive picture of the problem. These questions can include:

- Who, or what, is reporting the problem?
- What are the error codes and messages?
- How does the system fail? For example, is it a loop, hang, crash, performance degradation, or incorrect result?

Where does the problem occur?

Determining where the problem originates is not always easy, but it is one of the most important steps in resolving a problem. Many layers of technology can exist between the reporting and failing components. Networks, the data grid, and servers are only a few of the components to consider when you are investigating problems.

The following questions help you to focus on where the problem occurs to isolate the problem layer:

- Is the problem specific to one platform or operating system, or is it common across multiple platforms or operating systems?
- Is the current environment and configuration supported?
- Do all users have the problem?
- (For multi-site installations.) Do all sites have the problem?

If one layer reports the problem, the problem does not necessarily originate in that layer. Part of identifying where a problem originates is understanding the environment in which it exists. Take some time to completely describe the problem environment, including the operating system and version, all corresponding software and versions, and hardware information. Confirm that you are running within an environment that is a supported configuration; many problems can be traced back to incompatible levels of software that are not intended to run together or have not been fully tested together.

When does the problem occur?

Develop a detailed timeline of events leading up to a failure, especially for those cases that are one-time occurrences. You can most easily develop a timeline by working backward: Start at the time an error was reported (as precisely as possible, even down to the millisecond), and work backward through the available logs and information. Typically, you need to look only as far as the first suspicious event that you find in a diagnostic log.

To develop a detailed timeline of events, answer these questions:

- Does the problem happen only at a certain time of day or night?
- How often does the problem happen?
- What sequence of events leads up to the time that the problem is reported?
- Does the problem happen after an environment change, such as upgrading or installing software or hardware?

Responding to these types of questions can give you a frame of reference in which to investigate the problem.

Under which conditions does the problem occur?

Knowing which systems and applications are running at the time that a problem occurs is an important part of troubleshooting. These questions about your environment can help you to identify the root cause of the problem:

- Does the problem always occur when the same task is being performed?
- Does a certain sequence of events need to happen for the problem to occur?
- Do any other applications fail at the same time?

Answering these types of questions can help you explain the environment in which the problem occurs and correlate any dependencies. Remember that just because multiple problems might have occurred around the same time, the problems are not necessarily related.

Can the problem be reproduced?

From a troubleshooting standpoint, the ideal problem is one that can be reproduced. Typically, when a problem can be reproduced you have a larger set of tools or procedures at your disposal to help you investigate. Consequently, problems that you can reproduce are often easier to debug and solve.

However, problems that you can reproduce can have a disadvantage: If the problem is of significant business impact, you do not want it to recur. If possible, re-create the problem in a test or development environment, which typically offers you more flexibility and control during your investigation.

- Can the problem be re-created on a test system?
- Are multiple users or applications encountering the same type of problem?
- Can the problem be recreated by running a single command, a set of commands, or a particular application?

Searching knowledge bases

You can often find solutions to problems by searching IBM knowledge bases. You can optimize your results by using available resources, support tools, and search methods.

About this task

You can find useful information by searching the information center for WebSphere® eXtreme Scale . However, sometimes you need to look beyond the information center to answer your questions or resolve problems.

Procedure

To search knowledge bases for information that you need, use one or more of the following approaches:

- Search for content by using the IBM® Support Assistant (ISA).

ISA is a no-charge software serviceability workbench that helps you answer questions and resolve problems with IBM software products. You can find instructions for downloading and installing ISA on the ISA website.

- Find the content that you need by using the IBM Support Portal.
  The IBM Support Portal is a unified, centralized view of all technical support tools and information for all IBM systems, software, and services. The IBM Support Portal lets you access the IBM electronic support portfolio from one place. You can tailor the pages to focus on the information and resources that you need for problem prevention and faster problem resolution. Familiarize yourself with the IBM Support Portal by viewing the demo videos (https://www.ibm.com/blogs/SPNA/entry/the_ibm_support_portal_videos) about this tool. These videos introduce you to the IBM Support Portal, explore troubleshooting and other resources, and demonstrate how you can tailor the page by moving, adding, and deleting portlets.

- Search for content about WebSphere eXtreme Scale by using one of the following additional technical resources:
  - WebSphere eXtreme Scale release notes
  - WebSphere eXtreme Scale Support website
  - WebSphere eXtreme Scale forum
- Search for content by using the IBM masthead search. You can use the IBM masthead search by typing your search string into the Search field at the top of any ibm.com® page.
- Search for content by using any external search engine, such as Google, Yahoo, or Bing. If you use an external search engine, your results are more likely to include information that is outside the ibm.com domain. However, sometimes you can find useful problem-solving information about IBM products in newsgroups, forums, and blogs that are not on ibm.com.
  Tip: Include "IBM" and the name of the product in your search if you are looking for information about an IBM product.

Getting fixes

A product fix might be available to resolve your problem.

Procedure

To find and install fixes:

1. Obtain the tools required to get the fix. Use the IBM Update Installer to install and apply various types of maintenance packages for WebSphere eXtreme Scale or WebSphere eXtreme Scale Client. Because the Update Installer undergoes regular maintenance, you must use the most current version of the tool.
2. Determine which fix you need. See the Recommended fixes for WebSphere eXtreme Scale to select the latest fix. When you select a fix, the download document for that fix opens.
3. Download the fix. In the download document, click the link for the latest fix in the "Download package" section.
4. Apply the fix. Follow the instructions in the "Installation Instructions" section of the download document.
5. Subscribe to receive weekly e-mail notifications about fixes and other IBM Support information.

- Getting fixes from Fix Central
  You can use Fix Central to find the fixes that are recommended by IBM Support for a variety of products, including WebSphere eXtreme Scale . With Fix Central, you can search, select, order, and download fixes for your system with a choice of delivery options. A WebSphere eXtreme Scale product fix might be available to resolve your problem.

Getting fixes from Fix Central

You can use Fix Central to find the fixes that are recommended by IBM Support for a variety of products, including WebSphere® eXtreme Scale . With Fix Central, you can search, select, order, and download fixes for your system with a choice of delivery options. A WebSphere eXtreme Scale product fix might be available to resolve your problem.

Procedure

To find and install fixes:

1. Obtain the tools that are required to get the fix. If it is not installed, obtain your product update installer. You can download the installer from Fix Central. This site provides download, installation, and configuration instructions for the update installer.
2. Select as the product, and select one or more check boxes that are relevant to the problem that you want to resolve.
3. Identify and select the fix that is required.
4. Download the fix.
   a. Open the download document and follow the link in the "Download Package" section.
   b. When downloading the file, ensure that the name of the maintenance file is not changed. This change might be intentional, or it might be an inadvertent change that is caused by certain web browsers or download utilities.
5. Apply the fix.
   a. Follow the instructions in the "Installation Instructions" section of the download document.
   b. For more information, see the "Installing fixes with the Update Installer" topic in the product documentation.
6. Optional: Subscribe to receive weekly e-mail notifications about fixes and other IBM Support updates.

Contacting IBM Support

IBM Support provides assistance with product defects, answers FAQs, and helps users resolve problems with the product.

Before you begin

After trying to find your answer or solution by using other self-help options, such as release notes, you can contact IBM Support. Before contacting IBM Support, your company or organization must have an active IBM maintenance contract, and you must be authorized to submit problems to IBM. For information about the types of available support, see the Support portfolio topic in the "*Software Support Handbook*".

Procedure

To contact IBM Support about a problem:

1. Define the problem, gather background information, and determine the severity of the problem. For more information, see the Getting IBM support topic in the *Software Support Handbook*.
2. Gather diagnostic information.
3. Submit the problem to IBM Support in one of the following ways:
   - With IBM® Support Assistant (ISA). For more information, see IBM Support Assistant for WebSphere eXtreme Scale.
   - Online through the IBM Support Portal: You can open, update, and view all of your service requests from the Service Request portlet on the Service Request page.
   - By phone: For the phone number to call in your region, see the Directory of worldwide contacts web page.

Results

If the problem that you submit is for a software defect or for missing or inaccurate documentation, IBM Support creates an Authorized Program Analysis Report (APAR). The APAR describes the problem in detail. Whenever possible, IBM Support provides a workaround that you can implement until the APAR is resolved and a fix is delivered. IBM publishes resolved APARs on the IBM Support website daily, so that other users who experience the same problem can benefit from the same resolution.

Exchanging information with IBM

To diagnose or identify a problem, you might need to provide IBM Support with data and information from your system. In other cases, IBM Support might provide you with tools or utilities to use for problem determination.

Sending information to IBM Support

To reduce the time that is required to resolve your problem, you can send trace and diagnostic information to IBM Support.

Procedure

To submit diagnostic information to IBM Support:

1. Open a problem management record (PMR).
2. Collect the diagnostic data that you need. Diagnostic data helps reduce the time that it takes to resolve your PMR. You can collect the diagnostic data manually or automatically:
   - Collect the data manually.
   - Collect the data automatically.
3. Compress the files by using the .zip or .tar file format.
4. Transfer the files to IBM. You can use one of the following methods to transfer the files to IBM:
   - IBM® Support Assistant
   - The Service Request tool
   - Standard data upload methods: FTP, HTTP
   - Secure data upload methods: FTPS, SFTP, HTTPS
   - E-mail

   If you are using a z/OS product and you use ServiceLink / IBMLink to submit PMRs, you can send diagnostic data to IBM Support in an e-mail or by using FTP.

   All of these data exchange methods are explained on the IBM Support website.

Receiving information from IBM Support

Occasionally an IBM technical-support representative might ask you to download diagnostic tools or other files. You can use FTP to download these files.

Before you begin

Ensure that your IBM technical-support representative provided you with the preferred server to use for downloading the files and the exact directory and file names to access.

Procedure

To download files from IBM Support:

1. Use FTP to connect to the site that your IBM technical-support representative provided and log in as `anonymous`. Use your e-mail address as the password.
2. Change to the appropriate directory:
    a. Change to the /fromibm directory.

    ```
    cd fromibm
    ```

    b. Change to the directory that your IBM technical-support representative provided.

    ```
    cd nameofdirectory
    ```

3. Enable binary mode for your session.

    ```
    binary
    ```

4. Use the **get** command to download the file that your IBM technical-support representative specified.

    ```
    get filename.extension
    ```

5. End your FTP session.

    ```
    quit
    ```

Subscribing to Support updates

To stay informed of important information about the IBM products that you use, you can subscribe to updates.

About this task

By subscribing to receive updates about the product, you can receive important technical information and updates for specific IBM Support tools and resources. You can subscribe to updates by using one of two approaches:

Social media subscriptions
: The following RSS feed is available for the product:

    - RSS feed for WebSphere® eXtreme Scale forum

    For general information about RSS, including steps for getting started and a list of RSS-enabled IBM web pages, visit the IBM Software Support RSS feeds site.

My Notifications
: With My Notifications, you can subscribe to Support updates for any IBM product. My Notifications replaces My Support, which is a similar tool that you might have used in the past. With My Notifications, you can specify that you want to receive daily or weekly e-mail announcements. You can specify what type of information you want to receive, such as publications, hints and tips, product flashes (also known as alerts), downloads, and drivers. My Notifications enables you to customize and categorize the products about which you want to be informed and the delivery methods that best suit your needs.

Procedure

To subscribe to Support updates:

1. Subscribe to the RSS feed for the WebSphere eXtreme Scale forum .
    a. On the subscription page, click the RSS feed icon.
    b. Select the option that you want to use to subscribe to the feed.
    c. Click Subscribe.
2. Subscribe to My Notifications by going to the IBM® Support Portal and click My Notifications in the Notifications portlet.
3. Sign in using your IBM ID and password, and click Submit.
4. Identify what and how you want to receive updates.
    a. Click the Subscribe tab.
    b. Select the appropriate software brand or type of hardware.
    c. Select one or more products by name and click Continue.
    d. Select your preferences for how to receive updates, whether by e-mail, online in a designated folder, or as an RSS or Atom feed.
    e. Select the types of documentation updates that you want to receive, for example, new information about product downloads and discussion group comments.
    f. Click Submit.

Results

Until you modify your RSS feeds and My Notifications preferences, you receive notifications of updates that you have requested. You can modify your preferences when needed; for example, if you stop using one product and begin using another product.

Related information

IBM Software Support RSS feeds

Subscribe to My Notifications support content updates

My Notifications for IBM technical support

My Notifications for IBM technical support overview

Enabling logging

You can use logs to monitor and troubleshoot your environment.

About this task

Logs are saved different locations and formats depending on your configuration.

Procedure

- **Enable logs in a stand-alone environment.**
  With stand-alone catalog servers, the logs are in the location where you run the start server command. For container servers, you can use the default location or set a custom log location:
  - **Default log location:** The logs are in the directory where the start server command was run. If you start the servers in the *wxs_home*/bin directory, the logs and trace files are in the logs/*<server_name>* directories in the bin directory.
  - **Custom log location:** To specify an alternate location for container server logs, create a properties file, such as server.properties, with the following contents:

    ```
    workingDirectory=<directory>
    traceSpec=
    systemStreamToFileEnabled=true
    ```

    The workingDirectory property is the root directory for the logs and optional trace file. WebSphere® eXtreme Scale creates a directory with the name of the container server with a SystemOut.log file, a SystemErr.log file, and a trace file. To use a properties file during container startup, use the -serverProps option and provide the server properties file location.
- **Enable logs in WebSphere Application Server.**
  See WebSphere Application Server: Enabling and disabling logging for more information.
- **Retrieve FFDC files.**
  FFDC files are for IBM® support to aid in debug. These files might be requested by IBM support when a problem occurs. These files are in a directory labeled, `ffdc`, and contain files that resemble the following:

  ```
  server2_exception.log
  server2_20802080_07.03.05_10.52.18_0.txt
  ```

What to do next

View the log files in their specified locations. Common messages to look for in the SystemOut.log file are start confirmation messages, such as the following example:

```
CWOBJ1001I: ObjectGrid Server catalogServer01 is ready to process requests.
```

For more information about a specific message in the log files, see Messages.

Related tasks:
Collecting trace
Starting stand-alone servers
Administering with the xscmd utility
Related reference:
Server trace options
Messages

Collecting trace

You can use trace to monitor and troubleshoot your environment. You must provide trace for a server when you work with IBM® support.

About this task

Collecting trace can help you monitor and fix problems in your deployment of WebSphere® eXtreme Scale. How you collect trace depends on your configuration. See Server trace options for a list of the different trace specifications you can collect.

Procedure

- **Collect trace within a WebSphere Application Server environment.**
  If your catalog and container servers are in a WebSphere Application Server environment, see WebSphere Application Server: Working with trace for more information.
- **Collect trace with the stand-alone catalog or container server start command.**
  You can set trace on a catalog service or container server by using the -traceSpec and -traceFile parameters with the start server command. For example:

```
startOgServer.sh catalogServer -traceSpec ObjectGridPlacement=all=enabled -traceFile  /home/user1/logs/trace.log
```

The -traceFile parameter is optional. If you do not set a -traceFile location, the trace file goes to the same location as the system out log files.

For more information about these parameters, see startOgServer script .

- **Collect trace on the stand-alone catalog or container server with a properties file.**
  To collect trace from a properties file, create a file, such as a server.properties file, with the following contents:

  ```
  workingDirectory=<directory>
  traceSpec=<trace_specification>
  systemStreamToFileEnabled=true
  ```

  The workingDirectory property is the root directory for the logs and optional trace file. If the workingDirectory value is not set, the default working directory is the location used to start the servers, such as *wxs_home*/bin. To use a properties file during server startup, use the -serverProps parameter with the **startOgServer** command and provide the server properties file location.
  For more information about the server properties file and how to use the file, see Server properties file.
- **Collect trace on a stand-alone client.**
  You can start trace collection on a stand-alone client by adding system properties to the startup script for the client application. In the following example, trace settings are specified for the `com.ibm.samples.MyClientProgram` application:

  ```
  java -DtraceSettingsFile=MyTraceSettings.properties
  -Djava.util.logging.manager=com.ibm.ws.bootstrap.WsLogManager
  -Djava.util.logging.configureByServer=true com.ibm.samples.MyClientProgram
  ```

  Fore more information, see WebSphere Application Server: Enabling trace on client and stand-alone applications.
- **Collect trace with the ObjectGridManager interface.**
  You can also set trace during run time on an ObjectGridManager interface. Setting trace on an ObjectGridManager interface can be used to get trace on an eXtreme Scale client while it connects to an eXtreme Scale and commits transactions. To set trace on an ObjectGridManager interface, supply a trace specification and a trace log.

  ```
  ObjectGridManager manager = ObjectGridManagerFactory.getObjectGridManager();
  ...
  manager.setTraceEnabled(true);
  manager.setTraceFileName("logs/myClient.log");
  manager.setTraceSpecification("ObjectGridReplication=all=enabled");
  ```

  For more information about the ObjectGridManager interface, see Interacting with an ObjectGrid using the ObjectGridManager interface.
- **Collect trace on container servers with the xscmd utility.**
  To collect trace with the **xscmd** utility, use the -c setTraceSpec command. Use the **xscmd** utility to collect trace on a stand-alone environment during run time instead of during startup. You can collect trace on all servers and catalog services or you can filter the servers based on the ObjectGrid name, and other properties. For example, to collect ObjectGridReplication trace with access to the catalog service server, run:

  ```
  xscmd -c setTraceSpec -spec "ObjectGridReplication=all=enabled"
  ```

  You can also disable trace by setting the trace specification to `*=all=disabled`.

Results

Trace files are written to the specified location.

- Server trace options
  You can enable trace to provide information about your environment to IBM support.

Related tasks:
Enabling logging
Starting stand-alone servers
Administering with the xscmd utility
Related reference:
Server trace options
Messages

Server trace options

You can enable trace to provide information about your environment to IBM® support.

Trace option components

WebSphere® eXtreme Scale trace is divided into several different components. You can specify the level of trace to use for a catalog server or container server. Common levels of trace include: all, debug, entryExit, and event.
An example trace string follows:

```
ObjectGridComponent=level=enabled
```

You can concatenate trace strings. Use the `*` (asterisk) symbol to specify a wildcard value, such as `ObjectGrid*=all=enabled`. If you need to provide a trace to IBM support, a specific trace string is requested. For example, if a problem with replication occurs, the `ObjectGridReplication=debug=enabled` trace string might be requested.

Trace specification

Table 1. Trace options. Trace options for WebSphere eXtreme Scale

Trace option	Description
ObjectGrid	General core cache engine.
ObjectGridCacheInvalidator	Near-cache invalidation
ObjectGridCatalogServer	General catalog service.
ObjectGridChannel	Static deployment topology communications.
ObjectGridClientInfo	DB2® client information.
ObjectGridClientInfoUser	DB2 user information.
ObjectgridCORBA	Dynamic deployment topology communications.
ObjectGridDataGrid	The AgentManager API.
ObjectGridDynaCache	The WebSphere eXtreme Scale dynamic cache provider.
ObjectGridEntityManager	The EntityManager API. Use with the Projector option.
ObjectGridEvictors	ObjectGrid built-in evictors.
ObjectGridJPA	Java™ Persistence API (JPA) loaders.
ObjectGridJPACache	JPA cache plug-ins.
ObjectGridLocking	ObjectGrid cache entry lock manager.
ObjectGridMBean	Management beans.
ObjectGridMonitor	Historical monitoring infrastructure.
ObjectGridNative	WebSphere eXtreme Scale native code trace, including eXtremeMemory native code.
ObjectGridOSGi	The WebSphere eXtreme Scale OSGi integration components.
ObjectGridPlacement	Catalog server shard placement service.
ObjectGridPubSub	Catalog and container server shard placement service.
ObjectGridQuery	ObjectGrid query.
ObjectGridReplication	Replication service.
ObjectGridRest	REST gateway.
ObjectGridRouting	Client/server routing details.
ObjectGridSecurity	Security trace.
ObjectGridSerializer	The DataSerializer plug-in infrastructure.
ObjectGridStats	ObjectGrid statistics.
ObjectGridTransactionManager	The WebSphere eXtreme Scale transaction manager.
ObjectGridWriteBehind	ObjectGrid write behind.
ObjectGridXM	General IBM eXtremeMemory trace.
ObjectGridXMEviction	eXtremeMemory eviction trace.
ObjectGridXMTransport	eXtremeMemory general transport trace.
ObjectGridXMTransportInbound	eXtremeMemory inbound specific transport trace.
ObjectGridXMTransportOutbound	eXtremeMemory outbound specific transport trace.
Projector	The engine within the EntityManager API.
QueryEngine	The query engine for the Object Query API and EntityManager Query API.
QueryEnginePlan	Query plan trace.
TCPChannel	The IBM eXtremeIO TCP/IP channel.
WXSRevision	Revision control for replication.
XsByteBuffer	WebSphere eXtreme Scale byte buffer trace.

Related tasks:
Enabling logging
Collecting trace
Starting stand-alone servers
Administering with the xscmd utility

Analyzing log and trace data

You can use the log analysis tools to analyze how your runtime environment is performing and solve problems that occur in the environment.

About this task

You can generate reports from the existing log and trace files in the environment. These visual reports can be used for the following purposes:

- **To analyze runtime environment status and performance:**
  - Deployment environment consistency
  - Logging frequency
  - Running topology versus configured topology
  - Unplanned topology changes
  - Partition replication status
  - Statistics of memory, throughput, processor usage, and so on
- **To troubleshoot problems in the environment:**
  - Topology views at specific points in time
  - Statistics of memory, throughput, processor usage during client failures
  - Current fix pack levels, tuning settings

- Log analysis overview
  You can use the **xsLogAnalyzer** tool to help troubleshoot issues in the environment.
- Running log analysis
  You can run the **xsLogAnalyzer** tool on a set of log and trace files from any computer.
- Creating custom scanners for log analysis
  You can create custom scanners for log analysis. After you configure the scanner, the results are generated in the reports when you run the **xsLogAnalyzer** tool. The custom scanner scans the logs for event records based on the regular expressions that you specified.
- Troubleshooting log analysis
  Use the following troubleshooting information to diagnose and fix problems with the **xsLogAnalyzer** tool and its generated reports.

Log analysis overview

You can use the **xsLogAnalyzer** tool to help troubleshoot issues in the environment.

All failover messages

Displays the total number of failover messages as a chart over time. Also displays a list of the failover messages, including the servers that have been affected

All eXtreme Scale critical messages

Displays message IDs along with the associated explanations and user actions, which can save you the time from searching for messages.

All exceptions

Displays the top five exceptions, including the messages and how many times they occurred, and what servers were affected by the exception.

Topology summary

Displays a diagram of how your topology is configured according to the log files. You can use this summary to compare to your actual configuration, possibly identifying configuration errors.

Topology consistency: Object Request Broker (ORB) comparison table

Displays ORB settings in the environment. You can use this table to help determine if the settings are consistent across your environment.

Event timeline view

Displays a timeline diagram of different actions that have occurred on the data grid, including life cycle events, exceptions, critical messages, and first-failure data capture (FFDC) events.

Running log analysis

You can run the **xsLogAnalyzer** tool on a set of log and trace files from any computer.

Before you begin

- Enable logs and trace. See Enabling logging and Collecting trace for more information.

- Collect your log files. The log files can be in various locations depending on how you configured them. If you are using the default log settings, you can get the log files from the following locations:
  - In a stand-alone installation: *wxs_install_root*/bin/logs/*<server_name>*
  - In an installation that is integrated with WebSphere® Application Server: *was_root*/logs/*<server_name>*
- Collect your trace files. The trace files can be in various locations depending on how you configured them. If you are using the default trace settings, you can get the trace files from the following locations:
  - In a stand-alone installation: If no specific trace value is set, the trace files are written to the same location as the system out log files.
  - In an installation that is integrated with WebSphere Application Server: *was_root*/profiles/*server_name*/logs.

  Copy the log and trace files to the computer from which you are planning to use the log analyzer tool.
- If you want to create custom scanners in your generated report, create a scanner specifications properties file and configuration file before you run the tool. For more information, see Creating custom scanners for log analysis.

Procedure

1. Run the **xsLogAnalyzer** tool.
   The script is in the following locations :
   - In a stand-alone installation: *wxs_install_root*/ObjectGrid/bin
   - In an installation that is integrated with WebSphere Application Server: *was_root*/bin

   Tip: If your log files are large, consider using the -startTime, -endTime, and -maxRecords parameters when you run the report to restrict the number of log entries that are scanned. Using these parameters when you run the report makes the reports easier to read and run more effectively. You can run multiple reports on the same set of log files.

   ```
   xsLogAnalyzer.sh|bat -logsRoot c:\myxslogs -outDir c:\myxslogs\out
   -startTime 11.09.27_15.10.56.089 -endTime 11.09.27_16.10.56.089 -maxRecords 100
   ```

   -logsRoot
   : Specifies the absolute path to the log directory that you want to evaluate (required).

   -outDir
   : Specifies an existing directory to write the report output. If you do not specify a value, the report is written to the root location of the **xsLogAnalyzer** tool.

   -startTime
   : Specifies the start time to evaluate in the logs. The date is in the following format: *year.month.day_hour.minute.second.millisecond*

   -endTime
   : Specifies the end time to evaluate in the logs. The date is in the following format: *year.month.day_hour.minute.second.millisecond*

   -trace
   : Specifies a trace string, such as `ObjectGrid*=all=enabled`.

   -maxRecords
   : Specifies the maximum number of records to generate in the report. The default is `100`. If you specify the value as 50, the first 50 records are generated for the specified time period.

2. Open the generated files. If you did not define an output directory, the reports are generated in a folder called report_*date_time*. To open the main page of the reports, open the index.html file.
3. Use the reports to analyze the log data. Use the following tips to maximize the performance of the report displays:
   - To maximize the performance of queries on the log data, use as specific information as possible. For example, a query for `server` takes much longer to run and returns more results than `server_host_name`.
   - Some views have a limited number of data points that are displayed at one time. You can adjust the segment of time that is being viewed by changing the current data, such as start and end time, in the view.

What to do next

For more information about troubleshooting the **xsLogAnalyzer** tool and the generated reports, see Troubleshooting log analysis.

Creating custom scanners for log analysis

You can create custom scanners for log analysis. After you configure the scanner, the results are generated in the reports when you run the **xsLogAnalyzer** tool. The custom scanner scans the logs for event records based on the regular expressions that you specified.

Procedure

1. Create a scanner specifications properties file that specifies the general expression to run for the custom scanner.
   a. Create and save a properties file. The file must be in the *loganalyzer_root*/config/custom directory. You can name the file as: you like. The file is used by the new scanner, so naming the scanner in the properties file is useful, for example: *my_new_server_scanner_spec*.properties.
   b. Include the following properties in the *my_new_server_scanner_spec*.properties file:

      ```
      include.regular_expression = REGULAR_EXPRESSION_TO_SCAN
      ```

      The *REGULAR_EXPRESSION_TO_SCAN* variable is a regular expression on which to filter the log files.
      Example: To scan for instances of lines that contain both the "`xception`" and "`rror`" strings regardless of the order, set the include.regular_expression property to the following value:

      ```
      include.regular_expression = (xception.+rror)|(rror.+xception)
      ```

      This regular expression causes events to be recorded if the string "`rror`" comes before or after the "`xception`" string.

Example:

To scan through each line in the logs for instances of lines that contain either the phrase "`xception`" or the phrase "`rror`" strings regardless of the order, set the include.regular_expression property to the following value:

```
include.regular_expression = (xception)|(rror)
```

This regular expression causes events to be recorded if the either the "`rror`" string or the "`xception`" string exist.

2. Create a configuration file that the **xsLogAnalyer** tool uses to create the scanner.
   a. Create and save a configuration file. The file must be in the *loganalyzer_root*/config/custom directory. You can name the file as *scanner_name*Scanner.config, where *scanner_name* is a unique name for the new scanner. For example, you might name the file serverScanner.config
   b. Include the following properties in the *scanner_name*Scanner.config file:

```
scannerSpecificationFiles = LOCATION_OF_SCANNER_SPECIFICATION_FILE
```

The *LOCATION_OF_SCANNER_SPECIFICATION_FILE* variable is the path and location of the specification file that you created in the previous step. For example: *loganalyzer_root*/config/custom/my_new_scanner_spec.properties. You can also specify multiple scanner specification files by using a semi-colon separated list:

```
scannerSpecificationFiles = LOCATION_OF_SCANNER_SPECIFICATION_FILE1;LOCATION_OF_SCANNER_SPECIFICATION_FILE2
```

3. Run the **xsLogAnalyzer** tool. For more information, see Running log analysis.

Results

After you run the **xsLogAnalyzer** tool, the report contains new tabs in the report for the custom scanners that you configured. Each tab contains the following views:

Charts
: A plotted graph that illustrates recorded events. The events are displayed in the order in which the events were found.

Tables
: A tabular representation of the recorded events.

Summary reports

Troubleshooting log analysis

Use the following troubleshooting information to diagnose and fix problems with the **xsLogAnalyzer** tool and its generated reports.

Procedure

- **Problem:** Out of memory conditions occur when you are using the **xsLogAnalyzer** tool to generate reports. An example of an error that might occur follows: `java.lang.OutOfMemoryError: GC overhead limit exceeded`.
  Solution: The **xsLogAnalyzer** tool runs within a Java virtual machine (JVM). You can configure the JVM to increase the heap size before you run the **xsLogAnalyzer** tool by specifying some settings when you run the tool. Increasing the heap size enables more event records to be stored in JVM memory. Start with a setting of 2048M, assuming the operating system has enough main memory. On the same command-line instance in which you are planning to run the **xsLogAnalyzer** tool, set the maximum JVM heap size:

  ```
  java -XmxHEAP_SIZEm
  ```

  The *HEAP_SIZE* value can be any integer and represents the number of megabytes that are allocated to JVM heap.
  For example, you might run `java -Xmx2048m`. If the out of memory messages continue, or you do not have the resources to allocate 2048m or more of memory, limit the number of events that are being held in the heap. You can limit the number of events in the heap up by passing the -maxRecords parameter to the .**xsLogAnalyzer** command
- **Problem:** When you open a generated report from the **xsLogAnalyzer** tool, the browser hangs or does not load the page.
  Cause: The generated HTML files are too large and cannot be loaded by the browser. These files are large because the scope of the log files that you are analyzing is too broad.

  Solution: Consider using the -startTime, -endTime, and -maxRecords parameters when you run the **xsLogAnalyzer** tool to restrict the number of log entries that are scanned. Using these parameters when you run the report makes the reports easier to read and run more effectively. You can run multiple reports on the same set of log files.

Troubleshooting installation

Use this information to troubleshoot issues with your installation and updates.

Procedure

- **Problem:** When you run the installation command from a remote computer, such as `\\mymachine\downloads\`, the following message displays: `CMD.EXE was started with the above path as the current directory. UNC paths are not supported. Defaulting to Windows directory.` As a result, the installation does not complete correctly.

Solution: Map the remote computer to a network drive. For example, in Windows, you can right-click My computer and choose Map Network Drive and include the uniform naming conventions (UNC) path to the remote computer. You can then run the installation script from the network drive successfully, for example: y:\mymachine\downloads\WXS\install.bat.

- **Problem:** The installation completes unsuccessfully.
  Solution: Check the log files to see where the installation failed. When the installation completes unsuccessfully, the logs are in the *wxs_install_root*/logs/wxs directory.

- **Problem:** A catastrophic failure occurs during the installation.
  Solution: Check the log files to see where the installation failed. When the installation fails when it is partially completed, the logs can generally be found in the *user_root*/wxs_install_logs/ directory.

- Windows **Problem:** If you are installing the WebSphere® eXtreme Scale Client on Windows, you might see the following text in the results of the installation:

  ```
  Success: The installation of the following product was successful:
  WebSphere eXtreme Scale Client.  Some configuration steps have errors.
  For more information, refer to the following log file:
  <WebSphere Application Server install root>\logs\wxs_client\install\log.txt"
  Review the installation log (log.txt) and review the deployment manager
  augmentation log.
  ```

  Solution: If you see a failure with the iscdeploy.sh file, you can ignore the error. This error does not cause any problems.
- Linux **Problem:**
  If you have a full installation and try to apply WebSphere eXtreme Scale Client only maintenance with the update installer, you see the following message:

  ```
  Prerequisite checking has failed. Click Back to select a different package,
  or click Cancel to exit.

  Failure messages are:

  Required feature wxs.client.primary is not found.
  ```

  If you have WebSphere eXtreme Scale Client installed and try to apply a full maintenance package with the update installer, you see the following message:

  ```
  Prerequisite checking has failed. Click Back to select a different package,
  or click Cancel to exit.

  Failure messages are:

  Required feature wxs.primary is not found.
  ```

  Solution: The maintenance package that you install must match the type of installation. Download and apply the maintenance package that applies to your installation type.
- Linux **Problem:** The installation hangs.
  Solution: Sometimes, when installing WebSphere eXtreme Scale on Linux as a non-root user, the installer can hang. This is likely because the maximum number of open files is set too low on your Linux operating system. You will need to raise the allowed limit in the /etc/limits.conf or /etc/security/limits.conf file (where the file is located depends on your specific Linux distribution) to at least 8192.

Related tasks:
Using the Update Installer to install maintenance packages

Troubleshooting client connectivity

There are several common problems specific to clients and client connectivity that you can solve as described in the following sections.

Procedure

- **Problem:** If you are using the EntityManager API or byte array maps with the COPY_TO_BYTES copy mode, client data access methods result in various serialization-related exceptions or a NullPointerException exception.
  - The following error occurs when you are using the COPY_TO_BYTES copy mode:

    ```
    java.lang.NullPointerException
            at com.ibm.ws.objectgrid.map.BaseMap$BaseMapObjectTransformer2.inflateObject(BaseMap.java:5278)
            at com.ibm.ws.objectgrid.map.BaseMap$BaseMapObjectTransformer.inflateValue(BaseMap.java:5155)
    ```

  - The following error occurs when you are using the EntityManager API:

    ```
    java.lang.NullPointerException
                    at com.ibm.ws.objectgrid.em.GraphTraversalHelper.fluffFetchMD(GraphTraversalHelper.java:323)
                    at com.ibm.ws.objectgrid.em.GraphTraversalHelper.fluffFetchMD(GraphTraversalHelper.java:343)
                    at com.ibm.ws.objectgrid.em.GraphTraversalHelper.getObjectGraph(GraphTraversalHelper.java:102)
                    at
    com.ibm.ws.objectgrid.ServerCoreEventProcessor.getFromMap(ServerCoreEventProcessor.java:709)
                    at
    com.ibm.ws.objectgrid.ServerCoreEventProcessor.processGetRequest(ServerCoreEventProcessor.java:323)
    ```

  Cause: The EntityManager API and COPY_TO_BYTES copy mode use a metadata repository that is embedded in the data grid. When clients connect, the data grid stores the repository identifiers in the client and caches the identifiers for the duration of the client connection. If you restart the data grid, you

lose all metadata and the regenerated identifiers do not match the cached identifiers on the client.
Solution: If you are using the EntityManager API or the COPY_TO_BYTES copy mode, disconnect and reconnect all of the clients if the ObjectGrid is stopped and restarted. Disconnecting and reconnecting the clients refreshes the metadata identifier cache. You can disconnect clients by using the ObjectGridManager.disconnect method or the ObjectGrid.destroy method.

- **Problem:** The client hangs during a getObjectGrid method call.
  A client might seem to hang when calling the getObjectGrid method on the ObjectGridManager or throw an exception: com.ibm.websphere.projector.MetadataException. The EntityMetadata repository is not available and the timeout threshold is reached.

  Cause: The reason is the client is waiting for the entity metadata on the ObjectGrid server to become available.

  Solution: This error can occur when a container server has been started, but placement has not yet started. Take the following actions:
  - Examine the deployment policy for the ObjectGrid and verify that the number of active containers is greater than or equal to both the numInitialContainers and minSyncReplicas attributes in the deployment policy descriptor file.
  - Examine the setting for the placementDeferralInterval property in the container server server properties file to see how much time needs to pass before placement operations occur.
  - If you used the **xscmd -c suspendBalancing** command to stop the balancing of shards for a specific data grid and map set, use the **xscmd -c resumeBalancing** to start balancing again.

Related concepts:
Creating ObjectGrid instances with the ObjectGridManager interface

Troubleshooting cache integration

Use this information to troubleshoot issues with your cache integration configuration, including HTTP session and dynamic cache configurations.

Procedure

- **Problem:** HTTP session IDs are not being reused.
  Cause: You can reuse session IDs. If you create a data grid for session persistence in Version 7.1.1 or later, session ID reuse is automatically enabled. However, if you created prior configurations, this setting might already be set with the wrong value.

  Solution: Check the following settings to verify that you have HTTP session ID reuse enabled:
  - The `reuseSessionId` property in the splicer.properties file must be set to `true`.
  - The `HttpSessionIdReuse` custom property value must be set to `true`. This custom property might be set on one of the following paths in the WebSphere® Application Server administrative console:
    - Servers > *server_name* > Session management > Custom properties
    - Dynamic clusters > *dynamic_cluster_name* > Server template > Session management > Custom properties
    - Servers > Server Types > WebSphere application servers > *server_name*, and then, under Server Infrastructure, click Java and process management > Process definition > Java virtual machine > Custom properties
    - Servers > Server Types > WebSphere application servers > *server_name* > Web container settings > Web container

  If you update any custom property values, reconfigure eXtreme Scale session management so the splicer.properties file becomes aware of the change.
- **Problem:** When you are using a data grid to store HTTP sessions and the transaction load is high, a CWOBJ0006W message displays in the SystemOut.log file.

  ```
  CWOBJ0006W: An exception occurred:
  com.ibm.websphere.objectgrid.ObjectGridRuntimeException:
  java.util.ConcurrentModificationException
  ```

  This message occurs only when the replicationInterval parameter in the splicer.properties file is set to a value greater than zero and the Web application modifies a List object that was set as an attribute on the HTTPSession.

  Solution: Clone the attribute that contains the modified List object and put the cloned attribute into the session object.

Related tasks:
Configuring HTTP session managers
Configuring the HTTP session manager with WebSphere Application Server
Configuring WebSphere Application Server HTTP session persistence to a data grid
Configuring HTTP session manager with WebSphere Portal
Configuring the HTTP session manager for various application servers
Related reference:
XML files for HTTP session manager configuration
Servlet context initialization parameters
splicer.properties file

Troubleshooting the JPA cache plug-in

Use this information to troubleshoot issues with your JPA cache plug-in configuration. These problems can occur in both Hibernate and OpenJPA configurations.

Procedure

- **Problem:** The following exception displays: `CacheException: Failed to get ObjectGrid server.`
  With either an `EMBEDDED` or `EMBEDDED_PARTITION` ObjectGridType attribute value, the eXtreme Scale cache tries to obtain a server instance from the run time. In a Java™ Platform, Standard Edition environment, an eXtreme Scale server with embedded catalog service is started. The embedded catalog service tries to listen to port 2809. If that port is being used by another process, the error occurs.

  Solution: If external catalog service endpoints are specified, for example, with the objectGridServer.properties file, this error occurs if the host name or port is specified incorrectly. Correct the port conflict.

- **Problem:** The following exception displays: `CacheException: Failed to get REMOTE ObjectGrid for configured REMOTE ObjectGrid. objectGridName = [ObjectGridName], PU name = [persistenceUnitName]`
  This error occurs because the cache cannot get the ObjectGrid instance from the provided catalog service end points.

  Solution: This problem typically occurs because of an incorrect host name or port.

- **Problem:** The following exception displays: `CacheException: Cannot have two PUs [persistenceUnitName_1, persistenceUnitName_2] configured with same ObjectGridName [ObjectGridName] of EMBEDDED ObjectGridType`
  This exception results if you have many persistence units configured and the eXtreme Scale caches of these units are configured with the same ObjectGrid name and `EMBEDDED` ObjectGridType attribute value. These persistence unit configurations could be in the same or different persistence.xml files.

  Solution: You must verify that the ObjectGrid name is unique for each persistence unit when the ObjectGridType attribute value is `EMBEDDED`.

- **Problem:** The following exception displays: `CacheException: REMOTE ObjectGrid [ObjectGridName] does not include required BackingMaps [mapName_1, mapName_2,...]`
  With a REMOTE ObjectGrid type, if the obtained client-side ObjectGrid does not have complete entity backing maps to support the persistence unit cache, this exception occurs. For example, five entity classes are listed in the persistence unit configuration, but the obtained ObjectGrid only has two BackingMaps. Even though the obtained ObjectGrid might have 10 BackingMaps, if any one of the five required entity BackingMaps are not found in the 10 backing maps, this exception still occurs.

  Solution: Make sure that your backing map configuration supports the persistence unit cache.

Troubleshooting IBM eXtremeMemory

Use the following information to troubleshoot eXtremeMemory.

Procedure

Problem: If the shared resource, libstdc++.so.5, is not installed, then when you start the container server, IBM eXtremeMemory native libraries do not load.
Linux **Symptom:** On a Linux 64-bit operating system, if you try to start a container server with the enableXM server property set to `true`, and the `libstdc++.so.5` shared resource is not installed, you get an error similar to the following example:

```
00000000 Initialization W CWOBJ0006W: An exception occurred: java.lang.reflect.InvocationTargetException
        at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
        at sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:56)
        at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:39)
        at java.lang.reflect.Constructor.newInstance(Constructor.java:527)
        at com.ibm.websphere.objectgrid.server.ServerFactory.initialize(ServerFactory.java:350)
        at com.ibm.websphere.objectgrid.server.ServerFactory$2.run(ServerFactory.java:303)
        at java.security.AccessController.doPrivileged(AccessController.java:202)
        at com.ibm.websphere.objectgrid.server.ServerFactory.getInstance(ServerFactory.java:301)
        at com.ibm.ws.objectgrid.InitializationService.main(InitializationService.java:302)

Caused by: com.ibm.websphere.objectgrid.ObjectGridRuntimeException: java.lang.UnsatisfiedLinkError:
        OffheapMapdbg (Not found in java.library.path)
        at com.ibm.ws.objectgrid.ServerImpl.<init&gt;(ServerImpl.java:1033)
... 9 more Caused by: java.lang.UnsatisfiedLinkError: OffheapMapdbg (Not found in java.library.path)
        at java.lang.ClassLoader.loadLibraryWithPath(ClassLoader.java:1011)
        at java.lang.ClassLoader.loadLibraryWithClassLoader(ClassLoader.java:975)
 at java.lang.System.loadLibrary(System.java:469)
        at com.ibm.ws.objectgrid.io.offheap.ObjectGridHashTableOH.initializeNative(ObjectGridHashTableOH.java:112)
        at com.ibm.ws.objectgrid.io.offheap.ObjectGridHashTableOH.<clinit&gt;(ObjectGridHashTableOH.java:87)
        at java.lang.J9VMInternals.initializeImpl(Native Method)
        at java.lang.J9VMInternals.initialize(J9VMInternals.java:200)
        at com.ibm.ws.objectgrid.ServerImpl.<init&gt;(ServerImpl.java:1028)
... 9 more
```

Cause: The shared resource `libstdc++.so.5` has not been installed.

Diagnosing the problem: To verify that the resource `libstdc++.so.5` is installed, issue the following command from the ObjectGrid/native directory of your installation:

```
ldd libOffheapMap.so
```

If you do not have the shared library installed, you get the following error:

```
ldd libOffheapMap.so
libstdc++.so.5 => not found
```

Resolving the problem: Use the package installer of your 64-bit Linux distribution to install the required resource file. The package might be listed as `compat-libstdc++-33.x86_64` or `libstdc++5`. After installing the required resource, verify that the `libstdc++5`package is installed by issuing the following command from the ObjectGrid directory of your installation:

```
ldd libOffheapMap.so
```

Troubleshooting administration

Use the following information to troubleshoot administration, including starting and stopping servers, using the **xscmd** utility, and so on.

Procedure

- **Problem:** Administration scripts are missing from the *profile_root*/bin directory of a WebSphere® Application Server installation.
  Cause: When you update the installation, new script files do not automatically get installed in the profiles.

  Solution: If you want to run a script from your *profile_root*/bin directory, unaugment and reaugment the profile with the latest release. For more information, see Unaugmenting a profile using the command prompt and Creating and augmenting profiles for WebSphere eXtreme Scale.

- **Problem:** When you are running a **xscmd** command, the following message is printed to the screen:

  ```
  java.lang.IllegalStateException: Placement service MBean not available.
  []
          at
  com.ibm.websphere.samples.objectgrid.admin.OGAdmin.main(OGAdmin.java:1449)
          at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
          at
  sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:60)
          at
  sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:37)
          at java.lang.reflect.Method.invoke(Method.java:611)
          at com.ibm.ws.bootstrap.WSLauncher.main(WSLauncher.java:267)
  Ending at: 2011-11-10 18:13:00.000000484
  ```

  Cause: A connection problem occurred with the catalog server.

  Solution: Verify that your catalog servers are running and are available through the network. This message can also occur when you have a catalog service domain defined, but less than two catalog servers are running. The environment is not available until two catalog servers are started.

- 8.5+ **Problem:** When you are running a **xscmd** command, the following message is printed to the screen:

  ```
  CWXSI0066E: Unmatched argument argument_name was detected.
  ```

  Cause: You entered a command format that the **xscmd** utility did not recognize.

  Solution: Check the format of the command. You might encounter this issue when running regular expressions with the **-c findbyKey** command. For more information, see Querying and invalidating data.

Related concepts:
Example: Configuring catalog service domains
Administering

Troubleshooting multiple data center configurations

Use this information to troubleshoot multiple data center configurations, including linking between catalog service domains.

Before you begin

You must use the **xscmd** utility to troubleshoot your multiple data center configurations. For more information, see Administering with the xscmd utility.

Procedure

- **Problem:** Data is missing in one or more catalog service domains. For example, you might run the **xscmd -c establishLink** command. When you look at the data for each linked catalog service domain, the data looks different, for example from the **xscmd -c showMapSizes** command.
  Solution: You can troubleshoot this problem with the **xscmd -c showLinkedPrimaries** command. This command prints each primary shard, and including which foreign primaries are linked.

  In the described scenario, you might discover from running the **xscmd -c showLinkedPrimaries** command that the first catalog service domain primary shards are linked to the second catalog service domain primary shards, but the second catalog service domain does not have links to the first catalog service domain. You might consider rerunning the **xscmd -c establishLink** command from the second catalog service domain to the first catalog service domain.

- **Problem:** The catalog service domains are not replicating data. The output of the command **showMapsizes** or **showDomainReplicationState** do not match between the catalog service domains as expected. The command **showLinkedPrimaries** shows links in the recovery state instead of the online state.
  Diagnosis: Investigate the multi-master links between the primary shards in the recovery state. The recovery state indicates that WebSphere® eXtreme Scale cannot successfully replicate between the primary shards in each catalog service domain. When a primary shard encounters an exception, it goes into an auto-recovery state and sends a ping to the foreign primary shard. If the ping is successful, replication starts again. If the ping fails, the primary

shard sleeps and pings again in the future. Each primary shard is responsible for maintaining replication with its foreign primary in the foreign domain. For example, the primary shard for partition 1 in domain 1 replicates directly with the primary shard for partition 1 in domain 2.

1. Review the output for the command **showLinkedPrimaries** and locate a shard in recovery state. Example output:

```
CWXSI0068I: Executing command: showLinkedPrimaries
CWXSI0091I: Verifying the primary shards have the correct number of links to foreign primary shards.

*** Displaying results for inventory data grid and aSet map set. Expected number of online links: 1.

*** Listing Primary Shards with the incorrect number of links for local domain:
domain1, Container: server0_C-0, Server: server0,
Host: myHost.rchland.ibm.com ***

Grid Name Map Set Name Partition Domain  Container      Status
--------- ------------ --------- ------  ---------      -------
inventory aSet         0         domain2 server20_C-1   recovery
inventory aSet         1         domain2 server20_C-1   recovery
```

2. Review the SystemOut or JVM logs and FFDC of a link in recovery state. In the showLinkedPrimaries example that is provided, take note of the first entry, that is partition `0`, for the grid inventory and map set `aSet`. The local primary shard for partition `0` runs on `server0` and the foreign primary shard for partition `0` runs on `server20`. To find out more information about the link, locate the SystemOut or JVM log file for `server0`. Search the file for the inventory grid for partition `0`. To aid in the search, the shard identification string is formatted as `objectGridName:mapSetName:partitionID` in the log. In this case, the shard identification string is `inventory:aSet:0`. You should search for several messages in the `CWOBJ1500-CWOBJ1599` range. The relevant messages for this showLinkedPrimaries example include `CWOBJ1511I`, `CWOBJ1542I`, `CWOBJ1550W`and `CWOBJ1551I`.
Example log messages:

```
ReplicatedPar I   CWOBJ1511I: inventory:aSet:0 (primary) is open for business.
PrimaryShardI I   CWOBJ1542I: Primary inventory:aSet:0 started or continued replicating
from foreign primary (domain2:server20_C-1). Replicating for maps: [movie, book]
PrimaryShardI W   CWOBJ1550W:
The primary (inventory:aSet:0) shard received exceptions while replicating from the primary shard on
the domain2:server20_C-1 primary container.
The primary shard continues to poll the primary shard.
Exception received: org.omg.CORBA.NO_RESPONSE: Request 180 timed out vmcid: IBM minor code:B01 completed:
Maybe
at com.ibm.rmi.iiop.Connection.getCallStream(Connection.java:2339)
at com.ibm.rmi.iiop.Connection.send(Connection.java:2266)
at com.ibm.rmi.iiop.ClientRequestImpl.invoke(ClientRequestImpl.java:330)
at com.ibm.rmi.corba.ClientDelegate.invoke(ClientDelegate.java:445)
at com.ibm.CORBA.iiop.ClientDelegate.invoke(ClientDelegate.java:1193)
at com.ibm.rmi.corba.ClientDelegate.invoke(ClientDelegate.java:800)
at com.ibm.CORBA.iiop.ClientDelegate.invoke(ClientDelegate.java:1223)
at org.omg.CORBA.portable.ObjectImpl._invoke(ObjectImpl.java:484)
at com.ibm.ws.objectgrid.partition._IDLPrimaryShardStub.queryRevision(_IDLPrimaryShardStub.java:420)
at
com.ibm.ws.objectgrid.partition.IDLPrimaryShardWrapperImpl.queryRevision(IDLPrimaryShardWrapperImpl.java:96)
at com.ibm.ws.objectgrid.replication.PrimaryShardImpl$RevisionQueryHandler.run(PrimaryShardImpl.java:4209)
at java.util.concurrent.ThreadPoolExecutor$Worker.runTask(ThreadPoolExecutor.java:886)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:908)
at com.ibm.ws.objectgrid.thread.XSThreadPool$Worker.run(XSThreadPool.java:309)
CWOBJ1551I: Primary inventory:aSet:0successfully recovered and replicated after several exceptions from the
primary on domain2:server20_C-1.

If the primary shard automatically recovers, a CWOBJ1551I message occurs.
```

- **Problem**: The multimaster replication link was dismissed, but the foreign domain or collective could not be contacted. The link is in the DISMISSING_LINK state in the monitoring console, or the link is displayed in the DISMISSING_LINK state when you run the `xscmd -c showLinkedDomains -v` command. The foreign domain or collective cannot be restarted or contacted to resolve the dismiss link request. The link stays in DISMISSING_LINK state because the local domain tries again to connect to the foreign domain to complete the dismissal request.
Solution: Run the `xscmd -c dismissLink` command with the -force option to dismiss the link once with the foreign domain and then clean up the local domain.

Related concepts:
Planning multiple data center topologies
JPA level 2 (L2) cache plug-in
Related tasks:
Configuring multiple data center topologies
Configuring the OpenJPA cache plug-in
Configuring the Hibernate cache plug-in
Related reference:
JPA cache configuration properties for both OpenJPA and Hibernate Version 3.0
Example: OpenJPA ObjectGrid XML files
Example: Hibernate ObjectGrid XML files
Related information:
dW Improve response time and data availability with WebSphere eXtreme Scale multi-master capability
com.ibm.websphere.objectgrid.openJPA package
com.ibm.websphere.objectgrid.hibernate.cache package

Troubleshooting loaders

Use this information to troubleshoot issues with your database loaders.

Procedure

- **Problem:** The loader is unable to communicate with the database. A LoaderNotAvailableException exception occurs.
  Explanation: The loader plug-in can fail when it is unable to communicate to the database back end. This failure can happen if the database server or the network connection is down. The write-behind loader queues the updates and tries to push the data changes to the loader periodically. The loader must notify the ObjectGrid run time that there is a database connectivity problem by throwing a LoaderNotAvailableException exception.

  Solution: The Loader implementation must be able to distinguish a data failure or a physical loader failure. Data failure should be thrown or rethrown as a LoaderException or an OptimisticCollisionException, but a physical loader failure must be thrown or rethrown as a LoaderNotAvailableException. ObjectGrid handles these two exceptions differently:
  - If a LoaderException is caught by the write-behind loader, the write-behind loader considers the exception a failure, such as duplicate key failure. The write-behind loader unbatches the update, and tries the update one record at one time to isolate the data failure. If A {{LoaderException}}is caught again during the one record update, a failed update record is created and logged in the failed update map.
  - If a LoaderNotAvailableException is caught by the write-behind loader, the write-behind loader considers it failed because it cannot connect to the database end, for example, the database back-end is down, a database connection is not available, or the network is down. The write-behind loader waits for 15 seconds and then try the batch update to the database again.

  The common mistake is to throw a LoaderException while a LoaderNotAvailableException must be thrown. All the records queued in the write-behind loader become failed update records, which defeats the purpose of back-end failure isolation.
- **Problem:** When you are using an OpenJPA loader with DB2® in WebSphere® Application Server, a closed cursor exception occurs.
  The following exception is from DB2 in the org.apache.openjpa.persistence.PersistenceException log file:

  ```
  [jcc][t4][10120][10898][3.57.82] Invalid operation: result set is closed.
  ```

  Solution: By default, the application server configures the resultSetHoldability custom property with a value of `2` (CLOSE_CURSORS_AT_COMMIT). This property causes DB2 to close its resultSet/cursor at transaction boundaries. To remove the exception, change the value of the custom property to `1` (HOLD_CURSORS_OVER_COMMIT). Set the resultSetHoldability custom property on the following path in the WebSphere Application Server cell: Resources > JDBC provider > DB2 Universal JDBC Driver Provider > DataSources > *data_source_name* > Custom properties > New.

- **Problem** DB2 displays an exception: `The current transaction has been rolled back because of a deadlock or timeout. Reason code "2".. SQLCODE=-911, SQLSTATE=40001, DRIVER=3.50.152`
  This exception occurs because of a lock contention problem when you are running with OpenJPA with DB2 in WebSphere Application Server. The default isolation level for WebSphere Application Server is Repeatable Read (RR), which obtains long-lived locks with DB2.

  Solution:
  Set the isolation level to Read Committed to reduce the lock contention. Set the webSphereDefaultIsolationLevel data source custom property to set the isolation level to `2`(TRANSACTION_READ_COMMITTED) on the following path in the WebSphere Application Server cell: Resources > JDBC provider > *JDBC_provider* > Data sources > *data_source_name* > Custom properties > New. For more information about the webSphereDefaultIsolationLevel custom property and transaction isolation levels, see Requirements for setting data access isolation levels.

- **Problem:** When you are using the preload function of the JPALoader or JPAEntityLoader, the following CWOBJ1511 message does not display for the partition in a container server: `CWOBJ1511I: GRID_NAME:MAPSET_NAME:PARTITION_ID (primary) is open for business.`
  Instead, a TargetNotAvailableException exception occurs in the container server, which activates the partition that is specified by the preloadPartition property.

  Solution: Set the preloadMode attribute to `true` if you use a JPALoader or JPAEntityLoader to preload data into the map. If the preloadPartition property of the JPALoader and JPAEntityLoader is set to a value between `0` and `total_number_of_partitions - 1`, then the JPALoader and JPAEntityLoader try to preload the data from backend database into the map. The following snippet of code illustrates how the preloadMode attribute is set to enable asynchronous preload:

  ```
  BackingMap bm = og.defineMap( "map1" );
  bm.setPreloadMode( true );
  ```

  You can also set the preloadMode attribute by using an XML file as illustrated in the following example:

  ```
  <backingMap name="map1" preloadMode="true" pluginCollectionRef="map1"
                lockStrategy="OPTIMISTIC" />
  ```

Related concepts:
Programming for JPA integration
Configuring cache integration
Related tasks:
Configuring JPA loaders

Troubleshooting XML configuration

When you configure eXtreme Scale, you can encounter unexpected behavior with your XML files. The following sections describe problems that can occur and solutions.

Procedure

- **Problem:** Your deployment policy and ObjectGrid XML files must match.
  The deployment policy and ObjectGrid XML files must match. If they do not have matching ObjectGrid names and map names, errors occur.

If the backingMap list in an ObjectGrid XML file does not match the map references list in a deployment policy XML file, an error occurs on the catalog server.

For example, the following ObjectGrid XML file and deployment policy XML file are used to start a container process. The deployment policy file has more map references than are listed in the ObjectGrid XML file.

```
ObjectGrid.xml - incorrect example

<?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="accounting">
            <backingMap name="payroll" readOnly="false" />
        </objectGrid>
    </objectGrids>
</objectGridConfig>

deploymentPolicy.xml - incorrect example

<?xml version="1.0" encoding="UTF-8"?>
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy ../deploymentPolicy.xsd"
 xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">
    <objectgridDeployment objectgridName="accounting">
        <mapSet name="mapSet1" numberOfPartitions="4" minSyncReplicas="1"
                                    maxSyncReplicas="2" maxAsyncReplicas="1">
            <map ref="payroll"/>
            <map ref="ledger"/>
        </mapSet>
    </objectgridDeployment>
</deploymentPolicy>
```

Messages: An error message occurs in the SystemOut.log file when the deployment policy is incompatible with the ObjectGrid XML file. For the preceding example, the following message occurs:

```
CWOBJ3179E: The map ledger reference in the mapSet mapSet1 of ObjectGrid accounting
deployment descriptor file does not reference a valid backing map from the ObjectGrid
XML.
```

If the deployment policy is missing map references to backingMaps that are listed in the ObjectGrid XML file, an error message occurs in the SystemOut.log file. For example:

```
CWOBJ3178E: The map ledger in ObjectGrid accounting referenced in the ObjectGrid XML
was not found in the deployment descriptor file.
```

Solution: Determine which file has the correct list and alter the relevant code accordingly.

- **Problem:** Incorrect ObjectGrid names between XML files also causes and error.
  The name of the ObjectGrid is referenced in both the ObjectGrid XML file and the deployment policy XML file.

  Message: An ObjectGridException occurs with a caused by exception of IncompatibleDeploymentPolicyException. An example follows.

  Caused by: com.ibm.websphere.objectgrid.IncompatibleDeploymentPolicyException: The objectgridDeployment with objectGridName "accountin" does not have a corresponding objectGrid in the ObjectGrid XML.

  The ObjectGrid XML file is the master list of ObjectGrid names. If a deployment policy has an ObjectGrid name that is not contained in the ObjectGrid XML file, an error occurs.

  Solution: Verify details such as the spelling of the ObjectGrid name. Remove any extra names, or add missing ObjectGrid names, to the ObjectGrid XML or deployment policy XML files. In the example message, the objectGridName is misspelled as "accountin" instead of "accounting".

- **Problem:** Some of the attributes in the XML file can only be assigned certain values. These attributes have acceptable values enumerated by the schema. The following list provides some of the attributes:
  - authorizationMechanism attribute on the objectGrid element
  - copyMode attribute on the backingMap element
  - lockStrategy attribute on the backingMap element
  - ttlEvictorType attribute on the backingMap element
  - type attribute on the property element
  - initialState on the objectGrid element
  - evictionTriggers on the backingMap element

  If one of these attributes is assigned an invalid value, XML validation fails. In the following example XML file, an incorrect value of INVALID_COPY_MODE is used:

```
INVALID_COPY_MODE example
<?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="accounting">
            <backingMap name="payroll" copyMode="INVALID_COPY_MODE"/>
        <objectGrid/>
   </objectGrids>
</objectGridConfig>
```

The following message appears in the log.

```
CWOBJ2403E: The XML file is invalid. A problem has been detected
with < null > at line 5. The error message is cvc-enumeration-valid:
Value 'INVALID_COPY_MODE' is not facet-valid with respect to enumeration
'[COPY_ON_READ_AND_COMMIT, COPY_ON_READ, COPY_ON_WRITE, NO_COPY, COPY_TO_BYTES]'.
It must be a value from the enumeration.
```

- **Problem:** Missing or incorrect attributes or tags in an XML file causes errors, such as the following example in which the ObjectGrid XML file is missing the closing < /objectGrid > tag:

```
missing attributes - example 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="accounting">
            <backingMap name="payroll" />

   </objectGrids>
</objectGridConfig>
```

  Message:

```
CWOBJ2403E: The XML file is invalid. A problem has been detected with
< null > at line 7. The error message is The end-tag for element type "objectGrid"
must end with a '>' delimiter.
```

  An ObjectGridException about the invalid XML file occurs with the name of the XML file.

  Solution: Ensure that the necessary tags and attributes appear in your XML files with correct format.

- **Problem:** If an XML file is formatted with incorrect or missing syntax, the CWOBJ2403E appears in the log. For example, the following message is displayed when a quotation is missing on one of the XML attributes

```
CWOBJ2403E: The XML file is invalid. A problem has been detected with
< null > at line 7. The error message is Open quote is expected for attribute
"maxSyncReplicas" associated with an element type "mapSet".
```

  An ObjectGridException about the invalid XML file also occurs.

  Solution: Various solutions can be used for a given XML syntax error. Consult relevant documentation about XML script writing.

- **Problem:** Referencing a nonexistent plug-in collection causes an XML file to be invalid. For example, when using XML to define BackingMap plug-ins, the pluginCollectionRef attribute of the backingMap element must reference a backingMapPluginCollection. The pluginCollectionRef attribute must match the backingMapPluginCollection elements.
  Message:

  If the pluginCollectionRef attribute does not match any ID attributes of any of the backingMapPluginConfiguration elements, the following message, or one that is similar, is displayed in the log.

```
[7/14/05 14:02:01:971 CDT] 686c060e XmlErrorHandl E CWOBJ9002E:
This is an English only Error message: Invalid XML file. Line: 14; URI:
null; Message: Key 'pluginCollectionRef' with
value 'bookPlugins' not found for identity constraint of
element 'objectGridConfig'.
```

  The following XML file is used to produce the error. Notice that the name of the BackingMap book has its pluginCollectionRef attribute set to bookPlugins, and the single backingMapPluginCollection has an ID of collection1.

```
referencing a non-existent attribute XML - example

<?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="bookstore">
         <backingMap name="book" pluginCollectionRef="bookPlugin" />
      </objectGrid>
   </objectGrids>
   <backingMapPluginCollections>
    <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsxmlprob_collection1">
     <bean id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_txsxmlprob_Evictor"
       className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" />
     </backingMapPluginCollection>
    </backingMapPluginCollections>
</objectGridConfig>
```

  Solution:

  To fix the problem, ensure that the value of each pluginCollectionRef matches the ID of one of the backingMapPluginCollection elements. Simply change the name of pluginCollectionRef to collection1 to not receive this error. Alternatively, change the ID of the existing backingMapPluginCollection to match the pluginCollectionRef, or add an additional backingMapPluginCollection with an ID that matches the pluginCollectionRef to correct the error.

- **Problem:** The IBM® Software Development Kit (SDK) Version 5 contains an implementation of some Java™ API for XML Processing (JAXP) function to use for XML validation against a schema. When using an SDK that does not contain this implementation, attempts to validate might fail.
  When you attempt to validate XML with an SDK that does not have the necessary implementation, the log contains the following error:

  ```
  XmlConfigBuild XML validation is enabled
  SystemErr R com.ibm.websphere.objectgrid
  SystemErr R at com.ibm.ws.objectgrid.ObjectGridManagerImpl.getObjectGridConfigurations
          (ObjectGridManagerImpl.java:182)
  SystemErr R at com.ibm.ws.objectgrid.ObjectGridManagerImpl.createObjectGrid(ObjectGridManagerImpl.java:309)
  SystemErr R at com.ibm.ws.objectgrid.test.config.DocTest.main(DocTest.java:128)
  SystemErr R Caused by: java.lang.IllegalArgumentException: No attributes are implemented
  SystemErr R at org.apache.crimson.jaxp.DocumentBuilderFactoryImpl.setAttribute(DocumentBuilderFactoryImpl.java:93)
  SystemErr R at com.ibm.ws.objectgrid.config.XmlConfigBuilder.<init>XmlConfigBuilder.java:133)
  SystemErr R at com.ibm.websphere.objectgrid.ProcessConfigXML$2.runProcessConfigXML.java:99)...
  ```

  The SDK that is used does not contain an implementation of JAXP function that is necessary to validate XML files against a schema.

  Solution: If you want to validate XML by using an SDK that does not contain JAXP implementation, download Apache Xerces, and include its Java archive (JAR) files in the classpath. To avoid this problem, after you download Xerces and include the JAR files in the classpath, you can validate the XML file successfully.

Related reference:
Configuration files
ObjectGrid descriptor XML file
Deployment policy descriptor XML file
Entity metadata descriptor XML file
Security descriptor XML file
Client properties file
Spring descriptor XML file

Troubleshooting deadlocks

The following sections describe some of the most common deadlock scenarios and suggestions on how to avoid them.

Before you begin

Implement exception handling in your application. For more information, see Implementing exception handling in locking scenarios for Java applications .
The following exception displays as a result:

```
com.ibm.websphere.objectgrid.plugins.LockDeadlockException: Message
```

This message represents the string that is passed as a parameter when the exception is created and thrown.

Procedure

- **Problem:**A LockTimeoutException exception occurs.
  Description: When a transaction or client asks for a lock to be granted for a specific map entry, the request often waits for the current client to release the lock before the request is submitted. If the lock request remains idle for an extended time, and a lock is never granted, LockTimeoutException exception is created to prevent a deadlock, which is described in more detail in the following section. You are more likely to see this exception when you configure a pessimistic locking strategy, because the lock never releases until the transaction commits.

  Retrieve more details:

  - The LockTimeoutException exception contains the getLockRequestQueueDetails method, which returns a string. You can use this method to see a detailed description of the situation that triggers the exception. The following is an example of code that catches the exception, and displays an error message.

    ```
    try {
        ...
    }
    catch (LockTimeoutException lte) {
        System.out.println(lte.getLockRequestQueueDetails());
    }
    ```

    If you receive the exception in an ObjectGridException exception catch block, the following code determines the exception and displays the queue details. It also uses the findRootCause utility method.

    ```
    try {
    ...
    }
    catch (ObjectGridException oe) {
        Throwable Root = findRootCause( oe );
        if (Root instanceof LockTimeoutException) {
            LockTimeoutException lte = (LockTimeoutException)Root;
            System.out.println(lte.getLockRequestQueueDetails());
        }
    }
    ```

Solution: A LockTimeoutException exception prevents possible deadlocks in your application. An exception of this type results when the exception waits a set amount of time. You can set the amount of time that the exception waits by setting the lock timeout value. If a deadlock does not actually exist in your application, adjust the lock timeout to avoid the LockTimeoutException.

For more information about setting the lock timeout value, see Configuring the lock timeout value in the ObjectGrid descriptor XML file. You can also configure the timeout value programmatically:

- Configuring and implementing locking in Java applications

.

- **Problem:** A deadlock occurs on a single key.
  Description: The following scenarios describe how deadlocks can occur when a single key is accessed with an S lock and later updated. When this action occurs from two transactions simultaneously, a deadlock occurs.

Table 1. Single key deadlocks scenario

	Thread 1	**Thread 2**	
1	Start transaction	Start transaction	Each thread establishes an independent transaction.
2	get key1	get key1	S lock granted to both transactions for key1.
3	○ update key1		No U lock. Update performed in transactional cache.
4		○ update key1	No U lock. Update performed in the transactional cache
5	Commit transaction		Blocked: The S lock for key1 cannot be upgraded to an X lock because Thread 2 has an S lock.
6		Commit transaction	Deadlock: The S lock for key1 cannot be upgraded to an X lock because T1 has an S lock.

Table 2. Single key deadlocks, continued

	Thread 1	**Thread 2**	
1	Start transaction	Start transaction	Each thread establishes an independent transaction.
2	get key1		S lock granted for key1
3	○ getForUpdate key1	get key1	S lock is upgraded to a U lock for key1.
4		get key1	S lock granted for key1.
5		○ getForUpdate key1	Blocked: T1 already has U lock.
6	Commit transaction		Deadlock: The U lock for key1 cannot be upgraded.
7		Commit transaction	Deadlock: The S lock for key1 cannot be upgraded.

Table 3. Single key deadlocks, continued

	Thread 1	**Thread 2**	
1	Start transaction	Start transaction	Each thread establishes an independent transaction
2	get key1		S lock granted for key1.
3	○ getForUpdate key1		S lock is upgraded to a U lock for key1
4		get key1	S lock is granted for key1.
5		○ getForUpdate key1	Blocked: Thread 1 already has a U lock.
6	Commit transaction		Deadlock: The U lock for key1 cannot be upgraded to an X lock because Thread 2 has an S lock.

If the ObjectMap.getForUpdate is used to avoid the S lock, then the deadlock is avoided:

Table 4. Single key deadlocks, continued

	Thread 1	**Thread 2**	
1	Start transaction	Start transaction	Each thread establishes an independent transaction.
2	○ getForUpdate key1		U lock granted to thread 1 for key1.
3		○ getForUpdate key1	U lock request is blocked.
4	○ update key1	<blocked>	
5	Commit transaction	<blocked>	The U lock for key1 can be successfully upgraded to an X lock.
6		<released>	The U lock is finally granted to key1 for thread 2.
7		○ update key2	U lock granted to thread 2 for key2.
8		Commit transaction	The U lock for key1 can successfully be upgraded to an X lock.

Solutions:

- Use the getForUpdate method instead of get to acquire a U lock instead of an S lock.
- Use a transaction isolation level of read committed to avoid holding S locks. Reducing the transaction isolation level increases the possibility of non-repeatable reads. However, non-repeatable reads from one client are only possible if the transaction cache is explicitly invalidated by the same

client.
    - Use the optimistic locking strategy. Using the optimistic lock strategy requires handling optimistic collision exceptions.
- **Problem:** A deadlock occurs on ordered multiple keys.
  Description: This scenario describes what happens if two transactions attempt to update the same entry directly and hold S locks to other entries.

Table 5. Ordered multiple key deadlock scenario

	Thread 1	**Thread 2**	
1	Start transaction	Start transaction	Each thread establishes an independent transaction.
2	get key1	get key1	S lock granted to both transactions for key1.
3	get key2	get key2	S lock granted to both transactions for key2.
4	◦ update key1		No U lock. Update performed in transactional cache.
5		◦ update key2	No U lock. Update performed in transactional cache.
6.	Commit transaction		Blocked: The S lock for key 1 cannot be upgraded to an X lock because thread 2 has an S lock.
7		Commit transaction	Deadlock: The S lock for key 2 cannot be upgraded because thread 1 has an S lock.

You can use the ObjectMap.getForUpdate method to avoid the S lock, then you can avoid the deadlock:

Table 6. Ordered multiple key deadlock scenario, continued

	Thread 1	**Thread 2**	
1	Start transaction	Start transaction	Each thread establishes an independent transaction.
2	◦ getForUpdate key1		U lock granted to transaction T1 for key1.
3		◦ getForUpdate key1	U lock request is blocked.
4	get key2	<blocked>	S lock granted for T1 for key2.
5	◦ update key1	<blocked>	
6	Commit transaction	<blocked>	The U lock for key1 can be successfully upgraded to an X lock.
7		<released>	The U lock is finally granted to key1 for T2
8		get key2	S lock granted to T2 for key2.
9		◦ update key2	U lock granted to T2 for key2.
10		Commit transaction	The U lock for key1 can be successfully upgraded to an X lock.

Solutions:
- Use the getForUpdate method instead of the get method to acquire a U lock directly for the first key. This strategy works only if the method order is deterministic.
- Use a transaction isolation level of read committed to avoid holding S locks. This solution is the easiest to implement if the method order is not deterministic. Reducing the transaction isolation level increases the possibility of non-repeatable reads. However, non-repeatable reads are only possible if the transaction cache is explicitly invalidated.
- Use the optimistic locking strategy. Using the optimistic lock strategy requires handling optimistic collision exceptions.

- **Problem:** A deadlock occurs from an out of order U lock
  Description: If the order in which keys are requested cannot be guaranteed, then a deadlock can still occur.

Table 7. Out of order with U lock scenario

	Thread 1	**Thread 2**	
1	Start transaction	Start transaction	Each thread establishes an independent transaction.
2	◦ getForUpdate key1	◦ getForUpdate key2	U locks successfully granted for key1 and key2.
3	get key2	get key1	S lock granted for key1 and key2.
4	◦ update key1	◦ update key2	
5	Commit transaction		The U lock cannot be upgraded to an X lock because T2 has an S lock.
6		Commit transaction	The U lock cannot be upgraded to an X lock because T1 has an S lock.

Solutions:
- Wrap all work with a single global U lock (mutex). This method reduces concurrency, but handles all scenarios when access and order are non-deterministic.
- Use a transaction isolation level of read committed to avoid holding S locks. This solution is the easiest to implement if the method order is not deterministic and provides the greatest amount of concurrency. Reducing the transaction isolation level increases the possibility of non-repeatable reads. However, non-repeatable reads are only possible if the transaction cache is explicitly invalidated.
- Use the optimistic locking strategy. Using the optimistic lock strategy requires handling optimistic collision exceptions.

Related concepts:
Lock types

Troubleshooting security

Use this information to troubleshoot issues with your security configuration.

Procedure

- **Problem:** The client end of the connection requires Secure Sockets Layer (SSL), with the transportType setting set to `SSL-Required`. However, the server end of the connection does not support SSL, and has the transportType setting set to `TCP/IP`. As a result, the following exception gets chained to another exception in the log files:

  ```
  java.net.ConnectException: connect: Address is invalid on local machine, or
  port is not valid on remote machine
      at java.net.PlainSocketImpl.doConnect(PlainSocketImpl.java:389)
      at java.net.PlainSocketImpl.connectToAddress(PlainSocketImpl.java:250)
      at java.net.PlainSocketImpl.connect(PlainSocketImpl.java:237)
      at java.net.SocksSocketImpl.connect(SocksSocketImpl.java:385)
      at java.net.Socket.connect(Socket.java:540)
      at
  com.ibm.rmi.transport.TCPTransportConnection.createSocket(TCPTransportConnection.java:155)
      at
  com.ibm.rmi.transport.TCPTransportConnection.createSocket(TCPTransportConnection.java:167)
  ```

  The address in this exception could be a catalog server, container server, or client.
  Solution: See Configuring secure transport types for a table with the valid security configurations between clients and servers.

- When agent is used, the client sends the agent call to the server, and server sends the response back to the client to acknowledge the agent call. When the agent finishes processing, the server initiates a connection to send the agent results. This makes the container server a client from connect point of view. Therefore, if TLS or SSL is configured, make sure the client public certificate is imported in the server truststore.
- 8.5+ **Problem:** When users are authorized to access a WebSphere® eXtreme Scale data grid, those users might also be authorized to perform management operations using the **xscmd** command or the **stopOgServer** command. Most data grid deployers restrict administrative access to only a subset of the users who can access grid data.
  If you use the following command to access the data grid, you might also be authorized to perform administrative actions, such as listAllJMXAddresses:

  ```
  ./xscmd.sh -user <user> -password <password> <other_parameters>
  ```

  If this operation works for this user, then any **xscmd** operation might also be performed by the same user.
  Resolution: When eXtreme Scale components run with WebSphere Application Server, use the WebSphere Application Server administrative console to activate the security manager. Click Security > Global Security, and select the check boxes, Enable administrative security and Use Java 2 Security, to restrict application access to local resources.

  Access to the management operations is controlled by the WebSphere Application Server security manager and is granted only to the users who belong to the WebSphere Administrator role. The **xscmd** command must be run from the WebSphere Application Server directory.

  When eXtreme Scale components run in a stand-alone environment, additional steps are required to implement administrative security. You must run the catalog servers and container servers using the Java™ security manager, which requires a policy file.

  The policy file resembles the following example:
  Remember: There are typically MapPermission entries as well, 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 case, only the manager principal is authorized to do administrative operations using the **xscmd** command. Other lines can be added as necessary to give additional principals MBean permissions. A different type of principal is needed if you use LDAP authentication.

  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
  ```

  Windows

  ```
  startOGServer.bat <arguments> -jvmargs -Djava.security.auth.login.config=jaas.config -Djava.security.manager
  -Djava.security.policy="auth.policy" -Dobjectgrid.home=%OBJCTGRID_HOME%
  ```

  You specify -Djava.security.policy in this case, instead of -Djava.security.auth.policy.

Troubleshooting Liberty profile configurations

8.5+ Use this information to troubleshoot commonly experienced problems with the Liberty profile.

About this task

To help you identify and resolve problems, the product has a unified logging component. See Analyzing log and trace data.

Details of known restrictions that apply when using the Liberty profile are provided in the following two topics in the WebSphere® Application Server Information Center:

- Liberty profile: Runtime environment known restrictions
- Liberty profile: Developer Tools known restrictions

Procedure

- **Problem:** You experience problems that are not readily explained.
  Solution: Check that your Java™ development kits at Java Version 6 or later, and are at a current service level. See Minimum supported Java levels in the Liberty profile: Runtime environment known restrictions topic for more information.
- **Problem:** The following security error is displayed when you attempt to access an application that redirects to an SSL port, and the SSL port is not available:

  ```
  CWWKS9105E: Could not determine the SSL port for automatic redirection
  ```

  Solution: The port might not be available because of a missing SSL configuration or some error in the SSL configuration definition. Check the SSL configuration in the server.xml file to make sure that it exists and is correct.

IBM Support Assistant for WebSphere eXtreme Scale

You can use the IBM® Support Assistant to collect data, analyze symptoms, and access product information.

IBM Support Assistant Lite

IBM Support Assistant Lite for WebSphere® eXtreme Scale provides automatic data collection and symptom analysis support for problem determination scenarios.
IBM Support Assistant Lite reduces the amount of time it takes to reproduce a problem with the proper Reliability, Availability, and Serviceability tracing levels set (trace levels are set automatically by the tool) to streamline problem determination. If you need further assistance, IBM Support Assistant Lite also reduces the effort required to send the appropriate log information to IBM Support.

IBM Support Assistant Lite is included in each installation of WebSphere eXtreme Scale Version 7.1.0

IBM Support Assistant

IBM® Support Assistant (ISA) provides quick access to product, education, and support resources that can help you answer questions and resolve problems with IBM software products on your own, without needing to contact IBM Support. Different product-specific plug-ins let you customize IBM Support Assistant for the particular products you have installed. IBM Support Assistant can also collect system data, log files, and other information to help IBM Support determine the cause of a particular problem.

IBM Support Assistant is a utility to be installed on your workstation, not directly onto the WebSphere eXtreme Scale server system itself. The memory and resource requirements for the Assistant could negatively affect the performance of the WebSphere eXtreme Scale server system. The included portable diagnostic components are designed for minimal impact to the normal operation of a server.

You can use IBM Support Assistant to help you in the following ways:

- To search through IBM and non-IBM knowledge and information sources across multiple IBM products to answer a question or solve a problem
- To find additional information through product-specific Web resources; including product and support home pages, customer news groups and forums, skills and training resources and information about troubleshooting and commonly asked questions
- To extend your ability to diagnose product-specific problems with targeted diagnostic tools available in the Support Assistant
- To simplify collection of diagnostic data to help you and IBM resolve your problems (collecting either general or product/symptom-specific data)
- To help in reporting of problem incidents to IBM Support through a customized online interface, including the ability to attach the diagnostic data referenced above or any other information to new or existing incidents

Finally, you can use the built-in Updater facility to obtain support for additional software products and capabilities as they become available. To set up IBM Support Assistant for use with WebSphere eXtreme Scale, first install IBM Support Assistant using the files provided in the downloaded image from the IBM Support Overview Web page at: http://www-947.ibm.com/support/entry/portal/Overview/Software/Other_Software/IBM_Support_Assistant. Next, use IBM Support Assistant to locate and install any product updates. You can also choose to install plug-ins available for other IBM software in your environment. More information and the latest version of the IBM Support Assistant are available from the IBM Support Assistant Web page at: http://www.ibm.com/software/support/isa/.

Reference

	Use the reference information to quickly access configuration file descriptions, API information, messages, and more.

- ObjectGrid interface
  The following methods allow you to interact with an ObjectGrid instance.
- BackingMap interface
  Each ObjectGrid instance contains a collection of BackingMap objects. Use the defineMap method or the createMap method of the ObjectGrid interface to name and add each BackingMap to an ObjectGrid instance. These methods return a BackingMap instance that is then used to define the behavior of an individual Map. A BackingMap can be considered as an in-memory cache of committed data for an individual map.
- ExceptionMapper interface
  When a user plug-in implementation throws an exception, eXtreme Scale checks certain exceptions defined in the throws contract. However, sometimes an unchecked exception contains a contract exception or the exception does not observe the contract appropriately. Therefore a mechanism is necessary to map the exception to the contract exception if possible, such as ExceptionMapper.
- 8.5+ Regular expression syntax
  A regular expression is a structured string that is used to match other strings. You can use regular expressions for data invalidation.
- Configuration files
  WebSphere® eXtreme Scale is configured by a collection of XML and properties files.
- Messages
  When you encounter a message in a log or other parts of the product interface, you can look up the message by its component prefix to find out more information.
- User interface settings
  This reference information describes settings that you can view and configure on the pages of the WebSphere Application Serveradministrative console and elsewhere.

ObjectGrid interface

The following methods allow you to interact with an ObjectGrid instance.

Create and initialize

See the ObjectGridManager interface topic for the required steps for creating an ObjectGrid instance. Two distinct methods exist to create an ObjectGrid instance: programmatically or with XML configuration files. See ObjectGridManager interface for more information.

Get or set and factory methods

Any set methods must be called before you initialize the ObjectGrid instance. If you call a set method after the initialize method is called, a java.lang.IllegalStateException exception results. Each of the getSession methods of the ObjectGrid interface also implicitly call the initialize method. Therefore, you must call the set methods before calling any of the getSession methods. The only exception to this rule is with the setting, adding, and removing of ObjectGridEventListener objects. These objects are allowed to be processed after the initialization processing has completed.

The ObjectGrid interface contains the following major methods.

Table 1. ObjectGrid interface. Major methods of ObjectGrid.

Method	Description
BackingMap defineMap(String name);	defineMap: is a factory method to define a uniquely named BackingMap. For more information about backing maps, see BackingMap interface.
BackingMap getMap(String name);	getMap: Returns a BackingMap previously defined by calling defineMap. By using this method, you can configure the BackingMap, if it is not already configured through XML configuration.
BackingMap createMap(String name);	createMap: Creates a BackingMap, but does not cache it for use by this ObjectGrid. Use this method with the setMaps(List) method of the ObjectGrid interface, which caches BackingMaps for use with this ObjectGrid. Use these methods when you are configuring an ObjectGrid with the Spring Framework.
void setMaps(List mapList);	setMaps: Clears any BackingMaps that have been previously defined on this ObjectGrid and replaces them with the list of BackingMaps that is provided.
public Session getSession() throws ObjectGridException, TransactionCallbackException;	getSession: Returns a Session, which provides begin, commit, rollback functionality for a Unit of Work. For more information about Session objects, see Session interface.
Session getSession(CredentialGenerator cg);	getSession(CredentialGenerator cg): Get a session with a CredentialGenerator object. This method can only be called by the ObjectGrid client in a client server environment.
Session getSession(Subject subject);	getSession(Subject subject): Allows the use of a specific Subject object rather than the one configured on the ObjectGrid to get a Session.
void initialize() throws ObjectGridException;	initialize: ObjectGrid is initialized and available for general use. This method is called implicitly when the getSession method is called, if the ObjectGrid is not in an initialized state.
void destroy();	destroy: The framework is disassembled and cannot be used after this method is called.

Method	Description
void setTxTimeout(int timeout);	setTxTimeout: Use this method to set the amount of time, in seconds, that a transaction that is started by a session that this ObjectGrid instance created is allowed for completion. If a transaction does not complete within the specified amount of time, the Session that started the transaction is marked as being "timed out". Marking a Session as timed out causes the next ObjectMap method that is invoked by the timed out Session to result in a com.ibm.websphere.objectgrid.TransactionTimeoutException exception. The Session is marked as rollback only, which causes the transaction to be rolled back even if the application calls the commit method instead of the rollback method after the TransactionTimeoutException exception is caught by the application. A timeout value of 0 indicates that the transaction is allowed unlimited amount of time to complete. The transaction does not time out if a time out value of 0 is used. If this method is not called, then any Session that is returned by the getSession method of this interface has a transaction timeout value set to 0 by default. An application can override the transaction timeout setting on a per Session basis by using the setTransactionTimeout method of the com.ibm.websphere.objectgrid.Session interface. You can also configure transaction timeout with the objectGrid.xml file in the distributed case.
int getTxTimeout();	getTxTimeout: Returns the transaction timeout value in seconds. This method returns the same value that is passed as the timeout parameter on the setTxTimeout method. If the setTxTimeout method was not called, then the method returns 0 to indicate that the transaction is allowed an unlimited amount of time to complete.
public int getObjectGridType();	Returns the type of ObjectGrid. The return value is equivalent to one of the constants declared on this interface: LOCAL, SERVER, or CLIENT.
public void registerEntities(Class[] entities);	Register one or more entities based on the class metadata. Entity registration is required prior to ObjectGrid initialization to bind an Entity with a BackingMap and any defined indices. This method may be called multiple times.
public void registerEntities(URL entityXML);	Registers one ore more entities from an entity XML file. Entity registration is required prior to ObjectGrid initialization to bind an Entity with a BackingMap and any defined indices. This method may be called multiple times.
void setQueryConfig(QueryConfig queryConfig);	Set the QueryConfig object for this ObjectGrid. A QueryConfig object provides query configurations for executing object queries over the maps in this ObjectGrid.
//Security	
void setSecurityEnabled()	setSecurityEnabled: Enables security. Security is disabled by default.
void setPermissionCheckPeriod(long period);	setPermissionCheckPeriod: This method takes a single parameter that indicates how often to check the permission that is used to allow a client access. If the parameter is 0, all methods ask the authorization mechanism, either JAAS authorization or custom authorization, to check if the current subject has permission. This strategy might cause performance issues depending on the authorization implementation. However, this type of authorization is available if it is required. Alternatively, if the parameter is less than 0, it indicates the number of milliseconds to cache a set of permissions before returning to the authorization mechanism to refresh them. This parameter provides much better performance, but if the backend permissions are changed during this time the ObjectGrid might allow or prevent access even though the backend security provider has been modified.
void setAuthorizationMechanism(int authMechanism);	setAuthorizationMechanism: Set the authorization mechanism. The default is SecurityConstants.JAAS_AUTHORIZATION.
setSubjectSource(SubjectSource ss);	setSubjectSource: Sets the SubjectSource plugin. This plug-in can be used to get a Subject object that represents the ObjectGrid client. This subject is used for ObjectGrid authorization. The SubjectSource.getSubject method is called by the ObjectGrid runtime when the ObjectGrid.getSession method is used to get a session and the security is enabled. This plug-in is useful for an already authenticated client: it can retrieve the authenticated Subject object and then pass to the ObjectGrid instance. Another authentication is not necessary.
setSubjectValidation(SubjectValidation sv);	setSubjectValidation: Sets the SubjectValidation plugin for this ObjectGrid instance. This plug-in can be used to validate that a javax.security.auth.Subject subject that is passed to the ObjectGrid is a valid subject that has not been tampered with. An implementation of this plug-in needs support from the Subject object creator, because only the creator knows if the Subject object has been tampered with. However, a subject creator might not know if the Subject has been tampered with. In this case, this plug-in should not be used.

Method	Description
void setObjectGridAuthorization (ObjectGridAuthorization ogAuthorization);	Sets the ObjectGridAuthorization for this ObjectGrid instance. Passing null to this method removes a previously set ObjectGridAuthorization object from an earlier invocation of this method and indicates that this <code>ObjectGrid</code> is not associated with a ObjectGridAuthorization object. This method should only be used when ObjectGrid security is enabled. If the ObjectGrid security is disabled, the provided ObjectGridAuthorization object will not be used. A ObjectGridAuthorization plug-in can be used to authorize access to the ObjectGrid and maps.

ObjectGrid interface: plug-ins

The ObjectGrid interface has several optional plug-in points for more extensible interactions.

```
void addEventListener(ObjectGridEventListener cb);
void setEventListeners(List cbList);
void removeEventListener(ObjectGridEventListener cb);
void setTransactionCallback(TransactionCallback callback);
int reserveSlot(String);
// Security related plug-ins
void setSubjectValidation(SubjectValidation subjectValidation);
void setSubjectSource(SubjectSource source);
```

- ObjectGridEventListener: An ObjectGridEventListener interface is used to receive notifications when significant events occur on the ObjectGrid. These events include ObjectGrid initialization, beginning of a transaction, ending a transaction, and destroying an ObjectGrid. To listen for these events, create a class that implements the ObjectGridEventListener interface and add it to the ObjectGrid. These listeners are associated with each Session. See Listeners and Session interface for more information.
- TransactionCallback: A TransactionCallback listener interface allows transactional events such as begin, commit and rollback signals to send to this interface. Typically, a TransactionCallback listener interface is used with a Loader. For more information, see TransactionCallback plug-in and Loaders. These events can then be used to coordinate transactions with an external resource or within multiple loaders.
- reserveSlot: Allows plug-ins on this ObjectGrid to reserve slots for use in object instances that have slots like TxID.
- SubjectValidation. If security is enabled, this plug-in can be used to validate a javax.security.auth.Subject class that is passed to the ObjectGrid.
- SubjectSource: If security is enabled, this plug-in can be used to get a Subject object that represents the ObjectGrid client. This subject is then used for ObjectGrid authorization.

For more information about plug-ins, see Java plug-ins overview.

BackingMap interface

Each ObjectGrid instance contains a collection of BackingMap objects. Use the defineMap method or the createMap method of the ObjectGrid interface to name and add each BackingMap to an ObjectGrid instance. These methods return a BackingMap instance that is then used to define the behavior of an individual Map. A BackingMap can be considered as an in-memory cache of committed data for an individual map.

Session interface

The Session interface is used to begin a transaction and to obtain the ObjectMap or JavaMap that is required for performing transactional interaction between an application and a BackingMap object. However, the transaction changes are not applied to the BackingMap object until the transaction is committed. A BackingMap can be considered as an in-memory cache of committed data for an individual map. For more information, see Using Sessions to access data in the grid.

The BackingMap interface provides methods for setting BackingMap attributes. Some of the set methods allow extensibility of a BackingMap through several custom designed plug-ins. See the following list of the set methods for setting attributes and providing custom designed plug-in support:

```
// For setting BackingMap attributes.
public void setReadOnly(boolean readOnlyEnabled);
public void setNullValuesSupported(boolean nullValuesSupported);
public void setLockStrategy( LockStrategy lockStrategy );
public void setCopyMode(CopyMode mode, Class valueInterface);
public void setCopyKey(boolean b);
public void setNumberOfBuckets(int numBuckets);
public void setNumberOfLockBuckets(int numBuckets);
public void setLockTimeout(int seconds);
public void setTimeToLive(int seconds);
public void setTtlEvictorType(TTLType type);
public void setEvictionTriggers(String evictionTriggers);

// For setting an optional custom plug-in provided by application.
public abstract void setObjectTransformer(ObjectTransformer t);
public abstract void setOptimisticCallback(OptimisticCallback checker);
public abstract void setLoader(Loader loader);
public abstract void setPreloadMode(boolean async);
public abstract void setEvictor(Evictor e);
public void setMapEventListeners( List /*MapEventListener*/ eventListenerList );
public void addMapEventListener(MapEventListener eventListener );
public void removeMapEventListener(MapEventListener eventListener );
public void addMapIndexPlugin(MapIndexPlugin index);
public void setMapIndexPlugins(List /\* MapIndexPlugin \*/ indexList );
public void createDynamicIndex(String name, boolean isRangeIndex,
```

```
String attributeName, DynamicIndexCallback cb);
public void createDynamicIndex(MapIndexPlugin index, DynamicIndexCallback cb);
public void removeDynamicIndex(String name);
```

A corresponding get method exists for each of the set methods listed.

BackingMap attributes

Each BackingMap has the following attributes that can be set to modify or control the BackingMap behavior:

backingMap attributes

copyKey
: Specifies if the a copy of the key is required when a map entry is created. Copying the key object allows the application to use the same key object for each ObjectMap operation. Set the value to `true` to copy the key object when a map entry is created. The default value is `false`. (Optional)

CopyMode
: 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. 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.

  For more information about these settings, see Tuning the copy mode.

evictionTriggers
: Sets the types of additional eviction triggers to use. All evictors for the backing map use this list of additional triggers. To avoid an IllegalStateException, this attribute must be called before the ObjectGrid.initialize() method. Also, note that the ObjectGrid.getSession() method implicitly calls the ObjectGrid.initialize() method if the method has yet to be called by the application. Entries in the list of triggers are separated by semicolons. Current eviction triggers include MEMORY_USAGE_THRESHOLD. For more information, see Plug-ins for evicting cache objects. (Optional)

lockStrategy
: 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`. The default value is `OPTIMISTIC`. (Optional)
  The optimistic locking strategy is typically used when a map does not have a loader plug-in, is mostly read and not frequently written to or updated, and the locking is not provided by the persistence manager using eXtreme Scale as a side cache or by the application. An exclusive lock is obtained on a map entry that is inserted, updated, or removed at commit time. The lock ensures that the version information cannot be changed by another transaction while the transaction being committed is performing an optimistic version check.

  The pessimistic locking strategy is typically used for a map that does not have a loader plug-in, and locking is not provided by a persistence manager using eXtreme Scale as a side cache, by a loader plug-in, or by the application. The pessimistic locking strategy is used when the optimistic locking strategy fails too often because update transactions frequently collide on the same map entry.

  The no locking strategy indicates that the internal LockManager is not needed. Concurrency control is provided outside of eXtreme Scale, either by the persistence manager using eXtreme Scale as a side cache or application, or by the loader plug-in that uses database locks to control concurrency.

lockTimeout
: Sets the lock timeout that is used by the lock manager for the BackingMap instance. Set the lockStrategy attribute to `OPTIMISTIC` or `PESSIMISTIC` to create a lock manager for the BackingMap instance. To prevent deadlocks from occurring, the lock manager has a default timeout value of 15 seconds. If the timeout limit is exceeded, a LockTimeoutException exception occurs. The default value of 15 seconds is sufficient for most applications, but on a heavily loaded system, a timeout might occur when no deadlock exists. Use the lockTimeout attribute to increase the value from the default to prevent false timeout exceptions from occurring. Set the lockStrategy attribute to `NONE` to specify the BackingMap instance use no lock manager. (Optional)

name
: Specifies the name that is assigned to the backingMap instance. If this attribute is missing, the XML validation fails. (Required)

nullValuesSupported
: Set the value to `true` to support null values in the ObjectMap. When null values are supported, a get operation that returns null might mean that the value is null or that the map does not contain the key that is passed to the method. The default value is `true`. (Optional)

numberOfBuckets
: The BackingMap instance uses a hash map for implementation. The numberOfBuckets attribute specifies the number of buckets for the BackingMap instance to use. If multiple entries exist in the BackingMap, more buckets lead to better performance because the risk of collisions is lower as the number

of buckets increases. More buckets also lead to more concurrency. Specify a value of `0` to disable the near cache on a client. When you set the value to `0` for a client, set the value in the client override ObjectGrid XML descriptor file only. (Optional)

numberOfLockBuckets
: The lock manager uses a hash map to track entries that are locked by one or more transactions. The numberOfLockBuckets attribute sets the number of lock buckets that are used by the lock manager for the BackingMap instance. Set the lockStrategy attribute to `OPTIMISTIC` or `PESSIMISTIC` to create a lock manager for the BackingMap instance. If many entries exist, more lock buckets lead to better performance because the risk of collisions is lower as the number of buckets grows. More lock buckets also lead to more concurrency. Set the lockStrategy attribute to `NONE` to specify the BackingMap instance use no lock manager. (Optional)

pluginCollectionRef
: Specifies a reference to a backingMapPluginCollection plug-in. The value of this attribute must match the ID attribute of a backingMapCollection plug-in. Validation fails if no matching ID exists. Set the attribute to reuse BackingMap plug-ins. (Optional)

preloadMode
: Sets the preload mode if a loader plug-in is set for this BackingMap instance. The default value is `false`. If the attribute is set to `true`, the Loader.preloadMap(Session, BackingMap) method is invoked asynchronously. Otherwise, running the method is blocked when loading data so that the cache is unavailable until preload completes. Preloading occurs during initialization. (Optional)

readOnly
: Sets a BackingMap instance as read/write when you specify the attribute as `false`. When you specify the attribute as `true`, the BackingMap instance is read-only. (Optional)

template
: Specifies if dynamic maps can be used. Set this value to `true` if the BackingMap map is a template map. Template maps can be used to dynamically create maps after the ObjectGrid is started. Calls to Session.getMap(String) result in dynamic maps being created if the name passed to the method matches the regular expression specified in the name attribute of the backingMap. The default value is `false`. (Optional)

timeToLive
: Specifies in seconds how long each map entry is present. The default value of `0` means that the map entry is present forever, or until the application explicitly removes or invalidates the entry. Otherwise, the TTL evictor evicts the map entry based on this value. (Optional)

ttlEvictorType
: Specifies how the expiration time of a BackingMap entry is computed. Set this attribute to one of these values: `CREATION_TIME`, `LAST_ACCESS_TIME`, `LAST_UPDATE_TIME`, or `NONE`. The `CREATION_TIME` value indicates that an entry expiration time is the sum of the creation time of the entry plus the timeToLive attribute value. The `LAST_ACCESS_TIME` value indicates that an entry expiration time is the sum of the last access time of the entry (whether the entry was updated or merely read), plus the timeToLive attribute value. The `LAST_UPDATE_TIME` value indicates that an entry expiration time is the sum of the last update time of the entry plus the timeToLive attribute value. The `NONE` value, which is the default, indicates that an entry has no expiration time and is present in the BackingMap instance until the application explicitly removes or invalidates the entry. (Optional)

valueInterfaceClassName
: Specifies a class that is required when you set the CopyMode attribute to `COPY_ON_WRITE`. This attribute is ignored for all other modes. The `COPY_ON_WRITE` value uses a proxy when ObjectMap.get method calls are made. The proxy ensures that a copy of the value is not made unless the application calls a set method on the class that is specified as the valueInterfaceClassName attribute. (Optional)

viewRef
: Specifies that the backingMap is a view map. (Optional)

writeBehind
: Specifies that the write-behind support is enabled with write-behind parameters (Optional). Write-behind parameters consist of a maximum update time and a maximum key update count. The format of the write-behind parameter is "`[T(time)][;][C(count)]`". The database is updated when one of the following events occurs:

  - The maximum update time, specified in seconds, has passed since the last update.
  - The number of available updates in the queue map has reached the maximum update count.

  For more information, see Write-behind caching.

  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.

The following example demonstrates how to define the someMap BackingMap in the someGrid ObjectGrid instance and set various attributes of the BackingMap by using the set methods of the BackingMap interface:

```
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.LockStrategy;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;

...

ObjectGrid og =
ObjectGridManagerFactory.getObjectGridManager().createObjectGrid("someGrid");
BackingMap bm = objectGrid.getMap("someMap");
bm.setReadOnly( true ); // override default of read/write
bm.setNullValuesSupported(false); // override default of allowing Null values
bm.setLockStrategy( LockStrategy.PESSIMISTIC ); // override default of OPTIMISTIC
bm.setLockTimeout( 60 ); // override default of 15 seconds.
bm.setNumberOfBuckets(251); // override default (prime numbers work best)
bm.setNumberOfLockBuckets(251); // override default (prime numbers work best)
```

BackingMap plug-ins

The BackingMap interface has several optional plug points for more extensible interactions with the ObjectGrid:

- **Evictor**: An evictor plug-in is a default mechanism that is provided for evicting cache entries and for creating custom evictors. The built in time-to-live evictor uses a time-based algorithm to decide when an entry in BackingMap must be evicted. Some applications might need to use a different algorithm for deciding when a cache entry needs to be evicted. The Evictor plug-in makes a custom designed Evictor available to the BackingMap to use. The Evictor plug-in is in addition to the built in time-to-live evictor. You can use the provided custom Evictor plug-in that implements well-known algorithms such as

"least recently used" or "least frequently used". Applications can either plug-in one of the provided Evictor plug-ins or it can provide its own Evictor plug-in. For more information, see Plug-ins for evicting cache objects.

- **ObjectTransformer**: An ObjectTransformer plug-in allows you to serialize, deserialize, and copy objects in the cache. 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. For more information, see ObjectTransformer plug-in.
- **OptimisticCallback**: An OptimisticCallback plug-in allows you to customize versioning and comparison operations of cache objects when you are using the optimistic lock strategy. The OptimisticCallback plug-in has been replaced by the ValueDataSerializer.Versionable interface, which you can implement when you use the DataSerializer plug-in with the COPY_TO_BYTES copy mode or when you use the @Version annotation with the EntityManager API. For more information, see Plug-ins for versioning and comparing cache objects.
- **MapEventListener**: A MapEventListener plug-in provides callback notifications and significant cache state changes that occur for a BackingMap. An application might want to know about BackingMap events such as a map entry eviction or a preload of a BackingMap completion. A BackingMap calls methods on the MapEventListener plug-in to notify an application of BackingMap events. An application can receive notification of various BackingMap events by using the setMapEventListener method to provide one or more custom designed MapEventListener plug-ins to the BackingMap. The application can modify the listed MapEventListener objects by using the addMapEventListener method or the removeMapEventListener method. For more information, see MapEventListener plug-in.
- **BackingMapLifecycleListener**: A BackingMapLifecycleListener plug-in provides BackingMap life cycle events for the BackingMap instance. The BackingMapLifecycleListener plug-in can be used as an optional mix-in interface for all other BackingMap plug-ins.
- **BackingMapPlugin**: A BackingMapPlugin is an optional mix-in interface that provides extended life cycle management events for all other BackingMap plug-ins.
- **Indexing**: Use the indexing feature, which is represented by the MapIndexplug-in plug-in, to build an index or several indexes on a BackingMap map to support non-key data access.
- **Loader**: A Loader plug-in on an ObjectGrid map acts as a memory cache for data that is typically kept in a persistent store on either the same system or some other system. (Server side only) For example, a Java database connectivity (JDBC) Loader can be used to move data in and out of a BackingMap and one or more relational tables of a relational database. A relational database does not need to be used as the persistent store for a BackingMap. The Loader can also be used to moved data between a BackingMap and a file, between a BackingMap and a Hibernate map, between a BackingMap and a Java 2 Platform, Enterprise Edition (JEE) entity bean, between a BackingMap and another application server, and so on. The application must provide a custom-designed Loader plug-in to move data between the BackingMap and the persistent store for every technology that is used. If a Loader is not provided, the BackingMap becomes a simple in-memory cache. For more information, see Plug-ins for communicating with databases.
- **MapSerializerPlugin**: A MapSerializerPlugin allows you to serialize and inflate Java objects and non-Java data in the cache. It is used with the DataSerializer mix-in interfaces, allowing robust and flexible options for high-performance applications.

ExceptionMapper interface

When a user plug-in implementation throws an exception, eXtreme Scale checks certain exceptions defined in the throws contract. However, sometimes an unchecked exception contains a contract exception or the exception does not observe the contract appropriately. Therefore a mechanism is necessary to map the exception to the contract exception if possible, such as ExceptionMapper.

Configuring the bean

Consider that a JPALoader must throw a LoaderNotAvailableException when the database server or network is not functional or the database runs out of resources. However, the JPA provider implementation normally just throws a generic PersistenceException with an SQLException, the SQL state or error code of which could indicate the database server problem or network problem. To further complicate the situation, different databases have different SQL state and error codes for such problems. So the exception-mapping mechanism has to be specific to the database.

The ExceptionMapper interface is used to solve the problem. It has a method Throwable map(Throwable original) to map the original exception to a consumable exception.

For example, to solve the stated problem, the implementation class could introspect the SQL state and error code of the java.sql.SQLException chained in the JPA exception. Then, it can throw a LoaderNotAvailableException if the SQL state or error code indicates the database server or network is not functional or the database runs out of resources.

Currently, the ExceptionMapper bean can only be configured in the JPATxCallback ObjectGrid beans. It is used to map all the exceptions received from the JPATxCallback and JPALoader or JPAEntityLoader beans.

To configure an ExceptionMapper, you have to use a Spring-style configuration for the ExceptionMapper bean inside of the JPATxCallback bean.

See Configuring JPA loaders for information about how to use Spring-style configuration for a JPALoader.

The following code is an example in which we map the JPA exceptions to a LoaderNotAvailableException if it indicates a database server problem or network problem. This ExceptionMapper implementation is for using a JPA provider with an MSSQL database. First, define a set of SQL error codes and SQL states that indicate the particular network problem or database server problem. In the map method, first check the SQL error code against a list of known error codes, then the SQL states, and finally the message. If one of them matches, throw a LoaderNotAvaliableException.

Code example

```
public class JPAMSSQLExceptionMapper implements ExceptionMapper {

    static Set<Integer> loaderNotAvailableSQLErrorSet = new HashSet<Integer>();

    static Set<String> loaderNotAvailableSQLStateSet = new HashSet<String>();

    static {
        addInitialMaps();
    }
```

```
    public C3P0MSSQLExceptionMapper() {
        System.out.println("C3P0MSSQLExceptionMapper is constructed");
    }

    /**
     * @internal
     * This method is used to add intial maps to the hash, this is used
     * internally, and it is not for public view
     */
    private static void addInitialMaps() {
        //  http://msdn.microsoft.com/en-us/library/cc645603.aspx
        loaderNotAvailableSQLErrorSet.add(new Integer(230));
        loaderNotAvailableSQLErrorSet.add(new Integer(6002));

        // http://white-box.us/2009/03/08/sql-92-sqlstate-codes/
        /*
         * 08001 SQL client unable to establish SQL connection
         * 08002 connection name in use
         * 08003 connection does not exist
         * 08004 SQL server rejected SQL connection
         * 08006 connection failure
         * 08007 transaction resolution unknown
         */
        loaderNotAvailableSQLStateSet.add("08000");
        loaderNotAvailableSQLStateSet.add("08001");
        loaderNotAvailableSQLStateSet.add("08002");
        loaderNotAvailableSQLStateSet.add("08003");
        loaderNotAvailableSQLStateSet.add("08004");
        loaderNotAvailableSQLStateSet.add("08006");
        loaderNotAvailableSQLStateSet.add("08007");

        // http://msdn.microsoft.com/en-us/library/ms714687.aspx
        loaderNotAvailableSQLStateSet.add("08S01");
        loaderNotAvailableSQLStateSet.add("HY000");
    }

    private static Pattern[] sqlServerMessagePatterns = new Pattern[] {
            Pattern.compile("The TCP/IP connection to the host .* has failed.*"), Pattern.compile(".*Connection
reset.*") };

    private static Pattern[] sqlExceptionMessagePatterns = new Pattern[] { Pattern
            .compile(".*Connections could not be acquired from the underlying database.*") };

    private static String connection_reset = "Connection reset";

    public Throwable map(Throwable originalEx) {

        Throwable cause = originalEx;

        while (cause != null) {
            // keep looping to check the next chained exception
            if (cause instanceof SQLException) {
                // Only check if the exception is an SQLException

                SQLException sqle = (SQLException) cause;

                // If the loader not available SQL state set contains this SQL state, then
                // we return a LoaderNotAvailableException with the original exception chained in it.
                if (loaderNotAvailableSQLStateSet.contains(sqle.getSQLState())) {
                    return new LoaderNotAvailableException(originalEx);
                }

                // If the loader not available SQL error code set contains this error code, then
                // we return a LoaderNotAvailableException with the original exception chained in it
                if (loaderNotAvailableSQLErrorSet.contains(new Integer(sqle.getErrorCode()))) {
                    return new LoaderNotAvailableException(originalEx);
                }

                String msg = sqle.getMessage();
                for (int i=0; i<sqlExceptionMessagePatterns.length; i++) {
                    if (sqlExceptionMessagePatterns[i].matcher(msg).matches()) {
                        return new LoaderNotAvailableException(originalEx);
                    }
                }

            } else if (cause.getClass().getName().equals("com.microsoft.sqlserver.jdbc.SQLServerException")) {
                String msg = cause.getMessage();
                for (int i=0; i<sqlServerMessagePatterns.length; i++) {
                    if (sqlServerMessagePatterns[i].matcher(msg).matches()) {
                        return new LoaderNotAvailableException(originalEx);
                    }
                }
                System.out.println("msg " + msg + " does not match");
            }

            // Get the next chained exception
            Throwable newCause = cause.getCause();

            // Safe-guard check to avoid indefinite loop if the exception chains itself
            if (newCause == cause) {
```

```
                // Always return the original exception if cannot map it.
                return originalEx;
            } else {
                cause = newCause;
            }
        }

        // Always retrun the original exception if cannot map it.
        return originalEx;
    }
```

xscmd utility reference

You can use the following list of commands as a reference when you are using the **xscmd** utility.

General command parameters

The general format of **xscmd** utility commands follows. The optional parameters are in square brackets [ ].
8.5

```
Usage: xscmd -c <cmdName> | -h <cmdName> | -lc [<cmdGroupName>] | -lcg
             [-cgc <className>] [-ca <support>] [-sp
             <profileName>] [-ks <filePath>] [-ts <filePath>] [-trf
             <filePath>] [-prot <protocol>] [-cxpv <provider>] [-trs
             <traceSpec>] [-al <alias>] [-pwd <password>] [-tsp <password>]
             [-cep <endpoints>] [-ksp <password>] [-arc <integer>] [-tt
             <type>] [-tst <type>] [-ssp <profileName>] [-kst <type>] [-cgp
             <property>] [-user <username>]

Options:
  -al,--alias <alias>                    Alias name in the keystore.

  -arc,--authRetryCount <integer>        The retry count for authentication if
                                         the credential is expired. If the
                                         value is set to 0, then
                                         authentication retries do not occur.

  -c,--command <cmdName>                 Specifies the name of the command to
                                         execute

  -ca,--credAuth <support>               Set the client credential
                                         authentication support [Never,
                                         Supported, Required].

  -cep,--catalogEndPoints <endpoints>    Specifies one or more catalog service
                                         endpoints in the format
                                         <host>[:<listenerPort>][,<host>[:<liste
                                         nerPort>]]. Default endpoint:
                                         localhost:2809

  -cgc,--credGenClass <className>        Specifies the name of the class that
                                         implements the CredentialGenerator
                                         interface. This class is used to get
                                         credentials for clients.

  -cgp,--credGenProps <property>         Specifies the properties for the
                                         CredentialGenerator implementation
                                         class. The properties are set to the
                                         object with the setProperties(String)
                                         method.

  -cxpv,--contextProvider <provider>     Context provider.  Examples: IBMJSSE2,
                                         IBMJSSE, IBMJSSEFIPS.

  -h,--help <cmdName>                    Invokes general command-line help

  -ks,--keyStore <filePath>              Absolute path to keystore. Example:
                                         /etc/test/security/server.public

  -ksp,--keyStorePassword <password>     Specifies the password to the keystore.

  -kst,--keyStoreType <type>             Keystore type. Example: JKS, JCEK,
                                         PKCS12.

  -lc,--listCommands <cmdGroupName>      List all commands for a command group

  -lcg,--listCommandGroups               List all command groups

  -prot,--protocol <protocol>            Protocol.  Examples: SSL, SSLv2,
                                         SSLv3, TLS, TLSv1.

  -pwd,--password <password>             eXtreme Scale password security
                                         credential

  -sp,--secProfile <profileName>         Specifies a profile name.
```

-ssp,--saveSecProfile <profileName>	Save security parameter values in security profile.
-trf,--traceFile <filePath>	Specifies the absolute path to the generated trace file for xscmd command output
-trs,--traceSpec <traceSpec>	Specifies the trace specification for xscmd command output
-ts,--trustStore <filePath>	Absolute path to truststore. Example: /etc/test/security/server.public
-tsp,--trustStorePassword <password>	Truststore password
-tst,--trustStoreType <type>	Truststore type. Examples: JKS, JCEK, PKCS12.
-tt,--transportType <type>	Transport layer security configuration type. Examples: TCP/IP, SSL-Supported, SSL-Required.
-user,--username <username>	eXtreme Scale user name security credential

Restriction: You can use FIPS data encryption with the TLSv1 protocol only.

All commands

The full list of **xscmd** utility commands follows.
Note: Column names may change (except for a column type and position), and new columns may be added to the end of a table if a command is enhanced as appropriate.
For further help and parameters for a specific command run **xscmd -h command_name**. If you plan on developing a customized script with these commands, then you should redirect the standard error path to a null device by running the following: `./xscmd.sh -c commandName -Options 2> /dev/null`

Examples

The following examples demonstrate how to run a command from the **xscmd** utility:

```
./xscmd.sh -lc * This command lists all available commands
```

```
./xscmd.sh -h showMapSizes * This command lists help for the showMapSizes command
```

Important: The output and usage of the following commands are subject to change in the future. If a command is marked with an asterisk (*) it means that only the output of the command is subject to change:

- listHosts
- showPlacement
- placementServiceStatus
- showDomainReplicationState
- showReplicationState
- *osgiAll
- *osgiCheck
- *osgiCurrent
- *osgiUpdate
- *showLinkedPrimaries
- *triggerPlacement

Related concepts:
Planning for network ports
Related tasks:
Administering with the xscmd utility
8.5+ Querying and invalidating data
8.5+ Retrieving eXtreme Scale environment information with the xscmd utility
Stopping servers gracefully with the xscmd utility
Related information:
Port number settings in WebSphere Application Server versions
8.5+ http://docs.oracle.com/javase/1.4.2/docs/

Regular expression syntax

8.5+ A regular expression is a structured string that is used to match other strings. You can use regular expressions for data invalidation.

Regular expression examples

Regular expression	Description
.	

	Finds any one character.
`.*`	Finds any one character zero or more times.
`A.*`	Finds all strings that start with "A".
`.*A`	Finds all strings that end with "A".
`A?`	Finds "A", zero or one time.
`A*`	Finds "A", zero or many times.
`A+`	Finds "A", one time or many times.
`A?B*C+`	Finds any one character, followed by zero or one A, followed by zero or many Bs, followed by one or many Cs
`[ABC].*`	Finds strings that start with the letters "A", or "B", or "C".
`.*ABC.*`	Finds strings with the string "ABC" somewhere in the string.
`(ABC|DEF).*`	Finds strings that start with the string "ABC" or "DEF".

See the java.util.regex.Pattern API documentation for the full regular expression syntax.

Related tasks:

8.5+ Querying and invalidating data

Related reference:

java.utils.regex.Pattern API documentation

Configuration files

WebSphere® eXtreme Scale is configured by a collection of XML and properties files.

- Server properties file
  The server properties file contains several properties that define different settings for your server, such as trace settings, logging, and security configuration. The server properties file is used by both catalog service and container servers in both stand-alone servers and servers that are hosted in WebSphere Application Server.
- Client properties file
  Create a properties file based on your requirements for WebSphere eXtreme Scale client processes.
- REST data service properties file
  To configure the REST data service, edit the properties file for REST and define the required entity schema for a data grid.
- ObjectGrid descriptor XML file
  To configure WebSphere eXtreme Scale, use an ObjectGrid descriptor XML file and the ObjectGrid API.
- Deployment policy descriptor XML file
  To configure a deployment policy, use a deployment policy descriptor XML file.
- Entity metadata descriptor XML file
  The entity metadata descriptor file is an XML file that is used to define an entity schema for WebSphere eXtreme Scale. Define all of the entity metadata in the XML file, or define the entity metadata as annotations on the entity Java™ class file. The primary use is for entities that cannot use Java annotations.
- Security descriptor XML file
  Use a security descriptor XML file to configure an eXtreme Scale deployment topology with security enabled. You can use the elements in this file to configure different aspects of security.
- Spring descriptor XML file
  Use a Spring descriptor XML file to configure and integrate eXtreme Scale with Spring.
- Liberty profile configuration files
  You can specify server properties and Liberty web feature properties for WebSphere eXtreme Scale applications that run in the Liberty profile.

Related tasks:

Troubleshooting XML configuration

Server properties file

The server properties file contains several properties that define different settings for your server, such as trace settings, logging, and security configuration. The server properties file is used by both catalog service and container servers in both stand-alone servers and servers that are hosted in WebSphere® Application Server.

- Server properties
  - General server properties
  - Container server properties
  - Catalog server properties
- Security server properties
  - General security properties
  - Transport layer security properties
    - Secure Sockets Layer (SSL) properties
  - Authentication properties

Sample server properties file

You can use the sampleServer.properties file that is in the *wxs_home*/properties directory to create your properties file.

Specifying a server properties file

Specifying a setting by using one of the items later in the list overrides the previous setting. For example, if you specify a system property value for the server properties file, the properties in that file override the values in the objectGridServer.properties file that is in the class path.

- For servers that run in WebSphere Application Server:
  - Use a well-named file in the class path, for example *was_root*/properties. If you put this well-named file in the current directory, the file is not found unless the current directory is in the class path. The name that is used follows:

    ```
    objectGridServer.properties
    ```

  - Specify a system property that specifies a file in the system current directory. Put the file in the *was_root*/properties directory. The file cannot be in the class path:

    ```
    -Dobjectgrid.server.props=file_name
    ```

- For servers that run in the Liberty profile:
  - In server.xml file, specify a server properties file by setting the serverProps attribute of the xsServer element; for example:

    ```
    <xsServer ... serverProps="${server.config.dir}/server.properties" ... />
    ```

    For eXtreme Scale servers that run in the Liberty profile, specify a server properties file to configure security. You can configure other properties in the server.xml file. For more information, see Liberty profile configuration files.

- For stand-alone servers:
  - Use a well-named file in the class path, for example *wxs_home*/properties. If you put this well-named file in the current directory, the file is not found unless the current directory is in the class path. The name that is used follows:

    ```
    objectGridServer.properties
    ```

  - Specify the server properties file as a parameter when you run the start server command. You can specify an absolute path or a path that is relative to the directory from which you run the start server command.

    ```
    -serverProps file_name
    ```

- For embedded, stand alone servers:
  Use the embedded server API. Use the ServerFactory.getServerProperties and ServerFactory.getCatalogServerProperties methods. The data in the object is populated with the data from the properties files.

Server properties

General properties

diskOverflowCapBytes
: Specifies the maximum amount of disk space that is used by this server for disk overflow, in bytes. The default value specifies that there is no limit on how much is stored on disk.
Default: `Long.MAX_VALUE`

diskStoragePath
: Specifies the absolute path to a directory location used for storing overflow content.

diskOverflowMinDiskSpaceBytes
: Specifies that entries are not moved to disk if there is less than this amount of space free in diskStoragePath, in bytes.
Default: `0`

diskOverflowEnabled
: Enables the native overflow disk feature. You must enable eXtreme Memory for this feature to work.
Default: `false`

enableMBeans
: Enables ObjectGrid container Managed Beans (MBean). This property applies to both the container server and the catalog service.
Default: `true`

exitJVMOnTeardown
: Specifies whether the JVM is also stopped when the eXtreme Scale server is stopped in an OSGi framework. By default, the JVM continues to run when each server in an OSGi framework is stopped in the **xscmd** utility with the **-c teardown** command. If you want to stop the JVM as well, set this property to `true`.
Default: `false`

haManagerPort
: Specifies the port that is used by the high availability (HA) manager for heartbeat communication between peer container servers. The haManagerPort port is only used for peer-to-peer communication between container servers that are in same domain. If the haManagerPort property is not defined, then an ephemeral port is used. In WebSphere Application Server, this setting is inherited by the high availability manager port configuration.
Default: `A dynamic port is chosen`.

JMXConnectorPort
: Defines the Secure Sockets Layer (SSL) port to which the Java™ Management Extensions (JMX) service binds. Only required if an SSL transport protocol is needed for JMX data.

JMXServicePort

Required only for WebSphere eXtreme Scale in a stand-alone environment. Specifies the port number on which the MBean server listens for communication with Java Management Extensions (JMX).
Default: `1099`

jvmStatsFileName
Specifies the file name of the CSV statistics file for the JVM.
Default: `jvmstats`

jvmStatsLoggingEnabled
When set to `true`, enables log data for the JVM to be written to a CSV file.
Default: `true`

jvmStatsWriteRate
Specifies the write rate of the CSV statistics files for the JVM in writes per second.
Default: `10`

listenerHost
Specifies the host name to which the Object Request Broker (ORB) transport protocol binds for communication. The value must be a fully qualified domain name or IP address. If your configuration involves multiple network cards, set the listener host and port to the IP address for which to bind. By setting the listener and host port, it allows the transport mechanism in the JVM know which IP address to use. If you do not specify which IP address to use, symptoms such as connection timeouts, unusual API failures, and clients that seem to hang can occur.

listenerPort (catalog server)
Specifies the port number to which the Object Request Broker transport protocol binds for communication.
Default: `2809`

Note: When a data grid server is run inside and the ORB transport protocol is being used, another port ORB_LISTENER_ADDRESS must also be opened. The BOOTSTRAP_ADDRESS port forwards requests to this port.

listenerPort (container server)
Specifies the port number to which the ORB transport protocol binds for communication.
Default: An ephemeral port is chosen.

Note: When a data grid server is run inside WebSphere Application Server and the ORB transport protocol is being used, another port ORB_LISTENER_ADDRESS must also be opened. The BOOTSTRAP_ADDRESS port forwards requests to this port.

listenerPort (client)
Specifies the port number to which the ORB transport protocol binds for communication. This setting configures the client to communicate with the catalog and container service. If a listener is not configured with the ORB transport protocol, an ephemeral port is chosen at startup. This port can vary each time the client application is started.
Default: An ephemeral port is chosen.

Note: When a data grid client is run inside WebSphere Application Server and the ORB transport protocol is being used, another port ORB_LISTENER_ADDRESS must also be opened. The BOOTSTRAP_ADDRESS port forwards requests to this port.

mapStatsFileName
Specifies the file name of the CSV statistics file for the map.
Default: `mapstats`

mapStatsLoggingEnabled
When set to `true`, enables log data for the maps on the server to be written to a CSV file.
Default: `true`

mapStatsWriteRate
Specifies the write rate of the CSV statistics files for the map in writes per second.
Default: `10`

maxJVMStatsFiles
Indicates the maximum number of CSV statistics files that are generated for the JVM.
Default: `5`

maxJVMStatsFileSize
Indicates the maximum file size, in megabytes, of the CSV statistics files for the JVM.
Default: `100`

maxMapStatsFileSize
Indicates the maximum file size, in megabytes, of the CSV statistics files for the map.
Default: `100`

maxOGStatsFiles
Indicates the maximum number of CSV statistics files that are generated for the ObjectGrid instance.
Default: `5`

maxOGStatsFileSize
Indicates the maximum file size, in megabytes, of the CSV statistics files for the ObjectGrid instance.
Default: `100`

maxMapStatsFiles
Indicates the maximum number of CSV statistics files that are generated for the map.
Default: `5`

maxThreads
Specifies the maximum number of threads that are used by the internal thread pool in the run time for built-in evictor and DataGrid operations.

Default: `50`

minThreads
: Specifies the minimum number of threads that are used by the internal thread pool in the run time for built-in evictor and DataGrid operations.
Default: `10`

ogStatsFileName
: Specifies the file name of the CSV statistics file for the ObjectGrid instance.
Default: `ogstats`

ogStatsLoggingEnabled
: When set to `true`, enables log data for the ObjectGrid instance on the server to be written to a CSV file.
Default: `false`

ogStatsWriteRate
: Specifies the write rate of the CSV statistics files for the ObjectGrid instance in writes per second.
Default: `10`

serverName
: Sets the server name that is used to identify the server. This property applies to both the container server and the catalog service.

systemStreamToFileEnabled
: Enables the container to write the SystemOut, SystemErr, and trace output to a file. If this property is set to `false`, output is not written to a file and is instead written to the console.
Default: `true`

traceFile
: Specifies a file name to write trace information. This property applies to both the container server and the catalog service.

  Example: ../logs/c4Trace.log

  Restriction: The traceFile property is not supported in the Liberty profile.

traceSpec
: Enables trace and the trace specification string for the container server. Trace is disabled by default. This property applies to both the container server and the catalog service. Examples:

  - `ObjectGrid=all=enabled`
  - `ObjectGrid*=all=enabled`

  Restriction: The traceSpec property is not supported in the Liberty profile.

workingDirectory
: Specifies the location to where the container server output is written. When this value is not specified, the output is written to a log directory within the current directory. This property applies to both the container server and the catalog service.
Default: No value is defined.

zoneName
: Set the name of the zone to which the server belongs. This property applies to both the container server and the catalog service.

Container server properties

catalogServiceEndPoints
: Specifies the end points to connect to the catalog service domain. This value must be in the form `host:port,host:port`. The host value is the listenerHost value and the port value is the listenerPort value of the catalog server. This property applies to a container server only.

enableXM
: When set to `true`, enables IBM® eXtremeMemory on the server and configures the server to use IBM eXtremeIO for synchronous and asynchronous replication. Cache entries for maps that are compatible with eXtremeMemory are stored in native memory instead of on the Java heap. All container servers in the data grid must use the same value for the enableXM property.
Default:`false`

maxXMSize
: Sets the maximum amount of memory, in megabytes, used by the server for eXtremeMemory storage.
Default: 25% of the total memory on the system.

memoryThresholdPercentage
: Sets the memory threshold for memory-based eviction. The percentage specifies the maximum heap to be used in the Java virtual machine (JVM) before eviction occurs. The default value is `-1`, which indicates that the memory threshold is not set. If the memoryThresholdPercentage property is set, the MemoryPoolMXBean value is set with the provided value. For more information, see MemoryPoolMXBean interface in the Java API specification. However, eviction occurs only if eviction is enabled on an evictor. To enable memory-based eviction, see Plug-ins for evicting cache objects. This property applies to a container server only.
Default:`-1`

statsSpec
: Specifies the statistics specification for the container server.
Examples:

  `all=disabled`: Disables all statistics.

  `all=enabled`: Enables all statistics.

  For more information about enabling statistics, see Enabling statistics.

Catalog service properties

8.5+ allowableShardOverrage

8.5+ Specifies the percentage of container servers that a zone must have compared to the other zones in a multi-zone deployment before all the replica shards are placed in that zone. If the percentage of container servers in the zone is below the specified value, only a relative subset of the replicas available are placed. After the percentage exceeds the specified value, all the replicas are placed. Primary shards are always placed. For example, the allowableShardOverrage value is set to 0.75 (75 percent). If zone1 has two container servers, and zone2 has three container servers, the percentage of the container servers between the zones is 2/3 (67 percent). Because this percentage is less than the allowableShardOverrage value of 75 percent, not all the replicas for the data grid are necessarily placed until the zones have an equal number of container servers.

catalogClusterEndPoints

For stand-alone configurations only. Specifies a list of catalog service domain end points for the catalog service. This property specifies the catalog service end points to start the catalog service domain. Use the following comma-separated format:

```
serverName:hostName:clientPort:peerPort,<serverName:hostName:clientPort:peerPort>
```

serverName

Specifies the name of the catalog server.

hostName

Specifies the host name for the computer where the server is launched.

clientPort

Specifies the port that is used for peer catalog service communication.

peerPort

This value is the same as the haManagerPort. Specifies the port that is used for peer catalog service communication.

This property applies to the catalog service only. If you start more catalog servers, they must include the same servers in the catalogClusterEndPoints property. The order of the list can be different, but the servers that are contained in the list must be the same for each catalog server. Do not put any spaces in the list.

domainName

For stand-alone configurations only. Specifies the domain name that is used to uniquely identify this catalog service domain to clients when routing to multiple domains. This property applies to the catalog service only.

enableQuorum

Enables quorum for the catalog service. Quorum is used to ensure that most of the catalog service domain is available before partitions are moved to the available container servers. To enable quorum, set the value to `true` or `enabled`. The default value is `disabled`. This property applies to the catalog service only. For more information, see Catalog server quorums.

<*foreignDomain*>.endpoints

Specifies the connection information for the catalog servers of the foreign domains, such as domain B:
For example:

```
B.endPoints=hostB1:2809, hostB2:2809
```

If a foreign domain has multiple catalog servers, specify all of them.

foreignDomains

Specifies the names of catalog service domains to which you want to link in a multi-master replication topology. You can specify multiple catalog service domains with a comma-separated list. This property applies to the catalog service only.

```
foreignDomains=domain2,domain3,domain4
```

Restriction: The foreignDomains property is not supported in the Liberty profile.

heartBeatFrequencyLevel

Specifies how often a server failover is detected. An aggressive heartbeat interval can be useful when the processes and network are stable. If the network or processes are not optimally configured, heartbeats might be missed, which can result in a false failure detection. The heartbeat frequency level is a trade-off between use of resources and failure discovery time. The more frequent a heartbeat occurs, then more resources are used. However, failures are discovered more quickly. This property applies to the catalog service only.

Table 1. Valid heartbeat values

Value	Action	Description
-1	Aggressive	Specifies an aggressive heartbeat level. With this value, failures are detected more quickly, but more processor and network resources are used. This level is more sensitive to missing heartbeats when the server is busy. Failovers are typically detected within 5 seconds.
0	Typical (default)	Specifies a heartbeat level at a typical rate. With this value, failover detection occurs at a reasonable rate without overusing resources. Failovers are typically detected within 30 seconds.
1	Relaxed	Specifies a relaxed heartbeat level. With this value, a decreased heartbeat frequency increases the time to detect failures, but also decreases processor and network use. Failovers are typically detected within 180 seconds.

isCatalog

For stand-alone configurations only. When set to `true`, the server process automatically starts a catalog service.
Default: `false`

placementDeferralInterval

Specifies the interval in milliseconds for deferring the balancing and placement of shards on the container servers. Placement does not start until after the time specified in the property has passed. Increasing the deferral interval lowers processor utilization, but the placement of work items is completed over time. A decrease in the deferral interval increases short-term processor usage, but the placement of work items is more immediate and expedited.
If multiple container servers are starting in succession, the deferral interval timer is reset if a new container server starts within the given interval. For example, if a second container server starts 10 seconds after the first container server, placement does not start until 15 seconds after the second container server started. However, if a third container server starts 20 seconds after the second container server, placement has already begun on the first two container servers.

When container servers become unavailable, placement is triggered as soon as the catalog server learns of the event so that recovery can occur as quickly as possible.

Default: 15000 ms (15 seconds)

Security server properties

The server properties file is also used to configure eXtreme Scale server security. You use a single server property file to specify both the basic properties and the security properties.

General security properties

credentialAuthentication

Indicates whether this server supports credential authentication. Choose one of the following values:

- `Never`: The server does not support credential authentication.
- `Supported`: The server supports the credential authentication if the client also supports credential authentication.
- `Required`: The client requires credential authentication.

See Authenticating application clients for details about credential authentication.

securityEnabled

Enables the container server security when set to `true`. The default value is `false`. When you want authentication to the data grid, this property must match the securityEnabled property that is specified in the objectGridSecurity.xml file, which is provided to the catalog server. When you are running with transport security only (the transportType property is set to either SSL-Supported or SSL-Required), it is not necessary to set the securityEnabled property to `true` in the objectGridSecurity.xml file.

Transport layer security settings

transportType

Specifies the server transport type. Use one of the following values:

- `TCP/IP`: Indicates that the server supports TCP/IP connections only.
- `SSL-Supported`: Indicates that the server supports both TCP/IP and Secure Sockets Layer (SSL) connections. (Default)
- `SSL-Required`: Indicates that the server requires SSL connections.

SSL configuration properties

alias

Specifies the alias name in the keystore. This property is used if the keystore has multiple key pair certificates and you want to select one of the certificates.
Default: No value is defined.

clientAuthentication

By default, the server does not authenticate the client and this property is set to false. When set to true, the server expects to receive and authenticate the client credential during the SSL handshake.
Valid values: `true` or `false`

contextProvider

Specifies the name of the context provider for the SSL or TLS implementation. If you indicate a value that is not valid, a security exception result that indicates that the context provider type is incorrect.
Valid values: `IBMJSSE2`, `IBMJSSE`, `IBMJSSEFIPS`, and so on.
Note: Use the `IBMJSSE2` value when you have a Java runtime environment that is provided by IBM. The values, `IBMJSSE` and `IBMJSSEFIPS`, are deprecated. If you use SSL security with the ORB transport and a JRE that is not provided by IBM, then see the documentation from your JRE vendor for an appropriate context provider setting.

customSecureTokenManagerProps

Specifies the custom SecureTokenManager implementation class properties. This property is used only if the secureTokenManagerType value is `custom`. The value is set to the SecureTokenManager Object with the setProperties(String) method.

customTokenManagerClass

Specifies the name of your SecureTokenManager implementation class, if you specified the SecureTokenManagerType property value as `custom`. The implementation class must have a default constructor to be instantiated.

keyStore

Specifies a fully qualified path to the keystore file.
Example: etc/test/security/client.private

Important: The directory path does not support Windows backslashes. If you have used backslashes, you must escape any backslash (\ ) characters in the path. For example, if you want to use the path C:\opt\ibm, enter C:\\opt\\ibm in the properties file. Windows directories with spaces are not supported.

keyStorePassword

Specifies the string password to the keystore. You can encode this value or use the actual value.

keyStoreType

Indicates the type of keystore. If you indicate a value that is not valid, a runtime security exception results.
Valid values: `JKS`, `JCEK`, `PKCS12`, and so on.

Important: The directory path does not support Windows backslashes. If you have used backslashes, you must escape any backslash (\ ) characters in the path. For example, if you want to use the path C:\opt\ibm, enter C:\\opt\\ibm in the properties file. Windows directories with spaces are not supported.

protocol

Indicates the type of security protocol to use for the client. Set this protocol value that is based on the Java Secure Socket Extension (JSSE) provider you use. If you indicate a value that is not valid, a security exception result that indicates that the protocol value is incorrect.
Valid values: `SSL`, `SSLv2`,`SSLv3`, `TLS`, `TLSv1`.

SecureTokenManager

The SecureTokenManager setting is used for protecting the secret string for server mutual authentications and for protecting the single sign-on token.

secureTokenManagerType
Specifies the type of SecureTokenManager setting. You must use the same secureTokenManagerType setting in all of the servers in the catalog service domain, and all servers in linked catalog service domains. You can use one of the following settings:

- `none`: Indicates that no secure token manager is used. A secure token manager is required to protect the authenticationSecret attribute value when it is transmitted over the network. This setting also disables the use of a single sign-on token.
- `default`: Indicates that a token manager that is supplied with the WebSphere eXtreme Scale product is used. You must provide a SecureToken keystore configuration.
- `custom`: Indicates that you have your own token manager that you specified with the SecureTokenManager implementation class.

trustStore
Specifies a fully qualified path to the truststore file.
Example: etc/test/security/server.public

Important: The directory path does not support Windows backslashes. If you have used backslashes, you must escape any backslash (\ ) characters in the path. For example, if you want to use the path C:\opt\ibm, enter C:\\opt\\ibm in the properties file. Windows directories with spaces are not supported.

trustStorePassword
Specifies a string password to the truststore. You can encode this value or use the actual value.

trustStoreType
Indicates the type of truststore. If you indicate a value that is not valid, a runtime security exception results.
Valid values: `JKS`, `JCEK`, `PKCS12`, and so on.

Secure token keystore configuration
The secure token keystore configuration is only needed when secureTokenManager=default.

secureTokenKeyStore
Specifies the file path name for the keystore that stores the public-private key pair and the secret key.

secureTokenKeystorePassword
Specifies the password for the keystore that stores the public-private key pair and the secret key.

secureTokenKeyStoreType
Specifies the keystore type, for example, JCEKS. You can set this value that is based on the Java Secure Socket Extension (JSSE) provider that you use. However, the keystore must support secret keys.

secureTokenKeyPairAlias
Specifies the alias of the public-private key pair that is used for signing and verifying.

secureTokenKeyPairPassword
Specifies the password to protect the key pair alias that is used for signing and verifying.

secureTokenSecretKeyAlias
Specifies the secret key alias that is used for ciphering.

secureTokenSecretKeyPassword
Specifies the password to protect the secret key.

secureTokenCipherAlgorithm
Specifies the algorithm that is used for providing a cipher. You can set this value that is based on the Java Secure Socket Extension (JSSE) provider that you use.

secureTokenSignAlgorithm
Specifies the algorithm that is used for signing the object. You can set this value that is based on the JSSE provider that you use.

Authentication string

authenticationSecret
Specifies the secret string to challenge the server. When a server starts, it must present this string to the president server or catalog server. If the secret string matches what is in the president server, this server is allowed to join in. All of the servers in a catalog service domain, and the servers in any linked catalog service domains must use the same value this setting. The authenticationSecret value must be a long, hard to guess string. Do not use the authenticationSecret value that is in the sampleServer.properties in production deployments.

Related concepts:
OSGi framework overview
Statistics modules
Related tasks:
Installing the Eclipse Equinox OSGi framework with Eclipse Gemini for clients and servers
Monitoring with CSV files
Enabling statistics
Monitoring with the statistics API
Administering with the xscmd utility
Related reference:
Client properties file
REST data service properties file
ObjectGrid descriptor XML file
Deployment policy descriptor XML file
Entity metadata descriptor XML file
Security descriptor XML file
Spring descriptor XML file
Liberty profile configuration files
Sample properties files
Related information:
API documentation
Lesson 2.2: Configure catalog server security
Module 2: Configure WebSphere eXtreme Scale to use WebSphere Application Server Authentication plug-ins
Introduction: Starting and configuring the eXtreme Scale server and container to run plug-ins in the OSGi framework
StatsSpec class

Client properties file

Create a properties file based on your requirements for WebSphere® eXtreme Scale client processes.

Sample client properties file

You can use the sampleClient.properties file that is in the *wxs_home*/properties directory to create your properties file.

Specifying a client properties file

You can specify the client properties file in one of the following ways. Specifying a setting by using one of the items later in the list overrides the previous setting. For example, if you specify a system property value for the client properties file, the properties in that file override the values in the objectGridClient.properties file that is in the class path.

1. As a well-named file anywhere in the WebSphere Application Server class path. Putting this file in the system current directory is not supported:

   `objectGridClient.properties`

2. As a system property in either a stand-alone or WebSphere Application Server configuration. This value can specify a file in the system current directory, but not a file in the class path:

   `-Dobjectgrid.client.props=`*`file_name`*

   Note: In a WebSphere Application Server configuration, the client properties file must be in the classpath if you want to specify a particular client properties file to use with the system property; for example, *was_root*/properties or *profile_root*/properties, depending on whether you want the properties file to apply to a specific profile, or the entire installation.
3. As a programmatic override using the ClientClusterContext.getClientProperties method. The data in the object is populated with the data from the properties files. You cannot configure security properties with this method.

Client properties

Client properties

listenerHost
: Specifies the host name to which the Object Request Broker (ORB) transport protocol binds for communication. If your configuration involves multiple network cards, set the listener host and port to let the transport mechanism in the JVM know the IP address for which to bind. If you do not specify which IP address to use, symptoms such as connection timeouts, unusual API failures, and clients that seem to hang can occur.

  Default: `localhost`

  Valid values: port number

listenerPort
: Specifies the port number to which the Object Request Broker transport protocol binds for communication.
  Note: When a data grid server is run inside and the ORB transport protocol is being used, another port ORB_LISTENER_ADDRESS must also be opened. The BOOTSTRAP_ADDRESS port forwards requests to this port.
  Default: `2809`

  Valid values: fully qualified domain name or IP address

preferLocalProcess
: This property is not currently used. It is reserved for future use.

preferLocalHost
: This property is not currently used. It is reserved for future use.

preferZones
: Specifies a list of preferred routing zones. Each specified zone is separated by a comma in the form: `preferZones=ZoneA,ZoneB,ZoneC`
  Default: no value

  Valid value: comma-separated list of preferred zones

requestRetryTimeout
: Specifies how long to continue processing a request (in milliseconds) after an exception occurs.
  Default: -1

  Valid values:

  - A value of `0` indicates that the request should fail fast and skip over the internal retry logic.
  - A value of `-1` indicates that the request retry timeout is not set, meaning that the request duration is governed by the transaction timeout. (Default). The following levels of checking the request retry timeout are used to determine the default behavior:
    - Session instance requestRetryTimeout value
    - Client properties file requestRetryTimeout value
    - If neither of the previous values are set, then the lowest value between the transaction timeout value and 30 seconds is selected. For example, if the transaction timeout value has the default value of 10 minutes, then the request times out at 30 seconds. Alternatively, if you set the transaction timeout value to 20 seconds, then the request times out after 20 seconds.
  - A value over `0` indicates the request entry timeout value in milliseconds. Exceptions that are not successfully created are returned. Even when exceptions, such as DuplicateException, are tried again, they are also returned when they do not succeed. The transaction timeout is still used as

the maximum time to wait.

Security client properties

General security properties

securityEnabled

Enables WebSphere eXtreme Scale client security. This setting must match with the securityEnabled setting in theWebSphere eXtreme Scale server properties file. If the settings do not match, the client connection to the data grid fails.
Default: `false`

Valid values: `true`, `false`

Credential authentication configuration properties

credentialAuthentication

Specifies the client credential authentication support.
Use one of the following valid values:

Default: `Supported`

Valid values:

- `Never`: The client does not support credential authentication.
- `Supported`: The client supports credential authentication if the server also supports credential authentication. (Default)
- `Required`: The client requires credential authentication.

authenticationRetryCount

Specifies the number of times that authentication is tried if the credential is expired. If the value is set to `0`, attempts to authenticate are not tried again.
Default: `3`

Valid values: Integer value greater than or equal to `0`

credentialGeneratorClass

Specifies the name of the class that implements the com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator interface. To specify this property, the credentialAuthentication property must be set to `Supported` or `Required`. This class is used to get credentials for clients.
Several built-in classes support this interface; for example:

- com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredentialGenerator. A user ID and password separated by a space are required for this class.
- com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredentialGenerator

You can also create a custom class, as described in the eXtreme Scale API reference

Default: no value

Valid values: class name

credentialGeneratorProps

Specifies the properties for the CredentialGenerator implementation class. The properties are set to the object with the setProperties(String) method. Specify this property when the credentialAuthentication property is set to `Supported` or `Required`, and the credentialGeneratorClass requires properties.
Default: `SSL-Supported`

Valid values: `SSL-Supported`, `TCP/IP`, or `SSL-Required`

Transport layer security configuration properties

transportType

Specifies the client transport type. The possible values are:
Default: `SSL-Supported`

Valid values:

- `TCP/IP`: Indicates that the client only supports TCP/IP connections.
- `SSL-Supported`: Indicates that the client supports both TCP/IP and Secure Sockets Layer (SSL) connections. (Default)
- `SSL-Required`: Indicates that the client requires SSL connections.

SSL configuration properties

alias

Specifies the alias name in the keystore. This property is used if the keystore has multiple key pair certificates and you want to select one of the certificates. It is only required if client certificate authentication is configured on the server.
Default: no value

Valid value: alias name in keystore

contextProvider

Specifies the name of the context provider for the trust service. If you indicate a value that is not valid, a security exception results that indicates that the context provider type is incorrect.
Default: no value

Valid values: `IBMJSSE2`. If you are using a JRE from another vendor, see your vendor documentation for valid JSSE providers. A value must be specified if SSL is used.

keyStore
Specifies a fully qualified path to the keystore file. A value must be specified if SSL is used. The keyStore is not used unless client certificate authentication is configured on the server. If client certificate authentication is not used, you can specify the trustStore value for keyStore.
Default: no value

Valid values: fully qualified path to keystore
Important: The keyStore directory path does not support Windows backslashes. If you have used backslashes, you must escape any backslash (\ ) characters in the path. For example, if you want to use the path C:\opt\ibm, enter C:\\opt\\ibm in the properties file. Windows directories with spaces are not supported.
Example: etc/test/security/client.private

keyStorePassword
Specifies the string password to the keystore. A value must be specified if SSL is used. You can encode this value or use the actual value.

keyStoreType
Indicates the type of keystore. A value must be specified if SSL is used. If you indicate a value that is not valid, a runtime security exception occurs.
Default: no value

Valid values: `JKS`, `JCEK`, `PKCS12`, and so on.

protocol
Indicates the type of security protocol to use for the client. Set this protocol value based on which security provider you use. If you indicate a value that is not valid, a security exception results that indicates that the protocol value is incorrect. A value for protocol must be specified if SSL or TLS is used. If you are using a JRE from another vendor, see your vendor documentation for valid protocols.
Default:`SSL_TLS`. When you specify this default value, the server accepts either the SSL or TLS protocol from the client.

Valid values: When the IBM Runtime Environment is used, the following values are valid: `SSL`, `SSLv2`, `SSLv3`, `TLS`, `TLSv1`, , and `SSL_TLS`.

trustStoreType
Indicates the type of truststore. A value must be specified if SSL is used. If you indicate a value that is not valid, a runtime security exception results.
Valid values: `JKS`, `JCEKS`, `PKCS12`, and so on.

trustStore
Specifies a fully qualified path to the truststore file. A value must be specified if SSL is used.
Example: etc/test/security/server.public

trustStorePassword
Specifies a string password to the truststore. You can encode this value or use the actual value. A value must be specified if SSL is used.

Related concepts:
Data grid authentication
Data grid security
Related tasks:
Configuring eXtreme Scale connection factories
Configuring catalog servers and catalog service domains
Authenticating and authorizing clients
Authenticating application clients
Authorizing application clients
Configuring Secure Sockets Layer (SSL) parameters for clients or servers
Troubleshooting XML configuration
Related reference:
Server properties file
REST data service properties file
ObjectGrid descriptor XML file
Deployment policy descriptor XML file
Entity metadata descriptor XML file
Security descriptor XML file
Spring descriptor XML file
Liberty profile configuration files
Sample properties files
Class ClientSecurityConfigurationFactory
Related information:
Catalog service domain settings
API documentation
Lesson 2.1: Configure client server security
Module 2: Configure WebSphere eXtreme Scale to use WebSphere Application Server Authentication plug-ins
Credential API documentation

REST data service properties file

To configure the REST data service, edit the properties file for REST and define the required entity schema for a data grid.

The REST data service properties file is the main configuration file for the eXtreme Scale REST data service. This file is a Java™ property file with key-value pairs. By default, the REST data service runtime environment looks for a well-named wxsRestService.properties file in the classpath. The file can also be explicitly defined by using the system property: wxs.restservice.props.

```
-Dwxs.restservice.props=/usr/configs/dataservice.properties
```

When the REST data service is loaded, the property file used is displayed in the log files:

```
CWOBJ4004I: The eXtreme Scale REST data service properties files were loaded: [/usr/configs/RestService.properties]
```

The REST data service properties file supports the following properties:

Table 1. Properties for the REST data service

Property	Description

catalogServiceEndPoints	The required comma-delimited list of hosts and ports of a catalog service domain in the format: `<host:port>`. This is optional if using WebSphere® Application Server integrated with WebSphere eXtreme Scale to host the REST data service. See Starting a stand-alone catalog service for more information.
	`catalogServiceEndPoints=`
	`server1:2809,server2:2809`

objectGridNames	The required names of the data grids to expose to the REST service. At least one ObjectGrid name is required. Separate multiple ObjectGrid names using a comma: `ECommerceGrid,InventoryGrid`
objectGridClientXML	The optional name of the ObjectGrid client override XML file. The name specified here will be loaded from the classpath. The default is: /META-INF/objectGridClient.xml.
ogClientPropertyFile	The optional name of the ObjectGrid client property file. This file contains security properties that are required for enabling ObjectGrid client security. If the "securityEnabled" attribute is set in the property file, security will be enabled on the ObjectGrid client used by the REST service. The "credentialGeneratorProps" must also be set in the property file to a value in the format of "user:pass" or a value of {xor_encoded user:pass}
loginType	The type of authentication used by the REST Service when ObjectGrid client security is enabled. If ObjectGrid client security is not enabled, this property is ignored. If ObjectGrid client security is enabled and loginType is set to `basic`, the REST service will: • Use the credentials specified in the 'credentialGeneratorProps' property of the ObjectGrid client property file for ObjectGrid operations at service initialization. • Use HTTP BASIC authentication for per request ObjectGrid session operations If ObjectGrid client security is enabled and loginType is set to `none` the REST service will: • Use the credentials specified in the 'credentialGeneratorProps' property of the ObjectGrid client property file for ObjectGrid operations at service initialization. • Use the credentials specified in the 'credentialGeneratorProps' property of the ObjectGrid client property file for per request ObjectGrid session operations.
traceFile	The optional name of the file to redirect the trace output to. The default is logs/trace.log.
traceSpec	The optional trace specification that the eXtreme Scale runtime server should initially use. The default is `*=all=disabled`. To trace the entire REST data service, use: `ObjectGridRest*=all=enabled`
verboseOutput	If set to `true`, REST data service clients receive additional diagnostic information when failures occur. The default is `false`. This optional value should be set to false for production environments as sensitive information may be revealed.
maxResultsPerCollection	The optional maximum number of results that will be returned in a query. The default value is `unlimited`. Any positive integer is a valid value.
wxsRestAccessRightsFile	The optional name of the eXtreme Scale REST service access rights property file which specifies the access rights for the service operations and the ObjectGrid entities. If this property is specified, the REST service will try to load the file from the path specified, else it will try to load the file from its classpath.

WebSphere eXtreme Scale configuration

The eXtreme Scale REST data service interacts with eXtreme Scale using the EntityManager API. An entity schema is defined for an eXtreme Scale data grid and the metadata for the entities is automatically consumed by the REST data service. See Defining an entity schema for details about configuring an entity schema.

For example, you can define an entity representing a Person in an eXtreme Scale data grid as follows:

```
@Entity
public class Person {
                @Id String taxId;
                String firstName;
                String lastName;
}
```

Tip: The annotations used here are in the com.ibm.websphere.projector.annotations package.
The REST service automatically creates an ADO.NET Entity Data Model for Data Services (EDMX) document, which is available using the $metadata URI:

```
http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/$metadata
```

After the data grid is configured and running, configure and package an eXtreme Scale client. For details on configuring the eXtreme Scale REST data service client package, see the packaging and deployment information in Installing the REST data service.

Entity model

WebSphere eXtreme Scale entities are modeled using the entity annotations or an entity metadata descriptor file. For details on how to configure an entity schema, see Defining an entity schema. The eXtreme Scale REST service uses the entity metadata to automatically create an EDMX model for the data service.

This version of the WebSphere eXtreme Scale REST data service has the following schema restrictions:

- When defining entities in a partitioned data grid, all entities must have a direct or indirect single valued association to the root entity (a key association). The WCF data service client runtime environment must be able to access every entity directly through its canonical address. Therefore, the key of the root entity that is used for partition routing (the schema root) must be part of the key in the child entity.
  For example:

```
@Entity(schemaRoot=true)
public class Person {
    @Id String taxId;
    String firstName;
    String lastName;
    @OneToMany(mappedBy="person")
    List<Address> addresses;

}

@Entity
public class Address {
    @Id int addrId;
    @Id @ManyToOne Person person;
    String street;
}
```

- Bi-directional and uni-directional associations are supported. However, uni-directional associations may not always work from a Microsoft WCF Data Services client since they can only be navigated in one direction and the Microsoft specification requires all associations to be bi-directional.
- Referential constraints are not supported. The WebSphere eXtreme Scale runtime environment does not validate keys between entities. Associations between entities must be managed by the client.
- Complex types are not supported. The EntityManager API does not support embeddable attributes. All attributes are expected to be simple type attributes (see the simple attribute types listed below). Non-simple type attributes are treated as a binary object from the perspective of the client.
- Entity inheritance is not supported. The EntityManager API does not support inheritance.
- Media Resources and Media Links are not supported. The HasStream attribute of the EntityType in the Conceptual Schema Definition Language Document for Data Services is never used.

Mapping between EDM data types and Java data types

The OData protocol defines the following list of Entity Data Model (EDM) types in its abstract type system. The following topics describe how the eXtreme Scale REST adapter chooses the EDM type based on the basic type defined in the entity. For details on EDM types, see: MSDN Library: Abstract Type System.

The following EDM types are available in WCF Data Services:

- Edm.Binary
- Edm.Boolean
- Edm.Byte
- Edm.DateTime
- Edm.Time
- Edm.Decimal
- Edm.Double
- Edm.Single
- Edm.Float
- Edm.Guid *
- Edm.Int16
- Edm.Int32
- Edm.Int64
- Edm.SByte
- Edm.String

The EDM type: Edm.Guid is not supported by the eXtreme Scale REST data service.

Mapping Java types to EDM types

The eXtreme Scale REST data service automatically converts basic entity types into EDM types. The type mapping can be seen by displaying the Entity Data Model Extensions (EDMX) metadata document using the $metadata URI. The EDM type is used by clients to read and write data to the REST data service.

Table 2. Java types mapped to EDM types. The table shows the mapping from the Java type defined for an entity to the EDM data type. When retrieving data using a query, the data will be represented with these types:

Java Type	EDM Type
boolean java.lang.Boolean	Edm.Boolean
byte java.lang.Byte	Edm.SByte
short java.lang.Short	Edm.Int16

Java Type	EDM Type
int java.lang.Integer	Edm.Int32
long java.lang.Long	Edm.Int64
float java.lang.Float	Edm.Single
double java.lang.Double	Edm.Double
java.math.BigDecimal	Edm.Decimal
java.math.BigInteger	java.math.BigInteger
java.lang.String	Edm.String
char	char
java.lang.Character	java.lang.Character
Char[]	Char[]
java.lang.Character[]	java.lang.Character[]
java.util.Calendar	Edm.DateTime
java.util.Date	java.util.Date
java.sql.Date	java.sql.Date
java.sql.Timestamp	java.sql.Timestamp
java.sql.Time	java.sql.Time
Other types	Edm.Binary

Mapping from EDM types to Java types

For Update requests and Insert requests, the payload specifies the data to be updated or inserted into the eXtreme Scale REST data service. The service can automatically convert compatible data types to the data types defined in the EDMX document. The REST data service converts the XML encoded string representations of the value into the correct type using the following two-step process:

1. A type check is performed to make sure the EDM type is compatible with the Java type. An EDM type is compatible with a Java type if the data supported by the EDM type is a subset of the data supported by the Java type. For example, Edm.int32 type is compatible with a Java long type, but Edm.int32 type is not compatible with a Java short type.
2. A target Java type object will be created which represents the string value in the payload.

Table 3. Compatible EDM type to Java type

EDM Type	Java Type
Edm.Boolean	boolean java.lang.Boolean
Edm.SByte	byte java.lang.Byte short java.lang.Short int java.lang.Integer long java.lang.Long float java.lang.Float double java.lang.Double java.math.BigDecimal java.math.BigInteger char java.lang.Character

EDM Type	Java Type
Edm.Byte, Edm.Int16	short java.lang.Short int java.lang.Integer long java.lang.Long float java.lang.Float double java.lang.Double java.math.BigDecimal java.math.BigInteger char java.lang.Character
Edm.Int32	int java.lang.Integer long java.lang.Long float java.lang.Float double java.lang.Double java.math.BigDecimal java.math.BigInteger
Edm.Int64	long java.lang.Long double java.lang.Double java.math.BigDecimal java.math.BigInteger
Edm.Double	double java.lang.Double java.math.BigDecimal
Edm.Decimal	double java.lang.Double java.math.BigDecimal java.math.BigInteger
Edm.Single	float java.lang.Float double java.lang.Double java.math.BigDecimal

EDM Type	Java Type
Edm.String	java.lang.String char java.lang.Character Char[] java.lang.Character[] java.math.BigDecimal java.math.BigInteger
Edm.DateTime	java.util.Calendar java.util.Date java.sql.Date java.sql.Time java.sql.Timestamp
Edm.Time	java.sql.Time java.sql.Timestamp

Mapping temporal types

Java includes five temporal types for storing date, time or both: java.util.Date, java.sql.Date, java.sql.Time, java.sql.Timestamp and java.util.Calendar. All of these types are expressed in the entity data model as Edm.DateTime. The eXtreme Scale REST data service automatically converts and normalizes the data depending on the Java type. This topic describes several issues that developers must be aware of when using any temporal type.

Time zone differences

In WCF Data Services, the descriptions of time values in the Edm.DateTime type are always expressed using the Coordinated Universal Time (UTC) standard, which is the internationally recognized name for Greenwich Mean Time (GMT). Coordinated Universal Time is the time as measured at zero degrees longitude, the UTC origin point. Daylight saving time is not applicable to UTC.

Converting between entity and EDM types

When a client sends a request to the REST data service, the date and time is represented as a GMT time zone time, like the following example:

```
"2000-02-29T21:30:30.654123456"
```

The REST data service will then construct the appropriate Java temporal type instance and insert it into the entity in the data grid.

When a client requests a property which is a Java temporal type from the eXtreme Scale REST data service, the value is always normalized as a GMT time zone value. For example, if an entity java.util.Date is constructed as follows:

```
Calendar c = Calendar.getInstance();
c.clear();
c.set(2000, 1, 29, 21, 30, 30);
Date d = c.getTime();
```

The date and time are represented using the default time zone of the Java process because Calendar.getInstance() will create a Calendar object with local time zone. If the local time zone is CST, then the date, when retrieved from the REST data service will be the GMT representation of the time: `"2000-03-01T03:30:30"`

java.sql.Date normalization

An eXtreme Scale entity can define an attribute with Java type java.sql.Date. This data type does not include the time and is normalized by the REST data service. This means that the eXtreme Scale runtime environment does not store any hours, minutes, seconds, or milliseconds information in the java.sql.Date attribute. Regardless of the time zone offset, the date is always represented as a local date.

For example, if the client updates a java.sql.Date property with the value “2009-01-01T03:00:00”, the REST data service, which is in the CST time zone (-06:00), will simply create a java.sql.Date instance of which the time is set to “2009-01-01T00:00:00” of the local CST time. There is no time zone conversion done to create the java.sql.Date value. When the REST service client retrieves the value of this attribute, it will be displayed as “2009-01-01T00:00:00Z”. If a time zone conversion were done, the value would be displayed as having the date of “2008-12-31”, which would be incorrect.

java.sql.Time normalization

Similar to java.sql.Date, the java.sql.Time values are normalized and do not include date information. This means that the eXtreme Scale run time does not store the year, month or day. The time is stored using the GMT time from the epoch January 1, 1970, which is consistent with the java.sql.Time implementation.

For example, if the client updates a java.sql.Time property with the value "2009-01-01T03:00:00", the REST data service, will create a java.sql.Time instance with the milliseconds set to 3*60*60*1000, which is equal to 3 hours. When the rest service retrieves the value, it will be displayed as "1970-01-01:03:00:00Z".

Associations

Associations define the relationship between two peer entities. The eXtreme Scale REST service reflects the associations modeled with entities defined with eXtreme Scale annotated entities or entities defined using an entity descriptor XML file.

Association maintenance

The eXtreme Scale REST data service does not support referential integrity constraints. The client should ensure that references are updated when entities are removed or added. If a target entity of an association is removed from the data grid, but the link between the source and target entity is not removed, then the link is broken. The eXtreme Scale REST data service and EntityManager API tolerates broken links and logs the broken links as CWPRJ1022W warnings. Broken associations are removed from the request payload.

Use a batch request to group association updates in a single transaction to avoid broken links. See the following section for details on batch requests.

The ADO.NET Entity Data Model ReferentialConstraint element is not used by the eXtreme Scale REST data service.

Association multiplicity

Entities can have multi-valued associations or single-valued associations. Multi-valued associations, or collections, are one-to-many or many-to-many associations. Single-valued associations are one-to-one or many-to-one associations.

In a partitioned data grid, all entities should have a single-valued key-association path to a root entity. Another section of this topic shows how to define a key association. Because the root entity is used to partition the entity, many-to-many associations are not allowed for partitioned data grids. For an example on how to model a relational entity schema for a partitioned data grid, see Scalable data model in eXtreme Scale.

The following example describes how the EntityManager API association types, modeled using annotated Java classes map to the ADO.NET Entity Data Model:

```
@Entity
public class Customer {
    @Id String customerId;
    @OneToOne TaxInfo taxInfo;
    @ManyToOne Address homeAddress;
    @OneToMany Collection<Order> orders;
    @ManyToMany Collection<SalesPerson> salespersons;
}

<Association Name="Customer_TaxInfo">
    <End Type="Model1.Customer" Role="Customer" Multiplicity="1" />
    <End Type="Model1.TaxInfo " Role="TaxInfo" Multiplicity="1" />
</Association>
<Association Name="Customer_Address">
    <End Type="Model1.Customer" Role="Customer" Multiplicity="1" />
    <End Type="Model1.Address" Role="TaxInfo" Multiplicity="*" />
</Association>
<Association Name="Customer_Order">
    <End Type="Model1.Customer" Role="Customer" Multiplicity="*" />
    <End Type="Model1.Order" Role="TaxInfo" Multiplicity="1" />
</Association>
<Association Name="Customer_SalesPerson">
    <End Type="Model1.Customer" Role="Customer" Multiplicity="*" />
    <End Type="Model1.SalesPerson" Role="TaxInfo" Multiplicity="*" />
</Association>
```

Bi-directional and uni-directional associations

Entities associations can be uni-directional or bi-directional. By specifying the "mappedBy" attribute on the @OneToOne, @OneToMany or @ManyToMany annotation or the "mapped-by" attribute on the one-to-one, one-to-many or many-to-many XML attribute tag, the entity becomes bi-directional. The OData protocol currently requires all entities to be bi-directional, allowing clients to generate navigation paths in both directions. The eXtreme Scale EntityManager API allows modeling uni-directional associations which can save memory and simplify maintenance of the associations. If a uni-directional association is used, the REST data services client must only navigate through the association using the defined association.

For example: If a uni-directional many-to-one association is defined between Address and Country, the following URI is not allowed:

/restservice/CustomerGrid/Country('USA')/addresses

Key associations

Single-valued associations (one-to-one and many-to-one) can also be included as all or part of the entities key. This is known as a key-association.

Key associations are required when using a partitioned data grid. The key association must be defined for all child entities in a partitioned entity schema. The OData protocol requires that all entities are directly addressable. This means that the key in the child entity must include the key used for partitioning.

In the following example, Customer has a one-to-many association to Order. The Customer entity is the root entity and the customerId attribute is used to partition the entity. Order has included the Customer as part of its identity:

```
@Entity(schemaRoot="true")
public class Customer {
    @Id String customerId;
    @OneToMany(mappedBy="customer") Order orders
}

@Entity
public class Order {
    @Id int orderId;
    @Id @ManyToOne Customer customer;
    java.util.Date orderDate;
}
```

When the REST data service generates the EDMX document for this model, the Customer key fields are automatically included as part of the Order entity:

```
<EntityType Name="Order">
<Key>
```

```
    <PropertyRef Name="orderId"/>
    <PropertyRef Name="customer_customerId"/>
</Key>

<Property Name="orderId" Type="Edm.Int64" Nullable="false"/>
<Property Name="customer_customerId" Type="Edm.String"
    Nullable="false"/>
<Property Name="orderDate" Type="Edm.DateTime" Nullable="true"/>
<NavigationProperty Name="customer"
    Relationship="NorthwindGridModel.Customer_orders"
    FromRole="Order" ToRole="Customer"/>

<NavigationProperty Name="orderDetails"
    Relationship="NorthwindGridModel.Order_orderDetails"
    FromRole="Order" ToRole="OrderDetail"/>
</EntityType>
```

When an entity is created, the key must never change. This means if the key association between a child entity and its parent must change, the child entity must be removed and re-created with a different parent. In a partitioned data grid, this will require two different batch change sets since the move will likely involve more than one partition.

Cascading operations

The EntityManager API allows a flexible cascade policy. Associations can be marked to cascade a persist, remove, invalidate or merge operation. Such cascade operations can happen on one or both sides of a bi-directional association.

The OData protocol only allows cascade delete operations on the single-side of the association. The CascadeType.REMOVE annotation or cascade-remove XML attribute cannot be defined on both sides of a one-to-one bi-directional association or on the many-side of a one-to-many association. The following example illustrates a valid Cascade.REMOVE bi-directional association:

```
@Entity(schemaRoot="true")
public class Customer {
    @Id String customerId;
    @OneToMany(mappedBy="customer", cascade=CascadeType.REMOVE)
    Order orders
}

@Entity
public class Order {
    @Id int orderId;
    @Id @ManyToOne Customer customer;
    java.util.Date orderDate;
}
```

The resulting EDMX association looks as follows:

```
<Association Name="Customer_orders">
    <End Type="NorthwindGridModel.Customer" Role="Customer"
        Multiplicity="1">
        <OnDelete Action="Cascade"/>
    </End>
    <End Type="NorthwindGridModel.Order" Role="Order"
        Multiplicity="*"/>
</Association>
```

Related reference:
Server properties file
Client properties file
ObjectGrid descriptor XML file
Deployment policy descriptor XML file
Entity metadata descriptor XML file
Security descriptor XML file
Spring descriptor XML file
Liberty profile configuration files

ObjectGrid descriptor XML file

To configure WebSphere® eXtreme Scale, use an ObjectGrid descriptor XML file and the ObjectGrid API.

In the following sections, sample XML files are provided to illustrate various configurations. Each element and attribute of the XML file is defined. Use the ObjectGrid descriptor XML schema to create the descriptor XML file. See objectGrid.xsd file for an example of the ObjectGrid descriptor XML schema.

A modified version of the original companyGrid.xml file is used. The following companyGridSingleMap.xml file is like the companyGrid.xml file. The companyGridSingleMap.xml file has one map, and the companyGrid.xml file has four maps. The elements and attributes of the file are described in detail following the example.

```
<?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="CompanyGrid">
```

```
                                        <backingMap name="Customer"/>
                                </objectGrid>
                        </objectGrids>
</objectGridConfig>
```

objectGridConfig element

The objectGridConfig element is the top-level element of the XML file. Write this element in your eXtreme Scale XML document as shown in the preceding example. This element sets up the namespace of the file and the schema location. The schema is defined in the objectGrid.xsd file.

- Number of occurrences: One
- Child element: objectGrids element and backingMapPluginCollections element

objectGrids element

The objectGrids element is a container for all the objectGrid elements. In the companyGridSingleMap.xml file, the objectGrids element contains one objectGrid, the CompanyGrid objectGrid.

- Number of occurrences: One or more
- Child element: objectGrid element

objectGrid element

Use the objectGrid element to define an ObjectGrid. Each of the attributes on the objectGrid element corresponds to a method on the ObjectGrid interface.

- Number of occurrences: One to many
- Child element: bean element, backingMap element, and querySchema element

Attributes

name
: Specifies the name that is assigned to ObjectGrid. The XML validation fails if this attribute is missing. (Required)

securityEnabled
: Enables security at the ObjectGrid level, which enables the access authorizations to the data in the map, when you set the attribute to `true`. The default is `true`. (Optional)

authorizationMechanism
: Sets the authorization mechanism for the element. You can set the attribute to one of two values: `AUTHORIZATION_MECHANISM_JAAS` or `AUTHORIZATION_MECHANISM_CUSTOM`. The default is `AUTHORIZATION_MECHANISM_JAAS`. You must set the securityEnabled attribute to `true` for the authorizationMechanism attribute to take effect. (Optional)

permissionCheckPeriod
: Specifies an integer value in seconds that indicates how often to check the permission that is used to allow a client access. The default is `0`. When you set the attribute value `0`, every get, put, update, remove, or evict method call asks the authorization mechanism, either Java™ Authentication and Authorization Service (JAAS) authorization or custom authorization, to check if the current subject has permission. A value greater than 0 indicates the number of seconds to cache a set of permissions before returning to the authorization mechanism to refresh. You must set the securityEnabled attribute to `true` for the permissionCheckPeriod attribute to take effect. (Optional)

accessByCreatorOnlyMode
: Specifies if a user (represented by the Principal objects associated with it) other than the creator of a cache entry can access, update or delete that entry. The default value is `disabled` when not specified, allowing any user access to the cache entry. Valid values also include `complement` and `supersede`. The `complement` value enables creator-only access, and also enforces map authorization. The `supersede` value enables creator-only access, and disables map authorization. (Optional)

txTimeout
: 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. If you set the value to `0`, the default setting of 10 minutes is used as the transaction timeout. (Optional)

txIsolation
: Sets the default transaction isolation level for all sessions created by the ObjectGrid. Define one of the following values:

  - `REPEATABLE_READ` (default): Specifies that dirty reads and non-repeatable reads are prevented; phantom reads can occur. This level prohibits a transaction from reading an uncommitted cache entry. It also prohibits the following scenario: one transaction reads an entry, a second transaction alters the entry, and the first transaction rereads the entry, getting different values the second time (a "non-repeatable read").
  - `READ_UNCOMMITTED`: Specifies that dirty reads, non-repeatable reads and phantom reads can occur. Cache entries can be changed by one transaction and read by another transaction before any changes in that entry have been committed. If any of the changes are rolled back, the second transaction retrieves an entry that is not valid.
  - `READ_COMMITTED`: Specifies that dirty reads are prevented; non-repeatable reads and phantom reads can occur. This level only prohibits a transaction from reading a cache entry that has uncommitted changes.

entityMetadataXMLFile
: Specifies the path to the entity descriptor XML file that defines the entity schema. Define entities before you start the catalog server so that each entity can bind with a backing map.

  - **For a relative directory:** Specify the location relative to the location of the objectgrid.xml file.
  - **For an absolute path:** Specify the location with a URL format, such as file:// or http://.

  (Optional)

```
<objectGrid
(1)     name="objectGridName"
```

```
(2)     securityEnabled="true" | "false"
(3)     authorizationMechanism="AUTHORIZATION_MECHANISM_JASS" | "AUTHORIZATION_MECHANISM_CUSTOM"
(4)     permissionCheckPeriod="permission_check_period"
(5)     txTimeout="seconds"
(6)     entityMetadataXMLFile="URL"
/>
```

In the following example, the companyGridObjectGridAttr.xml file demonstrates one way to configure the attributes of an objectGrid element. Security is enabled, the authorization mechanism is set to JAAS, and the permission check period is set to 45 seconds. The file also registers entities by specifying an entityMetadataXMLFile attribute.

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlnc: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="CompanyGrid" securityEnabled="true"
                        authorizationMechanism="AUTHORIZATION_MECHANISM_JASS"
                        permissionCheckPeriod="45"
                        entityMetadataXMLFile="companyGridEntities.xml">
                        <backingMap name="Customer"/>
                </objectGrid>
        </objectGrids>
</objectGridConfig>
```

The following code sample demonstrates the programmatic approach to achieving the same configuration as the companyGridObjectGridAttr.xml file in the preceding example.

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid companyGrid = objectGridManager.createObjectGrid("CompanyGrid", false);

companyGrid.setSecurityEnabled();
companyGrid.setAuthorizationMechanism(SecurityConstants.AUTHORIZATION_MECHANISM_JAAS);
companyGrid.setPermissionCheckPeriod(45);
companyGrid.registerEntities(new URL("file:companyGridEntities.xml"));
```

backingMap element

The backingMap element is used to define a BackingMap instance on an ObjectGrid. Each of the attributes on the backingMap element corresponds to a method on the BackingMap interface. For details, see BackingMap interface.

- Number of occurrences: Zero to many
- Child element: timeBasedDBUpdate element

backingMap attributes

copyKey
: Specifies if the a copy of the key is required when a map entry is created. Copying the key object allows the application to use the same key object for each ObjectMap operation. Set the value to `true` to copy the key object when a map entry is created. The default value is `false`. (Optional)

CopyMode
: 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. 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.

  For more information about these settings, see Tuning the copy mode.

evictionTriggers
: Sets the types of additional eviction triggers to use. All evictors for the backing map use this list of additional triggers. To avoid an IllegalStateException, this attribute must be called before the ObjectGrid.initialize() method. Also, note that the ObjectGrid.getSession() method implicitly calls the

ObjectGrid.initialize() method if the method has yet to be called by the application. Entries in the list of triggers are separated by semicolons. Current eviction triggers include MEMORY_USAGE_THRESHOLD. For more information, see Plug-ins for evicting cache objects. (Optional)

lockStrategy
: 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`. The default value is `OPTIMISTIC`. (Optional)

  The optimistic locking strategy is typically used when a map does not have a loader plug-in, is mostly read and not frequently written to or updated, and the locking is not provided by the persistence manager using eXtreme Scale as a side cache or by the application. An exclusive lock is obtained on a map entry that is inserted, updated, or removed at commit time. The lock ensures that the version information cannot be changed by another transaction while the transaction being committed is performing an optimistic version check.

  The pessimistic locking strategy is typically used for a map that does not have a loader plug-in, and locking is not provided by a persistence manager using eXtreme Scale as a side cache, by a loader plug-in, or by the application. The pessimistic locking strategy is used when the optimistic locking strategy fails too often because update transactions frequently collide on the same map entry.

  The no locking strategy indicates that the internal LockManager is not needed. Concurrency control is provided outside of eXtreme Scale, either by the persistence manager using eXtreme Scale as a side cache or application, or by the loader plug-in that uses database locks to control concurrency.

lockTimeout
: Sets the lock timeout that is used by the lock manager for the BackingMap instance. Set the lockStrategy attribute to `OPTIMISTIC` or `PESSIMISTIC` to create a lock manager for the BackingMap instance. To prevent deadlocks from occurring, the lock manager has a default timeout value of 15 seconds. If the timeout limit is exceeded, a LockTimeoutException exception occurs. The default value of 15 seconds is sufficient for most applications, but on a heavily loaded system, a timeout might occur when no deadlock exists. Use the lockTimeout attribute to increase the value from the default to prevent false timeout exceptions from occurring. Set the lockStrategy attribute to `NONE` to specify the BackingMap instance use no lock manager. (Optional)

name
: Specifies the name that is assigned to the backingMap instance. If this attribute is missing, the XML validation fails. (Required)

nullValuesSupported
: Set the value to `true` to support null values in the ObjectMap. When null values are supported, a get operation that returns null might mean that the value is null or that the map does not contain the key that is passed to the method. The default value is `true`. (Optional)

numberOfBuckets
: The BackingMap instance uses a hash map for implementation. The numberOfBuckets attribute specifies the number of buckets for the BackingMap instance to use. If multiple entries exist in the BackingMap, more buckets lead to better performance because the risk of collisions is lower as the number of buckets increases. More buckets also lead to more concurrency. Specify a value of `0` to disable the near cache on a client. When you set the value to `0` for a client, set the value in the client override ObjectGrid XML descriptor file only. (Optional)

numberOfLockBuckets
: The lock manager uses a hash map to track entries that are locked by one or more transactions. The numberOfLockBuckets attribute sets the number of lock buckets that are used by the lock manager for the BackingMap instance. Set the lockStrategy attribute to `OPTIMISTIC` or `PESSIMISTIC` to create a lock manager for the BackingMap instance. If many entries exist, more lock buckets lead to better performance because the risk of collisions is lower as the number of buckets grows. More lock buckets also lead to more concurrency. Set the lockStrategy attribute to `NONE` to specify the BackingMap instance use no lock manager. (Optional)

pluginCollectionRef
: Specifies a reference to a backingMapPluginCollection plug-in. The value of this attribute must match the ID attribute of a backingMapCollection plug-in. Validation fails if no matching ID exists. Set the attribute to reuse BackingMap plug-ins. (Optional)

preloadMode
: Sets the preload mode if a loader plug-in is set for this BackingMap instance. The default value is `false`. If the attribute is set to `true`, the Loader.preloadMap(Session, BackingMap) method is invoked asynchronously. Otherwise, running the method is blocked when loading data so that the cache is unavailable until preload completes. Preloading occurs during initialization. (Optional)

readOnly
: Sets a BackingMap instance as read/write when you specify the attribute as `false`. When you specify the attribute as `true`, the BackingMap instance is read-only. (Optional)

template
: Specifies if dynamic maps can be used. Set this value to `true` if the BackingMap map is a template map. Template maps can be used to dynamically create maps after the ObjectGrid is started. Calls to Session.getMap(String) result in dynamic maps being created if the name passed to the method matches the regular expression specified in the name attribute of the backingMap. The default value is `false`. (Optional)

timeToLive
: Specifies in seconds how long each map entry is present. The default value of `0` means that the map entry is present forever, or until the application explicitly removes or invalidates the entry. Otherwise, the TTL evictor evicts the map entry based on this value. (Optional)

ttlEvictorType
: Specifies how the expiration time of a BackingMap entry is computed. Set this attribute to one of these values: `CREATION_TIME`, `LAST_ACCESS_TIME`, `LAST_UPDATE_TIME`, or `NONE`. The `CREATION_TIME` value indicates that an entry expiration time is the sum of the creation time of the entry plus the timeToLive attribute value. The `LAST_ACCESS_TIME` value indicates that an entry expiration time is the sum of the last access time of the entry (whether the entry was updated or merely read), plus the timeToLive attribute value. The `LAST_UPDATE_TIME` value indicates that an entry expiration time is the sum of the last update time of the entry plus the timeToLive attribute value. The `NONE` value, which is the default, indicates that an entry has no expiration time and is present in the BackingMap instance until the application explicitly removes or invalidates the entry. (Optional)

valueInterfaceClassName
: Specifies a class that is required when you set the CopyMode attribute to `COPY_ON_WRITE`. This attribute is ignored for all other modes. The `COPY_ON_WRITE` value uses a proxy when ObjectMap.get method calls are made. The proxy ensures that a copy of the value is not made unless the application calls a set method on the class that is specified as the valueInterfaceClassName attribute. (Optional)

viewRef
: Specifies that the backingMap is a view map. (Optional)

writeBehind
: Specifies that the write-behind support is enabled with write-behind parameters (Optional). Write-behind parameters consist of a maximum update time and a maximum key update count. The format of the write-behind parameter is "`[T(time)][;][C(count)]`". The database is updated when one of the following events occurs:

  - The maximum update time, specified in seconds, has passed since the last update.
  - The number of available updates in the queue map has reached the maximum update count.

For more information, see Write-behind caching.

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.

```
<backingMap
(1)                     name="objectGridName"
(2)                     readOnly="true" | "false"
(3)                     template="true" | "false"
(4)                     pluginCollectionRef="reference to backingMapPluginCollection"
(5)                     numberOfBuckets="number of buckets"
(6)                     preloadMode="true" | "false"
(7)                     lockStrategy="OPTIMISTIC" | "PESSIMISTIC" | "NONE"
(8)                     numberOfLockBuckets="number of lock buckets"
(9)                     lockTimeout="lock timeout"
(10)                    copyMode="COPY_ON_READ_AND_COMMIT" | "COPY_ON_READ" | "COPY_ON_WRITE"
                                                   | "NO_COPY" | "COPY_TO_BYTES"
(11)                    valueInterfaceClassName="value interface class name"
(12)                    copyKey="true" | "false"
(13)                    nullValuesSupported="true" | "false"
(14)                    ttlEvictorType="CREATION_TIME" | "LAST_ACCESS_TIME" | "LAST_UPDATE_TIME" | NONE"
(15)                    timeToLive="time to live"
(16)                    streamRef="reference to a stream"
(17)                    viewRef="reference to a view"
(18)                    writeBehind="write-behind parameters"
/>
```

In the following example, the companyGridBackingMapAttr.xml file is used to demonstrate a sample backingMap configuration.

```
<?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="CompanyGrid">
                        <backingMap name="Customer" readOnly="true"
                                        numberOfBuckets="641" preloadMode="false"
                                        lockStrategy="OPTIMISTIC" numberOfLockBuckets="409"
                                        lockTimeout="30" copyMode="COPY_ON_WRITE"

valueInterfaceClassName="com.ibm.websphere.samples.objectgrid.CounterValueInterface"
                                        copyKey="true" nullValuesSupported="false"
                                        ttlEvictorType="LAST_ACCESS_TIME" timeToLive="3000"/>
                </objectGrid>
        </objectGrids>
</objectGridConfig>
```

The following sample code demonstrates the programmatic approach to achieve the same configuration as the companyGridBackingMapAttr.xml file in the preceding example:

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid companyGrid = objectGridManager.createObjectGrid("CompanyGrid", false);

BackingMap customerMap = companyGrid.defineMap("Customer");
customerMap.setReadOnly(true);
customerMap.setNumberOfBuckets(641);
customerMap.setPreloadMode(false);
customerMap.setLockStrategy(LockStrategy.OPTIMISTIC);
customerMap.setNumberOfLockBuckets(409);
customerMap.setLockTimeout(30);

// when setting copy mode to COPY_ON_WRITE, a valueInterface class is required
customerMap.setCopyMode(CopyMode.COPY_ON_WRITE,
        com.ibm.websphere.samples.objectgrid.CounterValueInterface.class);
customerMap.setCopyKey(true);
customerMap.setNullValuesSupported(false);
customerMap.setTtlEvictorType(TTLType.LAST_ACCESS_TIME);
customerMap.setTimeToLive(3000); // set time to live to 50 minutes
```

bean element

Use the bean element to define plug-ins. You can attach plug-ins to objectGrid and BackingMap elements.

- Number of occurrences within the objectGrid element: Zero to many
- Number of occurrences within the backingMapPluginCollection element: Zero to many
- Child element: property element

Attributes

id

Specifies the type of plug-in to create. (Required)
The valid plug-ins for a bean that is a child element of the objectGrid element are included in the following list:

- TransactionCallback plug-in
- ObjectGridEventListener plug-in
- SubjectSource plug-in

- SubjectValidation plug-in

The valid plug-ins for a bean that is a child element of the backingMapPluginCollection element are included in the following list:

- Loader plug-in
- ObjectTransformer plug-in
- OptimisticCallback plug-in
- Evictor plug-in
- MapEventListener plug-in
- MapIndex plug-in

className
: Specifies the name of the class or spring bean to instantiate to create the plug-in. The class must implement the plug-in type interface. For example, if you specify `ObjectGridEventListener` as the value for the id attribute, the className attribute value must refer to a class that implements the ObjectGridEventListener interface. The className or osgiService is required.

osgiService
: Specifies the name of the OSGi service to look up in the OSGi service manager. When running in the Eclipse Equinox OSGi framework with the Eclipse Gemini or Apache Aries Blueprint container, plug-ins can be defined using an OSGi Blueprint XML file. The other bean properties are not typically used when using an osgiService name, since the bean properties are configured in the Blueprint configuration file. See Configuring OSGi-enabled plug-ins using the ObjectGrid descriptor XML file for more information. The className or osgiService is required.

```
<bean
(1)     id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsogref_TransactionCallback"
| "ObjectGridEventListener" |"SubjectSource" |
     "SubjectValidation" | "Loader" | "ObjectTransformer" |
    "OptimisticCallback" | "Evictor" | "MapEventListener" | "MapIndexPlugin"
(2)     className="class name" | "(spring)bean name"
/>
```

In the following example, the companyGridBean.xml file is used to demonstrate how to configure plug-ins using the bean element. An ObjectGridEventListener plug-in is added to the CompanyGrid ObjectGrid. The className attribute for this bean is the com.ibm.websphere.objectgrid.plugins.builtins.TranPropListener class. This class implements the com.ibm.websphere.objectgrid.plugins.ObjectGridEventListener interface as required.

A BackingMap plug-in is also defined in the companyGridBean.xml file. An evictor plug-in is added to the Customer BackingMap instance. Because the bean ID is Evictor, the className attribute must specify a class that implements the com.ibm.websphere.objectgrid.plugins.Evictor interface. The com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor class implements this interface. The backingMap references its plug-ins using the pluginCollectionRef attribute.

```
<?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="CompanyGrid">
                        bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsogref_ObjectGridEventListener"
                                className="com.ibm.websphere.objectgrid.plugins.builtins.TranPropListener"/>
                        <backingMap name="Customer"
                                pluginCollectionRef="customerPlugins"/>
                </objectGrid>
        </objectGrids>
        <backingMapPluginCollections>
                <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsogref_customerPlugins">
                        <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsogref_Evictor"
                                        className="com.ibm.websphere.objectGrid.plugins.builtins.LRUEvictor/>
                </backingMapPluginCollection>
        </backingMapPluginCollections>
</objectGridConfig>
```

The following code sample demonstrates the programmatic approach to achieving the same configuration as the companyGridBean.xml file in the preceding example.

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid companyGrid = objectGridManager.createObjectGrid("CompanyGrid", false);
TranPropListener tranPropListener = new TranPropListener();
companyGrid.addEventListener(tranPropListener);

BackingMap customerMap = companyGrid.defineMap("Customer");
Evictor lruEvictor = new com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor();
customerMap.setEvictor(lruEvictor);
```

For more details about using plug-ins, consult Java plug-ins overview.

property element

Use the property element to add properties to plug-ins. The name of the property must correspond to a set method on the class referenced by the containing bean.

- Number of occurrences: Zero to many
- Child element: None

Attributes

name
: Specifies the name of the property. The value that is assigned to this attribute must correspond to a set method on the class that is provided as the className attribute on the containing bean. For example, if you set the className attribute of the bean to `com.ibm.MyPlugin`, and the name of the property that is provided is `size`, the `com.ibm.MyPlugin` class must have a setSize method. (Required)

type
: Specifies the type of the property. The type is passed to the set method that is identified by the name attribute. The valid values are the Java primitives, the java.lang counterparts, and java.lang.String. The name and type attributes must correspond to a method signature on the className attribute of the bean. For example, if you set the name as `size` and the type as `int`, a setSize(int) method must exist on the class that is specified as the className attribute for the bean. (Required)

value
: Specifies the value of the property. This value is converted to the type that is specified by the type attribute, and is then used as a parameter in the call to the set method that is identified by the name and type attributes. The value of this attribute is not validated in any way. (Required)

description
: Describes the property. (Optional)

```
<bean
(1)     name="name"
(2)     type="java.lang.String" | "boolean" | "java.lang.Boolean" | "int" |
                        "java.lang.Integer" | "double" | "java.lang.Double" | "byte" |
                        "java.lang.Byte" | "short" | "java.lang.Short" | "long" |
                        "java.lang.Long" | "float" | "java.lang.Float" | "char" |
                        "java.lang.Character"
(3)     value="value"
(4)     description="description"
/>
```

In the following example, the companyGridProperty.xml file is used to demonstrate how to add a property element to a bean. In this example, a property with the name maxSize and type int is added to an evictor. The com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor class has a method signature that matches the setMaxSize(int) method. An integer value of 499 is passed to the setMaxSize(int) method on the com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor class.

```
<?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="CompanyGrid">
                        <backingMap name="Customer"
                                pluginCollectionRef="customerPlugins"/>
                </objectGrid>
        </objectGrids>
        <backingMapPluginCollections>
                <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsogref_customerPlugins">
                        <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsogref_Evictor"
                                        className="com.ibm.websphere.objectGrid.plugins.builtins.LRUEvictor>
                                        <property name="maxSize" type="int" value="449"
                                                description="The maximum size of the LRU Evictor"/>
                        </bean>
                </backingMapPluginCollection>
        </backingMapPluginCollections>
</objectGridConfig>
```

The following code sample demonstrates the programmatic approach to achieving the same configuration as the companyGridProperty.xml file in the preceding example.

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid companyGrid = objectGridManager.createObjectGrid("CompanyGrid", false);

BackingMap customerMap = companyGrid.defineMap("Customer");

LRUEvictor lruEvictor = new com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor();
// if the XML file is used instead,
// the property that was added would cause the following call to occur
lruEvictor.setMaxSize(449);
customerMap.setEvictor(lruEvictor);
```

backingMapPluginsCollections element

The backingMapPluginsCollections element is a container for all the backingMapPluginCollection elements. In the companyGridProperty.xml file in the preceding section, the backingMapPluginCollections element contains one backingMapPluginCollection element with the ID `customerPlugins`.

- Number of occurrences: Zero to one
- Child element: backingMapPluginCollection element

backingMapPluginCollection element

The backingMapPluginCollection element defines the BackingMap plug-ins, and is identified by the id attribute. Specify the pluginCollectionRef attribute to reference the plug-ins. When configuring several BackingMaps plug-ins similarly, each BackingMap can reference the same backingMapPluginCollection element.

- Number of occurrences: Zero to many
- Child element: bean element

Attributes

id

Identifies the backingMapPluginCollection, and is referenced by the pluginCollectionRef attribute of the backingMap element. Each ID must be unique. If the value of a pluginCollectionRef attribute does not match the ID of one backingMapPluginCollection element, XML validation fails. Any number of backingMap elements can reference a single backingMapPluginCollection element. (Required)

```
<backingMapPluginCollection
(1)                     id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsogref_id"
/>
```

In the following example, the companyGridCollection.xml file is used to demonstrate how to use the backingMapPluginCollection element. In this file, the Customer BackingMap uses the customerPlugins backingMapPluginCollection to configure the Customer BackingMap with an LRUEvictor. The Item and OrderLine BackingMaps reference the collection2 backingMapPluginCollection. These BackingMaps each have an LFUEvictor set.

```
<?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="CompanyGrid">
                        <backingMap name="Customer"
                                pluginCollectionRef="customerPlugins"/>
                        <backingMap name="Item" pluginCollectionRef="collection2"/>
                        <backingMap name="OrderLine"
                                pluginCollectionRef="collection2"/>
                        <backingMap name="Order"/>
                </objectGrid>
        </objectGrids>
        <backingMapPluginCollections>
                <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsogref_customerPlugins">
                        <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsogref_Evictor"
                                        className="com.ibm.websphere.objectGrid.plugins.builtins.LRUEvictor/>
                </backingMapPluginCollection>
                <backingMapPluginCollection
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsogref_collection2">
                        <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsogref_Evictor"
                                        className="com.ibm.websphere.objectgrid.plugins.builtins.LFUEvictor/>
                        <bean
id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsogref_OptimisticCallback"

className="com.ibm.websphere.samples.objectgrid.EmployeeOptimisticCallBackImpl"/>
                </backingMapPluginCollection>
                </backingMapPluginCollections>
</objectGridConfig>
```

The following code sample demonstrates the programmatic approach to achieving the same configuration as the companyGridCollection.xml file in the preceding example.

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

ObjectGrid companyGrid = objectGridManager.createObjectGrid("CompanyGrid", false);
BackingMap customerMap = companyGrid .defineMap("Customer");
LRUEvictor customerEvictor = new LRUEvictor();
customerMap.setEvictor(customerEvictor);

BackingMap itemMap = companyGrid.defineMap("Item");
LFUEvictor itemEvictor = new LFUEvictor();
itemMap.setEvictor(itemEvictor);

BackingMap orderLineMap = companyGrid.defineMap("OrderLine");
LFUEvictor orderLineEvictor = new LFUEvictor();
orderLineMap.setEvictor(orderLineEvictor);

BackingMap orderMap = companyGrid.defineMap("Order");
```

querySchema element

The querySchema element defines relationships between BackingMaps and identifies the type of object in each map. This information is used by ObjectQuery to translate query language strings into map access calls. For more information, see Configuring an ObjectQuery schema.

- Number of occurrences: Zero to one
- Child element: mapSchemas element, relationships element

mapSchemas element

Each querySchema element has one mapSchemas element that contains one or more mapSchema elements.

- Number of occurrences: One
- Child element: mapSchema element

mapSchema element

A mapSchema element defines the type of object that is stored in a BackingMap and instructions on how to access the data.

- Number of occurrences: One or more
- Child element: None

Attributes

mapName
: Specifies the name of the BackingMap to add to the schema. (Required)

valueClass
: Specifies the type of object that is stored in the value portion of the BackingMap. (Required)

primaryKeyField
: Specifies the name of the primary key attribute in the valueClass attribute. The primary key must also be stored in the key portion of the BackingMap. (Optional)

accessType
: Identifies how the query engine introspects and accesses the persistent data in the valueClass object instances. If you set the value to `FIELD`, the class fields are introspected and added to the schema. If the value is `PROPERTY`, the attributes that are associated with get and is methods are used. The default value is `PROPERTY`. (Optional)

```
<mapSchema
(1)                     mapName="backingMapName"
(2)                     valueClass="com.mycompany.OrderBean"
(3)                     primaryKeyField="orderId"
(4)                     accessType="PROPERTY" | "FIELD"
/>
```

In the following example, the companyGridQuerySchemaAttr.xml file is used to demonstrate a sample mapSchema configuration.

```
<?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="CompanyGrid">
                        <backingMap name="Order"/>
                        <backingMap name="Customer"/>

                        <querySchema>
                                <mapSchemas>
                                        <mapSchema mapName="Order"
                                                valueClass="com.mycompany.OrderBean"
                                                primaryKeyField="orderNumber"
                                                accessType="FIELD"/>
                                        <mapSchema mapName="Customer"
                                                valueClass="com.mycompany.CustomerBean"
                                                primaryKeyField="id"
                                                accessType="FIELD"/>
                                </mapSchemas>
                        </querySchema>
                </objectGrid>
        </objectGrids>
</objectGridConfig>
```

The following code sample demonstrates the programmatic approach to achieving the same configuration as the companyGridQuerySchemaAttr.xml file in the preceding example.

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid companyGrid = objectGridManager.createObjectGrid("CompanyGrid", false);
companyGrid.defineMap("Order");
companyGrid.defineMap("Customer");

// Define the schema
QueryConfig queryCfg = new QueryConfig();
queryCfg.addQueryMapping(new QueryMapping(
    "Order", OrderBean.class.getName(), "orderNumber", QueryMapping.FIELD_ACCESS));
queryCfg.addQueryMapping(new QueryMapping(
    "Customer", CustomerBean.class.getName(), "id", QueryMapping.FIELD_ACCESS));
companyGrid.setQueryConfig(queryCfg);
```

relationships element

Each querySchema element has zero or one relationships element that contains one or more relationship elements.

- Number of occurrences: Zero or one
- Child element: relationship element

relationship element

A relationship element defines the relationship between two BackingMaps and the attributes in the valueClass attribute that bind the relationship.

- Number of occurrences: One or more
- Child element: None

Attributes

source
: Specifies the name of the valueClass of the source side of a relationship. (Required)

target
: Specifies the name of the valueClass of the target side of a relationship. (Required)

relationField
: Specifies the name of the attribute in the source valueClass that refers to the target. (Required)

invRelationField
: Specifies the name of the attribute in the target valueClass that refers to the source. If this attribute is not specified, the relationship is one directional. (Optional)

```
<mapSchema
(1)                     source="com.mycompany.OrderBean"
(2)                     target="com.mycompany.CustomerBean"
(3)                     relationField="customer"
(4)                     invRelationField="orders"
/>
```

In the following example, the companyGridQuerySchemaWithRelationshipAttr.xml file is used to demonstrate a sample mapSchema configuration that includes a bidirectional relationship.

```
<?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="CompanyGrid">
                        <backingMap name="Order"/>
                        <backingMap name="Customer"/>

                        <querySchema>
                                <mapSchemas>
                                        <mapSchema mapName="Order"
                                                valueClass="com.mycompany.OrderBean"
                                                primaryKeyField="orderNumber"
                                                accessType="FIELD"/>
                                        <mapSchema mapName="Customer"
                                                valueClass="com.mycompany.CustomerBean"
                                                primaryKeyField="id"
                                                accessType="FIELD"/>
                                </mapSchemas>
                                <relationships>
                                        <relationship
                                                source="com.mycompany.OrderBean"
                                                target="com.mycompany.CustomerBean"
                                                relationField="customer"/>
                                                invRelationField="orders"/>
                                </relationships>
                        </querySchema>
                </objectGrid>
        </objectGrids>
</objectGridConfig>
```

The following code sample demonstrates the programmatic approach to achieving the same configuration as the companyGridQuerySchemaWithRelationshipAttr.xml file in the preceding example.

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid companyGrid = objectGridManager.createObjectGrid("CompanyGrid", false);
companyGrid.defineMap("Order");
companyGrid.defineMap("Customer");

// Define the schema
QueryConfig queryCfg = new QueryConfig();
queryCfg.addQueryMapping(new QueryMapping(
    "Order", OrderBean.class.getName(), "orderNumber", QueryMapping.FIELD_ACCESS));
queryCfg.addQueryMapping(new QueryMapping(
    "Customer", CustomerBean.class.getName(), "id", QueryMapping.FIELD_ACCESS));
queryCfg.addQueryRelationship(new QueryRelationship(
     OrderBean.class.getName(), CustomerBean.class.getName(), "customer", "orders"));
companyGrid.setQueryConfig(queryCfg);
```

stream element

The stream element represents a stream to the stream query engine. Each attribute of the stream element corresponds to a method on the StreamMetadata interface.

- Number of occurrences: One to many
- Child element: basic element

Attributes

name
: Specifies the name of the stream. Validation fails if this attribute is not specified. (Required)

valueClass
: Specifies the class type of the value that is stored in the stream ObjectMap. The class type is used to convert the object to the stream events and to generate an SQL statement if the statement is not provided. (Required)

sql
: Specifies the SQL statement of the stream. If this property is not provided, a stream SQL is generated by reflecting the attributes or accessor methods on the valueClass attribute or by using the tuple attributes of the entity metadata. (Optional)

access
: Specifies the type to access the attributes of the value class. If you set the value to `FIELD`, the attributes are directly retrieved from the fields using Java reflection. Otherwise, accessor methods are used to read the attributes. The default value is `PROPERTY`. (Optional)

```
<stream
(1)     name="streamName"
(2)     valueClass="streamMapClassType"
(3)     sql="streamSQL create stream stockQuote
                        keyed by t ( transactionvolume INTEGER, price DECIMAL (9,2), issue VARCHAR(100) );"
(4)     access="PROPERTY" | "FIELD"
/>
```

view element

The view element represents a stream query view. Each stream element corresponds to a method on the ViewMetadata interface.

- Number of occurrences: One to many
- Child element: basic element, ID element

Attributes

name
: Specifies the name of the view. Validation fails if this attribute is not specified. (Required)

sql
: Specifies the SQL of the stream, which defines the view transformation. Validation fails if this attribute is not specified. (Required)

valueClass
: Specifies the class type of the value that is stored in this view of the ObjectMap. The class type is used to convert view events into the correct tuple format that is compatible with this class type. If the class type is not provided, a default format following the column definitions in the Stream Processing Technology Structured Query Language (SPTSQL) is used. If an entity metadata is defined for this view map, do not use the valueClass attribute. (Optional)

access
: Specifies the type to access the attributes of the value class. If you set the access type to `FIELD`, the column values are directly set to the fields using Java reflection. Otherwise, accessor methods are used to set the attributes. The default value is `PROPERTY`. (Optional)

```
<view
(1)                     name="viewName"
(2)                     valueClass="viewMapValueClass"
(3)                     sql="viewSQL CREATE VIEW last5MinuteAvgPrice AS
                                        SELECT issue, avg(price) as totalVolume
                                    FROM (SELECT * FROM stockQuote FETCH LATEST 5 MINUTES) group by issue;"/>
(4)                     access="PROPERTY" | "FIELD"
/>
```

basic element

The basic element is used to define a mapping from the attribute name in the value class or entity metadata to the column that is defined in the SPTSQL.

- Number of occurrences: Zero to many
- Child element: None

```
<basic
(1)                     name="attributeName"
(2)                     column="columnName"
/>
```

id element

The id element is used for a key attribute mapping.

- Number of occurrences: Zero to many
- Child element: None

```
<id
(1)                     name="idName"
(2)                     column="columnName"
/>
```

- objectGrid.xsd file
  Use the ObjectGrid descriptor XML schema to configure WebSphere eXtreme Scale.

Related concepts:

Evictors
Tuning evictors
Plug-ins for evicting cache objects
Custom evictors
Tuning the copy mode
CopyMode attribute
Improving performance with byte array maps
Related tasks:
Enabling evictors programmatically
Configuring evictors with XML files
Troubleshooting XML configuration
Related reference:
Server properties file
Client properties file
REST data service properties file
Deployment policy descriptor XML file
Entity metadata descriptor XML file
Security descriptor XML file
Spring descriptor XML file
Liberty profile configuration files
Related information:
API documentation
Getting started tutorial lesson 1.1: Defining data grids with configuration files

objectGrid.xsd file

Use the ObjectGrid descriptor XML schema to configure WebSphere® eXtreme Scale.

See the ObjectGrid descriptor XML file for descriptions of the elements and attributes defined in the objectGrid.xsd file. For information about the Spring objectgrid.xsd file, see Spring descriptor XML file.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:cc="http://ibm.com/ws/objectgrid/config"
xmlns:dgc="http://ibm.com/ws/objectgrid/config" elementFormDefault="qualified"
targetNamespace="http://ibm.com/ws/objectgrid/config">

    <xsd:element name="objectGridConfig">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element maxOccurs="1" minOccurs="1" name="objectGrids" type="dgc:objectGrids">
                    <xsd:unique name="objectGridNameUnique">
                          <xsd:selector xpath="dgc:objectGrid"/>
                          <xsd:field xpath="@name"/>
                    </xsd:unique>
                </xsd:element>
                <xsd:element maxOccurs="1" minOccurs="0" name="backingMapPluginCollections"
type="dgc:backingMapPluginCollections"/>
            </xsd:sequence>
        </xsd:complexType>

        <xsd:key name="backingMapPluginCollectionId">
            <xsd:selector xpath="dgc:backingMapPluginCollections/dgc:backingMapPluginCollection"/>
            <xsd:field xpath="@id"/>
        </xsd:key>

        <xsd:keyref name="pluginCollectionRef" refer="dgc:backingMapPluginCollectionId">
            <xsd:selector xpath="dgc:objectGrids/dgc:objectGrid/dgc:backingMap"/>
            <xsd:field xpath="@pluginCollectionRef"/>
        </xsd:keyref>
    </xsd:element>

    <xsd:complexType name="objectGrids">
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" name="objectGrid" type="dgc:objectGrid">
                <xsd:unique name="backingMapNameUnique">
                      <xsd:selector xpath="dgc:backingMap"/>
                      <xsd:field xpath="@name"/>
                </xsd:unique>
            </xsd:element>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="backingMapPluginCollections">
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="0" name="backingMapPluginCollection"
type="dgc:backingMapPluginCollection"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="objectGrid">
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="0" name="bean" type="dgc:bean"/>
```

```
            <xsd:element maxOccurs="unbounded" minOccurs="0" name="backingMap" type="dgc:backingMap"/>
            <xsd:element maxOccurs="1" minOccurs="0" name="querySchema" type="dgc:querySchema"/>
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string" use="required"/>
        <xsd:attribute name="authorizationMechanism" type="dgc:authorizationMechanism" use="optional"/>
        <xsd:attribute name="accessByCreatorOnlyMode" type="dgc:accessByCreatorOnlyMode" use="optional"/>
        <xsd:attribute name="securityEnabled" type="xsd:boolean" use="optional"/>
        <xsd:attribute name="txTimeout" type="xsd:int" use="optional"/>
        <xsd:attribute name="permissionCheckPeriod" type="xsd:int" use="optional"/>
        <xsd:attribute name="entityMetadataXMLFile" type="xsd:string" use="optional"/>
        <xsd:attribute name="initialState" type="dgc:initialState" use="optional"/>
        <xsd:attribute name="txIsolation" type="dgc:transactionIsolation" use="optional"/>
    </xsd:complexType>


    <xsd:complexType name="backingMap">
        <xsd:sequence>
           <xsd:element maxOccurs="1" minOccurs="0" name="timeBasedDBUpdate" type="dgc:timeBasedDBUpdate"/>
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string" use="required"/>
        <xsd:attribute name="readOnly" type="xsd:boolean" use="optional"/>
        <xsd:attribute name="pluginCollectionRef" type="xsd:string" use="optional"/>
        <xsd:attribute name="preloadMode" type="xsd:boolean" use="optional"/>
        <xsd:attribute name="lockStrategy" type="dgc:lockStrategy" use="optional"/>
        <xsd:attribute name="copyMode" type="dgc:copyMode" use="optional"/>
        <xsd:attribute name="valueInterfaceClassName" type="xsd:string" use="optional"/>
        <xsd:attribute name="numberOfBuckets" type="xsd:int" use="optional"/>
        <xsd:attribute name="nullValuesSupported" type="xsd:boolean" use="optional"/>
        <xsd:attribute name="lockTimeout" type="xsd:int" use="optional"/>
        <xsd:attribute name="numberOfLockBuckets" type="xsd:int" use="optional"/>
        <xsd:attribute name="copyKey" type="xsd:boolean" use="optional"/>
        <xsd:attribute name="timeToLive" type="xsd:int" use="optional"/>
        <xsd:attribute name="ttlEvictorType" type="dgc:ttlEvictorType" use="optional"/>
        <xsd:attribute name="writeBehind" type="xsd:string" use="optional"/>
        <xsd:attribute name="evictionTriggers" type="xsd:string" use="optional"/>
        <xsd:attribute name="template" type="xsd:boolean" use="optional"/>
    </xsd:complexType>


    <xsd:complexType name="bean">
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="0" name="property" type="dgc:property"/>
        </xsd:sequence>
        <xsd:attribute name="className" type="xsd:string" use="optional"/>
        <xsd:attribute name="id" type="dgc:beanId" use="required"/>
        <xsd:attribute name="osgiService" type="xsd:string" use="optional"/>
    </xsd:complexType>

    <xsd:complexType name="backingMapPluginCollection">
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="0" name="bean" type="dgc:bean"/>
        </xsd:sequence>
        <xsd:attribute name="id" type="xsd:string" use="required"/>
    </xsd:complexType>

    <xsd:complexType name="property">
        <xsd:attribute name="name" type="xsd:string" use="required"/>
        <xsd:attribute name="value" type="xsd:string" use="required"/>
        <xsd:attribute name="type" type="dgc:propertyType" use="required"/>
        <xsd:attribute name="description" type="xsd:string" use="optional"/>
    </xsd:complexType>

    <xsd:simpleType name="propertyType">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="java.lang.Boolean"/>
            <xsd:enumeration value="boolean"/>
            <xsd:enumeration value="java.lang.String"/>
            <xsd:enumeration value="java.lang.Integer"/>
            <xsd:enumeration value="int"/>
            <xsd:enumeration value="java.lang.Double"/>
            <xsd:enumeration value="double"/>
            <xsd:enumeration value="java.lang.Byte"/>
            <xsd:enumeration value="byte"/>
            <xsd:enumeration value="java.lang.Short"/>
            <xsd:enumeration value="short"/>
            <xsd:enumeration value="java.lang.Long"/>
            <xsd:enumeration value="long"/>
            <xsd:enumeration value="java.lang.Float"/>
            <xsd:enumeration value="float"/>
            <xsd:enumeration value="java.lang.Character"/>
            <xsd:enumeration value="char"/>
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="beanId">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="TransactionCallback"/>
            <xsd:enumeration value="ObjectGridEventListener"/>
            <xsd:enumeration value="ObjectGridLifecycleListener"/>
            <xsd:enumeration value="SubjectSource"/>
```

```
            <xsd:enumeration value="SubjectValidation"/>
            <xsd:enumeration value="ObjectGridAuthorization"/>

            <xsd:enumeration value="Loader"/>
            <xsd:enumeration value="ObjectTransformer"/>
            <xsd:enumeration value="OptimisticCallback"/>
            <xsd:enumeration value="Evictor"/>
            <xsd:enumeration value="MapEventListener"/>
            <xsd:enumeration value="BackingMapLifecycleListener"/>
            <xsd:enumeration value="MapIndexPlugin"/>
            <xsd:enumeration value="CollisionArbiter"/>
            <xsd:enumeration value="MapSerializerPlugin"/>
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="copyMode">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="COPY_ON_READ_AND_COMMIT"/>
            <xsd:enumeration value="COPY_ON_READ"/>
            <xsd:enumeration value="COPY_ON_WRITE"/>
            <xsd:enumeration value="NO_COPY"/>
            <xsd:enumeration value="COPY_TO_BYTES"/>
            <xsd:enumeration value="COPY_TO_BYTES_RAW"/>
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="lockStrategy">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="OPTIMISTIC"/>
            <xsd:enumeration value="PESSIMISTIC"/>
            <xsd:enumeration value="NONE"/>
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="ttlEvictorType">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="CREATION_TIME"/>
            <xsd:enumeration value="LAST_ACCESS_TIME"/>
            <xsd:enumeration value="LAST_UPDATE_TIME"/>
            <xsd:enumeration value="NONE"/>
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="authorizationMechanism">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="AUTHORIZATION_MECHANISM_JAAS"/>
            <xsd:enumeration value="AUTHORIZATION_MECHANISM_CUSTOM"/>
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="accessByCreatorOnlyMode">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="disabled"/>
            <xsd:enumeration value="complement"/>
            <xsd:enumeration value="supersede"/>
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:complexType name="timeBasedDBUpdate">
        <xsd:attribute name="persistenceUnitName" type="xsd:string" use="optional"/>
        <xsd:attribute name="mode" type="cc:dbUpdateMode" use="optional"/>
        <xsd:attribute name="timestampField" type="xsd:string" use="optional"/>
        <xsd:attribute name="entityClass" type="xsd:string" use="required"/>
        <xsd:attribute name="jpaPropertyFactory" type="xsd:string" use="optional"/>
    </xsd:complexType>

   <xsd:simpleType name="dbUpdateMode">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="INVALIDATE_ONLY"/>
            <xsd:enumeration value="UPDATE_ONLY"/>
            <xsd:enumeration value="INSERT_UPDATE"/>
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:complexType name="querySchema">
        <xsd:sequence>
            <xsd:element maxOccurs="1" minOccurs="1" name="mapSchemas" type="dgc:mapSchemas">
                <xsd:unique name="mapNameUnique">
                      <xsd:selector xpath="dgc:mapSchema"/>
                      <xsd:field xpath="@mapName"/>
                </xsd:unique>
            </xsd:element>
            <xsd:element maxOccurs="1" minOccurs="0" name="relationships" type="dgc:relationships"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="mapSchemas">
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" name="mapSchema" type="dgc:mapSchema"/>
        </xsd:sequence>
    </xsd:complexType>
```

```
    <xsd:complexType name="relationships">
        <xsd:sequence>
            <xsd:element maxOccurs="unbounded" minOccurs="1" name="relationship" type="dgc:relationship"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="mapSchema">
        <xsd:attribute name="mapName" type="xsd:string" use="required"/>
        <xsd:attribute name="valueClass" type="xsd:string" use="required"/>
        <xsd:attribute name="primaryKeyField" type="xsd:string" use="optional"/>
        <xsd:attribute name="accessType" type="cc:accessType" use="optional"/>
    </xsd:complexType>

    <xsd:complexType name="relationship">
        <xsd:attribute name="source" type="xsd:string" use="required"/>
        <xsd:attribute name="target" type="xsd:string" use="required"/>
        <xsd:attribute name="relationField" type="xsd:string" use="required"/>
        <xsd:attribute name="invRelationField" type="xsd:string" use="optional"/>
    </xsd:complexType>

    <xsd:simpleType name="accessType">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="PROPERTY"/>
            <xsd:enumeration value="FIELD"/>
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="initialState">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="OFFLINE"/>
            <xsd:enumeration value="PRELOAD"/>
            <xsd:enumeration value="ONLINE"/>
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="transactionIsolation">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="READ_UNCOMMITTED"/>
            <xsd:enumeration value="READ_COMMITTED"/>
            <xsd:enumeration value="REPEATABLE_READ"/>
        </xsd:restriction>
    </xsd:simpleType>

</xsd:schema>
```

Deployment policy descriptor XML file

To configure a deployment policy, use a deployment policy descriptor XML file.

In the following sections, the elements and attributes of the deployment policy descriptor XML file are defined. See the deploymentPolicy.xsd file for the corresponding deployment policy XML schema.

Figure 1. Elements in the deploymentPolicy.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy ../deploymentPolicy.xsd"
        xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">
        <objectgridDeployment objectgridName="myGrid">
                <mapSet
                                name="mapSetName"
                                numberOfPartitions="numberOfPartitions"
                                minSyncReplicas="minimumNumber"
                                maxSyncReplicas="maximumNumber"
                                maxAsyncReplicas="maximumNumber"
                                replicaReadEnabled="true|false"
                                numInitialContainers="numberOfInitialContainersBeforePlacement"
                                autoReplaceLostShards="true|false"
                                developmentMode="true|false"
                                placementStrategy="FIXED_PARTITIONS|PER_CONTAINER">
                                <map ref="backingMapReference" />

                                        <zoneMetadata>
                                                <shardMapping
                                                        shard="shardType"
                                                        zoneRuleRef="zoneRuleRefName" />
                                                <zoneRule
                                                                name="zoneRuleName"
                                                                exclusivePlacement="true|false" >
                                                                <zone name="ALPHA" />
                                                                <zone name="BETA" />
                                                                <zone name="GAMMA" />
                                                </zoneRule>
                                        </zoneMetadata>
                        </mapSet>
```

```
            </objectgridDeployment>
        </deploymentPolicy>
```

deploymentPolicy element

The deploymentPolicy element is the top-level element of the deployment policy XML file. This element sets up the namespace of the file and the schema location. The schema is defined in the deploymentPolicy.xsd file.

- **Number of occurrences:** One
- **Child element:** objectgridDeployment

objectgridDeployment element

The objectgridDeployment element is used to reference an ObjectGrid instance from the ObjectGrid XML file. Within the objectgridDeployment element, you can divide your maps into map sets.

- **Number of occurrences:** One or more
- **Child element:** mapSet

Attributes

objectgridName
: Specifies the name of the ObjectGrid instance to deploy. This attribute references an objectGrid element that is defined in the ObjectGrid XML file. (Required)

For example, the objectgridName attribute is set as `CompanyGrid` in the companyGridDpReplication.xml file. The objectgridName attribute references the `CompanyGrid` that is defined in the companyGrid.xml file. Read about the ObjectGrid descriptor XML file, which you must couple with the deployment policy file for each ObjectGrid instance.

mapSet element

The mapSet element is used to group maps together. The maps within a mapSet element are partitioned and replicated similarly. Each map must belong to only one mapSet element.

- **Number of occurrences:** One or more
- **Child elements:**
  - map
  - zoneMetadata

Attributes

name
: (Required) Specifies the name of the mapSet. This attribute must be unique within the objectgridDeployment element.

numberOfPartitions
: (Optional) Specifies the number of partitions for the mapSet element. The default value is 1. The number must be appropriate for the number of container servers that host the partitions. (Optional)

minSyncReplicas
: Specifies the minimum number of synchronous replicas for each partition in the mapSet. The default value is 0. 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.
  In the following configurations, when the minSyncReplicas value is set to a value greater than `0`, transactions are rejected from the data grid because a replica is expected:

  - Only one zone is available in a multiple zone configuration
  - Only one host is available and the developmentMode attribute is set to `false`.
  - If the allowableShardOverrage property is configured, transactions for a particular partition are rejected until the second zone has a number of container servers over the configured percentage.

maxSyncReplicas
: Specifies the maximum number of synchronous replicas for each partition in the mapSet. The default value is 0. 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 is not already met. (Optional)

maxAsyncReplicas
: Specifies the maximum number of asynchronous replicas for each partition in the mapSet. The default value is 0. After the primary and all synchronous replicas are placed for a partition, asynchronous replicas are placed until the maxAsyncReplicas value is met. (Optional)

replicaReadEnabled
: If this attribute is set to `true`, read requests are distributed between a partition primary and its replicas. If the replicaReadEnabled attribute is `false`, read requests are routed to the primary only. The default value is false. (Optional)

numInitialContainers
: Specifies the number of container servers that are required before initial placement occurs for the shards in this mapSet element. The default value is `1`. This attribute can help save process and network bandwidth when you are bringing a data grid online from a cold start. (Optional)
  You can also use the placementDeferralInterval property and the **xscmd -c suspendBalancing** command to delay the initial placement of shards on the container servers.
  Starting a container server sends an event to the catalog service. The first time that the number of active container servers is equal to the numInitialContainers value for a mapSet element, the catalog service places the shards from the mapSet, if the minSyncReplicas value can also be

satisfied. After the numInitialContainers value is met, each container server-started event can trigger a rebalancing of unplaced and previously placed shards. If you know approximately how many container servers you are going to start for this mapSet element, you can set the numInitialContainers value close to that number to avoid the rebalancing after every container server start. Placement occurs only when you reach the numInitialContainers value that is specified in the mapSet element.
To override the numInitialContainers value, for example, when you are performing maintenance on your servers and want shard placement to continue running, you can use the **xscmd -c triggerPlacement** command. This override is temporary and is applied when you run the command. After you run the command, all subsequent placement runs use the numInitialContainers value.

autoReplaceLostShards
: Specifies if lost shards are placed on other container servers. The default value is `true`. When a container server is stopped or fails, the shards that are running on the container server are lost. A lost primary shard causes one of its replica shards to be promoted to the primary shard for the corresponding partition. Because of this promotion, one of the replicas is lost. If you want lost shards to remain unplaced, set the autoReplaceLostShards attribute to `false`. This setting does not affect the promotion chain, but only the replacement of the last shard in the chain. (Optional)

developmentMode
: With this attribute, you can influence where a shard is placed in relation to its peer shards. The default value is `true`. 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 server. In either case, no two shards from the same partition are ever placed in the same container server. (Optional)

placementStrategy
: There are two placement strategies. The default strategy is to use a fixed partition strategy. By setting the attribute placementStrategy to `FIXED_PARTITIONS`, the number of primary shards that are placed across available container servers is equal to the number of partitions that are defined, and increased by the number of replicas. The other strategy is to use a per container strategy. By setting placementStrategy to `PER_CONTAINER`, the number of primary shards that are placed on each container server is equal to the number of partitions that are defined, with an equal number of replicas that are placed on other container servers. (Optional)

map element

Each map in a mapSet element references one of the backingMap elements that is defined in the corresponding ObjectGrid XML file. Every map in a distributed eXtreme Scale environment can belong to only one mapSet element.

- **Number of occurrences:** One or more
- **Child element:** None

Attributes

ref
: Provides a reference to a backingMap element in the ObjectGrid XML file. Each map in a mapSet element must reference a backingMap element from the ObjectGrid XML file. The value that is assigned to the ref attribute must match the name attribute of one of the backingMap elements in the ObjectGrid XML file. (Required)

zoneMetadata element

You can place shards into zones. With zones, you can control how eXtreme Scale places shards on a grid. Java virtual machines that host an eXtreme Scale server can be tagged with a zone identifier. The deployment file can include one or more zone rules, and these zone rules are associated with a shard type. The zoneMetadata element is a receptacle of zone configuration elements. Within the zoneMetadata element, zones can be defined and shard placement behavior can be influenced.

For more information, see Zones for replica placement.

- **Number of occurrences:** Zero or one
- **Child elements:**
  - shardMapping
  - zoneRule

Attributes: None

shardMapping element

The shardMapping element is used to associate a shard type with a zone rule. Placement of the shard is influenced by the mapping to the zone rule.

- **Number of occurrences:** Zero or one
- **Child elements:** None

Attributes

shard
: Specify the name of a shard with which to associate the zoneRule. (Required)

zoneRuleRef
: Specify the name of a zoneRule with which to associate the shard. (Optional)

zoneRule element

A zone rule specifies the possible set of zones in which a shard can be placed. The zoneRule element is used to specify a set of zones that a set of shard types can be placed within. The zone rule can also be used to determine how shards are grouped across the zones with the exclusivePlacement attribute.

- **Number of occurrences:** One or more

- **Child elements:** zone

Attributes

name
: Specify the name of the zone rule that you defined previously, as the zoneRuleRef in a shardMapping element. (Required)

exclusivePlacement
: An exclusive setting indicates that each shard type mapped to this zone rule is placed in a different zone in the zone list. An inclusive setting indicates that after a shard is placed in a zone from the list, then the other shard types that are mapped to this zone rule are also placed in that zone. At least 3 zones are required when you use an exclusive setting with 3 shards that are mapped to the same zone rule. The 3 shards include the primary, and 2 synchronous replicas. (Optional)

zone element

The zone element is used to name a zone within a zone rule. Each zone that is named must correspond to a zone name that is used to start servers.

Example

In the following example, the mapSet element is used to configure a deployment policy. The value is set to `mapSet1`, and is divided into 10 partitions. Each of these partitions must have at least one synchronous replica available and no more than two synchronous replicas. Each partition also has an asynchronous replica if the environment can support it. All synchronous replicas are placed before any asynchronous replicas are placed. Additionally, the catalog service does not attempt to place the shards for the mapSet1 element until the domain can support the minSyncReplicas value. Supporting the minSyncReplicas value requires two or more container servers: one for the primary and two for the synchronous replica.

```
<?xml version="1.0" encoding="UTF-8"?>
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy ../deploymentPolicy.xsd"
        xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">

        <objectgridDeployment objectgridName="CompanyGrid">
                <mapSet name="mapSet1" numberOfPartitions="10"
                        minSyncReplicas="1" maxSyncReplicas="2" maxAsyncReplicas="1"
                        numInitialContainers="10" autoReplaceLostShards="true"
                        developmentMode="false" replicaReadEnabled="true">
                        <map ref="Customer"/>
                        <map ref="Item"/>
                        <map ref="OrderLine"/>
                        <map ref="Order"/>
                </mapSet>
        </objectgridDeployment>

</deploymentPolicy>
```

Only 2 container servers are required to satisfy the replication settings. However, the numInitialContainers attribute requires 10 available container servers before the catalog service attempts to place any of the shards in this mapSet element. After the domain has 10 container servers that are able to support the CompanyGrid ObjectGrid, all shards in the mapSet1 element are placed.
When the autoReplaceLostShards attribute is set to true, any shard in this mapSet element that is lost as the result of container server failure is automatically replaced on another container server. This replacement occurs only if a container server is available to host the lost shard. Shards from the same partition cannot be placed on the same server for the mapSet1 element because the developmentMode attribute is set to `false`. Read-only requests are distributed across the primary shard and its replicas for each partition because the replicaReadEnabled value is `true`.

The companyGridDpMapSetAttr.xml file uses the ref attribute on the map to reference each of the backingMap elements from the companyGrid.xml file.

For more examples, see Zone-preferred routing.

- deploymentPolicy.xsd file
  Use the deployment policy XML schema to create a deployment descriptor XML file.

Related tasks:
Controlling placement
Administering with the xscmd utility
Starting stand-alone servers
Enabling data grid authorization
Troubleshooting XML configuration
Related reference:
Server properties file
Client properties file
REST data service properties file
ObjectGrid descriptor XML file
Entity metadata descriptor XML file
Security descriptor XML file
Spring descriptor XML file
Liberty profile configuration files
Related information:
API documentation
Getting started tutorial lesson 1.1: Defining data grids with configuration files
Interface PlacementServiceMBean

deploymentPolicy.xsd file

Use the deployment policy XML schema to create a deployment descriptor XML file.

See the Deployment policy descriptor XML file for descriptions of the elements and attributes defined in the deploymentPolicy.xsd file.

8.5+

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:dp="http://ibm.com/ws/objectgrid/deploymentPolicy"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://ibm.com/ws/objectgrid/deploymentPolicy"
    elementFormDefault="qualified">

    <xsd:element name="deploymentPolicy">
        <xsd:complexType>
            <xsd:choice>
                <xsd:element name="objectgridDeployment"
                    type="dp:objectgridDeployment" minOccurs="1"
                    maxOccurs="unbounded">
                    <xsd:unique name="mapSetNameUnique">
                        <xsd:selector xpath="dp:mapSet" />
                        <xsd:field xpath="@name" />
                    </xsd:unique>
                </xsd:element>
            </xsd:choice>
        </xsd:complexType>
    </xsd:element>

    <xsd:complexType name="objectgridDeployment">
        <xsd:sequence>
            <xsd:element name="mapSet" type="dp:mapSet"
                maxOccurs="unbounded" minOccurs="1">
                <xsd:unique name="mapNameUnique">
                    <xsd:selector xpath="dp:map" />
                    <xsd:field xpath="@ref" />
                </xsd:unique>
            </xsd:element>
        </xsd:sequence>
        <xsd:attribute name="objectgridName" type="xsd:string"
            use="required" />
    </xsd:complexType>


    <xsd:complexType name="mapSet">
        <xsd:sequence>
            <xsd:element name="map" type="dp:map" maxOccurs="unbounded"
                minOccurs="1" />
            <xsd:element name="zoneMetadata" type="dp:zoneMetadata"
                maxOccurs="1" minOccurs="0">

                <xsd:key name="zoneRuleName">
                    <xsd:selector xpath="dp:zoneRule" />
                    <xsd:field xpath="@name" />
                </xsd:key>

                <xsd:keyref name="zoneRuleRef"
                    refer="dp:zoneRuleName">
                    <xsd:selector xpath="dp:shardMapping" />
                    <xsd:field xpath="@zoneRuleRef" />
                </xsd:keyref>

            </xsd:element>
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string" use="required" />
        <xsd:attribute name="numberOfPartitions" type="xsd:int"
            use="optional" />
        <xsd:attribute name="minSyncReplicas" type="xsd:int"
            use="optional" />
        <xsd:attribute name="maxSyncReplicas" type="xsd:int"
            use="optional" />
        <xsd:attribute name="maxAsyncReplicas" type="xsd:int"
            use="optional" />
        <xsd:attribute name="replicaReadEnabled" type="xsd:boolean"
            use="optional" />
        <xsd:attribute name="numInitialContainers" type="xsd:int"
            use="optional" />
        <xsd:attribute name="autoReplaceLostShards" type="xsd:boolean"
            use="optional" />
        <xsd:attribute name="developmentMode" type="xsd:boolean"
            use="optional" />
        <xsd:attribute name="placementStrategy"
            type="dp:placementStrategy" use="optional" />
        <xsd:attribute name="placementScope"
            type="dp:placementScope" default="DOMAIN_SCOPE" use="optional" />
        <xsd:attribute name="placementScopeTopology"
            type="dp:placementScopeTopology" default="RING" use="optional" />
```

```
    </xsd:complexType>

    <xsd:simpleType name="placementStrategy">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="FIXED_PARTITIONS" />
            <xsd:enumeration value="PER_CONTAINER" />
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="placementScope">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="DOMAIN_SCOPE" />
            <xsd:enumeration value="CONTAINER_SCOPE" />
<!--            <xsd:enumeration value="ZONE_SCOPE" /> -->
        </xsd:restriction>
    </xsd:simpleType>

    <xsd:simpleType name="placementScopeTopology">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="RING" />
            <xsd:enumeration value="HUB" />
        </xsd:restriction>
    </xsd:simpleType>


    <xsd:complexType name="map">
        <xsd:attribute name="ref" use="required" />
    </xsd:complexType>

    <xsd:complexType name="zoneMetadata">
        <xsd:sequence>
            <xsd:element name="shardMapping" type="dp:shardMapping"
                maxOccurs="unbounded" minOccurs="1" />
            <xsd:element name="zoneRule" type="dp:zoneRule"
                maxOccurs="unbounded" minOccurs="1">

            </xsd:element>

        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="shardMapping">
        <xsd:attribute name="shard" use="required">
                <xsd:simpleType>
                        <xsd:restriction base="xsd:string">
                                <xsd:enumeration value="P"></xsd:enumeration>
                                <xsd:enumeration value="S"></xsd:enumeration>
                                <xsd:enumeration value="A"></xsd:enumeration>
                        </xsd:restriction>
                </xsd:simpleType>
        </xsd:attribute>
        <xsd:attribute name="zoneRuleRef" type="xsd:string"
            use="required" />
    </xsd:complexType>

    <xsd:complexType name="zoneRule">
        <xsd:sequence>
            <xsd:element name="zone" type="dp:zone"
                maxOccurs="unbounded" minOccurs="1" />
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string" use="required" />
        <xsd:attribute name="exclusivePlacement" type="xsd:boolean"
            use="optional" />
    </xsd:complexType>

    <xsd:complexType name="zone">
        <xsd:attribute name="name" type="xsd:string" use="required" />
    </xsd:complexType>

</xsd:schema>
```

Entity metadata descriptor XML file

The entity metadata descriptor file is an XML file that is used to define an entity schema for WebSphere® eXtreme Scale. Define all of the entity metadata in the XML file, or define the entity metadata as annotations on the entity Java™ class file. The primary use is for entities that cannot use Java annotations.

Use XML configuration to create entity metadata that is based on the XML file. When used in conjunction with annotation, some of the attributes that are defined in the XML configuration override the corresponding annotations. If you can override an element, the override is explicitly in the following sections. See emd.xsd file for an example of the entity metadata descriptor XML file.

id element

The id element implies that the attribute is a key. At a minimum, at least one id element must be specified. You can specify multiple id keys for use as a compound key.

Attributes

name
: Specifies the name of the attribute. The attribute must exist in the Java file.

alias
: Specifies the element alias. The alias value is overridden if used in conjunction with an annotated entity.

basic element

The basic element implies that the attribute is a primitive type or wrappers to primitive types:

- java.lang.String
- java.math.BigInteger
- java.math.BigDecimal
- java.util.Date
- java.util.Calendar
- java.sql.Date
- java.sql.Time
- java.sql.Timestamp
- byte[]
- Byte[]
- char[]
- Character[]
- Java Platform, Standard Edition Version 5 enum

It is not necessary to specify any attribute as basic. The basic element attributes are automatically configured using reflection.

Attributes

name
: Specifies the name of the attribute in the class.

alias
: Specifies the element alias. The alias value is overridden if used in conjunction with an annotated entity.

fetch
: Specifies the fetch type. Valid values include: `LAZY` or `EAGER`.

id-class element

The id_class element specifies a compound key class, which helps to find entities with compound keys.

Attributes

class-name
: Specifies the class name, which is an id-class, to use with the id-class element.

transient element

The transient element implies that it is ignored and not processed. It also can be overridden if used in conjunction with annotated entities.

Attributes

name
: Specifies the name of the attribute, which is ignored.

version element

Attributes

name
: Specifies the name of the attribute, which is ignored.

cascade-type element

Child elements

- **cascade-all**: Cascades the all operation to associations.
- **cascade-persist**: Cascades the persist operation to associations.
- **cascade-remove**: Cascades the remove operation to associations.
- **cascade-merge**: Currently not used.
- **cascade-refresh**: Currently not used.

one-to-one element

Attributes

name
: Specifies the name of the class, which has a one-to-one relationship.

alias
: Specifies a name alias.

target-entity
: Specifies the association class. This value is a fully-qualified class name.

fetch
: Specifies the fetch type. Valid values include: `LAZY` or `EAGER`.

mapped-by
: Specifies the field that owns the relationship. The mapped-by element is only specified on the inverse (non-owning) side of the association.

id
: Identifies the association as key.

Child elements

- **cascade**: cascade-type element

one-to-many element

Attributes

name
: Specifies the name of the attribute in the class.

alias
: Specifies a name alias.

target-entity
: Specifies the association class. This value is a fully-qualified class name.

fetch
: Specifies the fetch type. Valid values include: `LAZY` or `EAGER`.

mapped-by
: Specifies the field that owns the relationship. The mapped-by element is only specified on the inverse (non-owning) side of the association.

Child elements

- **order-by**
- **cascade**: cascade-type element

many-to-one element

Attributes

name
: Specifies the name of the attribute in the class.

alias
: Specifies a name alias.

target-entity
: Specifies the class to which this attribute refers. This value is a fully-qualified class name.

fetch
: Specifies the fetch type. Valid values include: `LAZY` or `EAGER`.

id
: Identifies the association as a key.

Child elements

- **cascade**: cascade-type element

many-to-many element

Attributes

name
: Specifies the name of the attribute in the class.

alias
: Specifies a name alias.

target-entity
: Specifies the class to which this attribute refers. This value is a fully-qualified class name.

fetch
: Specifies the fetch type. Valid values include: `LAZY` or `EAGER`.

mapped-by
: Specifies the field that owns the relationship. The mapped-by element is only specified on the inverse (non-owning) side of the association.

Child elements

- **order-by**
- **cascade**: cascade-type element

attributes element

Child elements

- id element
- basic element
- version element
- many-to-one element
- one-to-many element
- one-to-one element
- many-to-many element
- transient element

Entity element

Attributes

name(required)
: Specifies the name of the attribute in the class.

class-name
: Specifies the fully-qualified class name.

access
: Specifies the access type. The valid values are `PROPERTY` or `FIELD`.

schemaRoot
: Specifies that this entity is the schema root and is used as a parent class for partitioned data.

Child elements

- **description:** Specifies a description.
- id-class element
- attributes element

entity-mappings element

Child elements

- **description:** Specifies a description.
- Entity element

entity-listener element

Attributes

class-name (required)
: Specifies the name of the listener class.

Child elements

- PrePersist element
- PostPersist element
- PreRemove element
- PreUpdate element
- PostUpdate element
- PostLoad element

PrePersist element

Attributes

method-name (required)
: Specifies the lifecycle callback method for the PrePersist event.

PostPersist element

Attributes

method-name (required)
: Specifies the lifecycle callback method for the PostPersist event.

PreRemove element

Attributes

method-name (required)
: Specifies the lifecycle callback method for the PreRemove event.

PreUpdate element

Attributes

method-name (required)
Specifies the lifecycle callback method for the PreUpdate event.

PostUpdate element

Attributes

method-name (required)
Specifies the lifecycle callback method for the PostUpdate event.

PostLoad element

Attributes

method-name (required)
Specifies the lifecycle callback method for the PostLoad event.

- emd.xsd file
  Use the entity metadata XML schema definition to create a descriptor XML file and define an entity schema for WebSphere eXtreme Scale.

Related tasks:
Troubleshooting XML configuration
Related reference:
Server properties file
Client properties file
REST data service properties file
ObjectGrid descriptor XML file
Deployment policy descriptor XML file
Security descriptor XML file
Spring descriptor XML file
Liberty profile configuration files

emd.xsd file

Use the entity metadata XML schema definition to create a descriptor XML file and define an entity schema for WebSphere® eXtreme Scale.

See the Entity metadata descriptor XML file for the descriptions of each element and attribute of the emd.xsd file.

emd.xsd file

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:emd="http://ibm.com/ws/projector/config/emd"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://ibm.com/ws/projector/config/emd"
    elementFormDefault="qualified" attributeFormDefault="unqualified"
    version="1.0">

    <!-- ************************************************* -->
    <xsd:element name="entity-mappings">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="description" type="xsd:string" minOccurs="0" />
                <xsd:element name="entity" type="emd:entity" minOccurs="1" maxOccurs="unbounded" />
            </xsd:sequence>
        </xsd:complexType>
        <xsd:unique name="uniqueEntityClassName">
            <xsd:selector xpath="emd:entity" />
            <xsd:field xpath="@class-name" />
        </xsd:unique>
    </xsd:element>

    <!-- ************************************************* -->
    <xsd:complexType name="entity">
        <xsd:sequence>
            <xsd:element name="description" type="xsd:string" minOccurs="0" />
            <xsd:element name="id-class" type="emd:id-class" minOccurs="0" />
            <xsd:element name="attributes" type="emd:attributes" minOccurs="0" />
            <xsd:element name="entity-listeners" type="emd:entity-listeners" minOccurs="0" />
            <xsd:element name="pre-persist" type="emd:pre-persist" minOccurs="0" />
            <xsd:element name="post-persist" type="emd:post-persist" minOccurs="0" />
            <xsd:element name="pre-remove" type="emd:pre-remove" minOccurs="0" />
            <xsd:element name="post-remove" type="emd:post-remove" minOccurs="0" />
            <xsd:element name="pre-invalidate" type="emd:pre-invalidate" minOccurs="0" />
```

```
            <xsd:element name="post-invalidate" type="emd:post-invalidate" minOccurs="0" />
            <xsd:element name="pre-update" type="emd:pre-update" minOccurs="0" />
            <xsd:element name="post-update" type="emd:post-update" minOccurs="0" />
            <xsd:element name="post-load" type="emd:post-load" minOccurs="0" />
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string" use="required" />
        <xsd:attribute name="class-name" type="xsd:string" use="required" />
        <xsd:attribute name="access" type="emd:access-type" />
        <xsd:attribute name="schemaRoot" type="xsd:boolean" />
    </xsd:complexType>

    <!-- ************************************************* -->
    <xsd:complexType name="attributes">
        <xsd:sequence>
            <xsd:choice>
                <xsd:element name="id" type="emd:id" minOccurs="0" maxOccurs="unbounded" />
            </xsd:choice>
            <xsd:element name="basic" type="emd:basic" minOccurs="0" maxOccurs="unbounded" />
            <xsd:element name="version" type="emd:version" minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element name="many-to-one" type="emd:many-to-one" minOccurs="0" maxOccurs="unbounded" />
            <xsd:element name="one-to-many" type="emd:one-to-many" minOccurs="0" maxOccurs="unbounded" />
            <xsd:element name="one-to-one" type="emd:one-to-one" minOccurs="0" maxOccurs="unbounded" />
            <xsd:element name="many-to-many" type="emd:many-to-many" minOccurs="0" maxOccurs="unbounded" />
            <xsd:element name="transient" type="emd:transient" minOccurs="0" maxOccurs="unbounded" />
        </xsd:sequence>
    </xsd:complexType>

    <!-- ************************************************* -->
    <xsd:simpleType name="access-type">
        <xsd:restriction base="xsd:token">
            <xsd:enumeration value="PROPERTY" />
            <xsd:enumeration value="FIELD" />
        </xsd:restriction>
    </xsd:simpleType>

    <!-- ************************************************* -->
    <xsd:complexType name="id-class">
        <xsd:attribute name="class-name" type="xsd:string" use="required" />
    </xsd:complexType>

    <!-- ************************************************* -->
    <xsd:complexType name="id">
        <xsd:attribute name="name" type="xsd:string" use="required" />
        <xsd:attribute name="type" type="xsd:string" />
        <xsd:attribute name="alias" type="xsd:string" use="optional" />
    </xsd:complexType>

    <!-- ************************************************* -->
    <xsd:complexType name="transient">
        <xsd:attribute name="name" type="xsd:string" use="required" />
    </xsd:complexType>

    <!-- ************************************************* -->
    <xsd:complexType name="basic">
        <xsd:attribute name="name" type="xsd:string" use="required" />
        <xsd:attribute name="alias" type="xsd:string" />
        <xsd:attribute name="type" type="xsd:string" />
        <xsd:attribute name="fetch" type="emd:fetch-type" />
    </xsd:complexType>

    <!-- ************************************************* -->
    <xsd:simpleType name="fetch-type">
        <xsd:restriction base="xsd:token">
            <xsd:enumeration value="LAZY" />
            <xsd:enumeration value="EAGER" />
        </xsd:restriction>
    </xsd:simpleType>

    <!-- ************************************************* -->
    <xsd:complexType name="many-to-one">
        <xsd:sequence>
            <xsd:element name="cascade" type="emd:cascade-type" minOccurs="0" />
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string" use="required" />
        <xsd:attribute name="alias" type="xsd:string" />
        <xsd:attribute name="target-entity" type="xsd:string" />
        <xsd:attribute name="fetch" type="emd:fetch-type" />
        <xsd:attribute name="id" type="xsd:boolean" />
    </xsd:complexType>
    <!-- ************************************************* -->
    <xsd:complexType name="one-to-one">
        <xsd:sequence>
            <xsd:element name="cascade" type="emd:cascade-type" minOccurs="0" />
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string" use="required" />
        <xsd:attribute name="alias" type="xsd:string" />
        <xsd:attribute name="target-entity" type="xsd:string" />
        <xsd:attribute name="fetch" type="emd:fetch-type" />
        <xsd:attribute name="mapped-by" type="xsd:string" />
        <xsd:attribute name="id" type="xsd:boolean" />
    </xsd:complexType>
```

```
    <!-- ************************************************* -->
    <xsd:complexType name="one-to-many">
        <xsd:sequence>
            <xsd:element name="order-by" type="emd:order-by" minOccurs="0" />
            <xsd:element name="cascade" type="emd:cascade-type" minOccurs="0" />
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string" use="required" />
        <xsd:attribute name="alias" type="xsd:string" />
        <xsd:attribute name="target-entity" type="xsd:string" />
        <xsd:attribute name="fetch" type="emd:fetch-type" />
        <xsd:attribute name="mapped-by" type="xsd:string" />
    </xsd:complexType>

    <!-- ************************************************* -->
    <xsd:complexType name="many-to-many">
        <xsd:sequence>
            <xsd:element name="order-by" type="emd:order-by" minOccurs="0" />
            <xsd:element name="cascade" type="emd:cascade-type" minOccurs="0" />
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string" use="required" />
        <xsd:attribute name="alias" type="xsd:string" />
        <xsd:attribute name="target-entity" type="xsd:string" />
        <xsd:attribute name="fetch" type="emd:fetch-type" />
        <xsd:attribute name="mapped-by" type="xsd:string" />
    </xsd:complexType>

    <!-- ************************************************* -->
    <xsd:simpleType name="order-by">
        <xsd:restriction base="xsd:string" />
    </xsd:simpleType>

    <!-- ************************************************* -->
    <xsd:complexType name="cascade-type">
        <xsd:sequence>
            <xsd:element name="cascade-all" type="emd:emptyType" minOccurs="0" />
            <xsd:element name="cascade-persist" type="emd:emptyType" minOccurs="0" />
            <xsd:element name="cascade-remove" type="emd:emptyType" minOccurs="0" />
            <xsd:element name="cascade-invalidate" type="emd:emptyType" minOccurs="0" />
            <xsd:element name="cascade-merge" type="emd:emptyType" minOccurs="0" />
            <xsd:element name="cascade-refresh" type="emd:emptyType" minOccurs="0" />
        </xsd:sequence>
    </xsd:complexType>

    <!-- ************************************************* -->
    <xsd:complexType name="emptyType" />

    <!-- ************************************************* -->
    <xsd:complexType name="version">
          <xsd:attribute name="name" type="xsd:string" use="required"/>
          <xsd:attribute name="alias" type="xsd:string" />
          <xsd:attribute name="type" type="xsd:string" />
    </xsd:complexType>

    <!-- ************************************************* -->

    <xsd:complexType name="entity-listeners">
        <xsd:sequence>
            <xsd:element name="entity-listener" type="emd:entity-listener" minOccurs="0" maxOccurs="unbounded" />
        </xsd:sequence>
    </xsd:complexType>

    <!-- ************************************************* -->
    <xsd:complexType name="entity-listener">
        <xsd:sequence>
            <xsd:element name="pre-persist" type="emd:pre-persist" minOccurs="0" />
            <xsd:element name="post-persist" type="emd:post-persist" minOccurs="0" />
            <xsd:element name="pre-remove" type="emd:pre-remove" minOccurs="0" />
            <xsd:element name="post-remove" type="emd:post-remove" minOccurs="0" />
            <xsd:element name="pre-invalidate" type="emd:pre-invalidate" minOccurs="0" />
            <xsd:element name="post-invalidate" type="emd:post-invalidate" minOccurs="0" />
            <xsd:element name="pre-update" type="emd:pre-update" minOccurs="0" />
            <xsd:element name="post-update" type="emd:post-update" minOccurs="0" />
            <xsd:element name="post-load" type="emd:post-load" minOccurs="0" />
        </xsd:sequence>
        <xsd:attribute name="class-name" type="xsd:string" use="required" />
    </xsd:complexType>

    <!-- ************************************************* -->
    <xsd:complexType name="pre-persist">
        <xsd:attribute name="method-name" type="xsd:string" use="required" />
    </xsd:complexType>

    <!-- ************************************************* -->
    <xsd:complexType name="post-persist">
        <xsd:attribute name="method-name" type="xsd:string" use="required" />
    </xsd:complexType>

    <!-- ************************************************* -->
    <xsd:complexType name="pre-remove">
        <xsd:attribute name="method-name" type="xsd:string" use="required" />
    </xsd:complexType>
```

```
    <!-- ************************************************* -->
    <xsd:complexType name="post-remove">
        <xsd:attribute name="method-name" type="xsd:string" use="required" />
    </xsd:complexType>

    <!-- ************************************************* -->
    <xsd:complexType name="pre-invalidate">
        <xsd:attribute name="method-name" type="xsd:string" use="required" />
    </xsd:complexType>

    <!-- ************************************************* -->
    <xsd:complexType name="post-invalidate">
        <xsd:attribute name="method-name" type="xsd:string" use="required" />
    </xsd:complexType>

    <!-- ************************************************* -->
    <xsd:complexType name="pre-update">
        <xsd:attribute name="method-name" type="xsd:string" use="required" />
    </xsd:complexType>

    <!-- ************************************************* -->
    <xsd:complexType name="post-update">
        <xsd:attribute name="method-name" type="xsd:string" use="required" />
    </xsd:complexType>

    <!-- ************************************************* -->
    <xsd:complexType name="post-load">
        <xsd:attribute name="method-name" type="xsd:string" use="required" />
    </xsd:complexType>

</xsd:schema>
```

Security descriptor XML file

Use a security descriptor XML file to configure an eXtreme Scale deployment topology with security enabled. You can use the elements in this file to configure different aspects of security.

securityConfig element

The securityConfig element is the top-level element of the ObjectGrid security XML file. This element sets up the namespace of the file and the schema location. The schema is defined in the objectGridSecurity.xsd file.

- Number of occurrences: One
- Child elements: security

security element

Use the security element to define an ObjectGrid security.

- Number of occurrences: One
- Child elements: authenticator, adminAuthorization, and systemCredentialGenerator

Attributes

securityEnabled
: Enables security for the grid when set to true. The default value is false. If the value is set to false, grid-wide security is disabled. For more information, see Data grid security. (Optional)

singleSignOnEnabled
: Enables a client to connect to any server after it has authenticated with one of the servers when the value is set to `true`. Otherwise, a client must authenticate with each server before the client can connect. The default value is false. (Optional)

loginSessionExpirationTime
: Specifies the amount of time in seconds before the login session expires. If the login session expires, the client must authenticate again. (Optional)

adminAuthorizationEnabled
: Enables administrative authorization. If the value is set to true, all of the administrative tasks need authorization. The authorization mechanism that is used is specified by the value of the adminAuthorizationMechanism attribute. The default value is false. (Optional)

adminAuthorizationMechanism
: Indicates which authorization mechanism to use. WebSphere® eXtreme Scale supports two authorization mechanisms, Java™ Authentication and Authorization Service (JAAS) and custom authorization. The JAAS authorization mechanism uses the standard JAAS policy-based approach. To specify JAAS as the authorization mechanism, set the value to AUTHORIZATION_MECHANISM_JAAS. The custom authorization mechanism uses a user-plugged-in AdminAuthorization implementation. To specify a custom authorization mechanism, set the value to AUTHORIZATION_MECHANISM_CUSTOM. For more information on how these two mechanisms are used, see Authorizing application clients. (Optional)

The following security.xml file is a sample configuration to enable the data grid security.

security.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<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" singleSignOnEnabled="true"
        loginSessionExpirationTime="20"
        adminAuthorizationEnabled="true"
        adminAuthorizationMechanism="AUTHORIZATION_MECHANISM_JAAS" >

        <authenticator className ="com.ibm.websphere.objectgrid.security.plugins.
                                        builtins.WSTokenAuthenticator">
        </authenticator>

        <systemCredentialGenerator className ="com.ibm.websphere.objectgrid.security.
                                        plugins.builtins.WSTokenCredentialGenerator">
            <property name="properties" type="java.lang.String" value="runAs"
                                                        description="Using runAs subject" />
        </systemCredentialGenerator>

    </security>
</securityConfig>
```

authenticator element

Authenticates clients to eXtreme Scale servers in the data grid. The class that is specified by the className attribute must implement the com.ibm.websphere.objectgrid.security.plugins.Authenticator interface. The authenticator can use properties to call methods on the class that is specified by the className attribute. See property element for more information on using properties.
In the previous security.xml file example, the com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenAuthenticator class is specified as the authenticator. This class implements the com.ibm.websphere.objectgrid.security.plugins.Authenticator interface.

- Number of occurrences: zero or one
- Child element: property

Attributes

className
: Specifies a class that implements the com.ibm.websphere.objectgrid.security.plugins.Authenticator interface. Use this class to authenticate clients to the servers in the eXtreme Scale grid. (Required)

adminAuthorization element

Use the adminAuthorization element to set up administrative access to the data grid.

- Number of occurrences: zero or one
- Child element: property

Attributes

className
: Specifies a class that implements the com.ibm.websphere.objectgrid.security.plugins.AdminAuthorization interface. (Required)

systemCredentialGenerator element

Use a systemCredentialGenerator element to set up a system credential generator. This element only applies to a dynamic environment. In the dynamic configuration model, the dynamic container server connects to the catalog server as an eXtreme Scale client and the catalog server can connect to the eXtreme Scale container server as a client too. This system credential generator is used to represent a factory for the system credential.

- Number of occurrences: zero or one
- Child element: property

Attributes

className
: Specifies a class that implements the com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator interface. (Required)

See the previous security.xml file for an example of how to use a systemCredentialGenerator class. In this example, the system credential generator is a com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredentialGenerator class, which retrieves the RunAs Subject object from the thread.

property element

Calls set methods on the authenticator and adminAuthorization classes. The name of the property corresponds to a set method on the className attribute of the authenticator or adminAuthorization element.

- Number of occurrences: zero or more
- Child element: property

Attributes

name
: Specifies the name of the property. The value that is assigned to this attribute must correspond to a set method on the class that is provided as the className attribute on the containing bean. For example, if the className attribute of the bean is set to com.ibm.MyPlugin, and the name of the property

that is provided is size, then the com.ibm.MyPlugin class must have a setSize method. (Required)

type
: Specifies the type of the property. The type of the parameter is passed to the set method that is identified by the name attribute. The valid values are the Java primitives, their java.lang counterparts, and java.lang.String. The name and type attributes must correspond to a method signature on the className attribute of the bean. For example, if the name is size and the type is int, then a setSize(int) method must exist on the class that is specified as the className attribute for the bean. (Required)

value
: Specifies the value of the property. This value is converted to the type that is specified by the type attribute, and is then used as a parameter in the call to the set method that is identified by the name and type attributes. The value of this attribute is not validated in any way. The plug-in implementor must verify that the value passed in is valid. (Required)

description
: Provides a description of the property. (Optional)

See objectGridSecurity.xsd file for more information.

- objectGridSecurity.xsd file
  Use the following ObjectGrid security XML schema to enable security.

Related tasks:
Troubleshooting XML configuration
Related reference:
Server properties file
Client properties file
REST data service properties file
ObjectGrid descriptor XML file
Deployment policy descriptor XML file
Entity metadata descriptor XML file
Spring descriptor XML file
Liberty profile configuration files
Related information:
API documentation

objectGridSecurity.xsd file

Use the following ObjectGrid security XML schema to enable security.

See the Security descriptor XML file for descriptions of the elements and attributes defined in the objectGridSecurity.xsd file.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:cc="http://ibm.com/ws/objectgrid/config/security"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://ibm.com/ws/objectgrid/config/security"
        elementFormDefault="qualified">

        <xsd:element name="securityConfig">
                <xsd:complexType>
                        <xsd:sequence>
                                <xsd:element name="security" type="cc:security" />
                        </xsd:sequence>
                </xsd:complexType>
        </xsd:element>

        <xsd:complexType name="security">
                <xsd:sequence>
                        <xsd:element name="authenticator" type="cc:bean" minOccurs="0"
                                maxOccurs="1" />
                        <xsd:element name="adminAuthorization" type="cc:bean" minOccurs="0"
                                maxOccurs="1" />
                        <xsd:element name="systemCredentialGenerator" type="cc:bean" minOccurs="0"
                                maxOccurs="1" />
                </xsd:sequence>
                <xsd:attribute name="securityEnabled" type="xsd:boolean" use="optional" />
                <xsd:attribute name="singleSignOnEnabled" type="xsd:boolean" use="optional"/>
                <xsd:attribute name="loginSessionExpirationTime" type="xsd:int" use="optional"/>
                <xsd:attribute name="adminAuthorizationMechanism" type="cc:adminAuthorizationMechanism"
                        use="optional"/>
                <xsd:attribute name="adminAuthorizationEnabled" type="xsd:boolean" use="optional" />
        </xsd:complexType>

        <xsd:complexType name="bean">
                <xsd:sequence>
                        <xsd:element name="property" type="cc:property" maxOccurs="unbounded" minOccurs="0" />
                </xsd:sequence>
                <xsd:attribute name="className" type="xsd:string" use="required" />
        </xsd:complexType>

        <xsd:complexType name="property">
                <xsd:attribute name="name" type="xsd:string" use="required" />
                <xsd:attribute name="value" type="xsd:string" use="required" />
                <xsd:attribute name="type" type="cc:propertyType" use="required" />
                <xsd:attribute name="description" type="xsd:string" use="optional" />
```

```
        </xsd:complexType>

        <xsd:simpleType name="propertyType">
                <xsd:restriction base="xsd:string">
                        <xsd:enumeration value="java.lang.Boolean" />
                        <xsd:enumeration value="boolean" />
                        <xsd:enumeration value="java.lang.String" />
                        <xsd:enumeration value="java.lang.Integer" />
                        <xsd:enumeration value="int" />
                        <xsd:enumeration value="java.lang.Double" />
                        <xsd:enumeration value="double" />
                        <xsd:enumeration value="java.lang.Byte" />
                        <xsd:enumeration value="byte" />
                        <xsd:enumeration value="java.lang.Short" />
                        <xsd:enumeration value="short" />
                        <xsd:enumeration value="java.lang.Long" />
                        <xsd:enumeration value="long" />
                        <xsd:enumeration value="java.lang.Float" />
                        <xsd:enumeration value="float" />
                        <xsd:enumeration value="java.lang.Character" />
                        <xsd:enumeration value="char" />
                </xsd:restriction>
        </xsd:simpleType>

        <xsd:simpleType name="adminAuthorizationMechanism">
                <xsd:restriction base="xsd:string">
                        <xsd:enumeration value="AUTHORIZATION_MECHANISM_JAAS" />
                        <xsd:enumeration value="AUTHORIZATION_MECHANISM_CUSTOM" />
                </xsd:restriction>
        </xsd:simpleType>

</xsd:schema>
```

Spring descriptor XML file

Use a Spring descriptor XML file to configure and integrate eXtreme Scale with Spring.

In the following sections, each element and attribute of the Spring objectgrid.xsd file is defined. The Spring objectgrid.xsd file is in the ogspring.jar file and the ObjectGrid namespace com/ibm/ws/objectgrid/spring/namespace. See the Spring objectgrid.xsd file for an example of the descriptor XML schema.

register element

Use the register element to register the default bean factory for the ObjectGrid.

- Number of occurrences: Zero to many
- Child element: None

Attributes

id
: Specifies the name of the default bean directory for a particular ObjectGrid.

gridname
: Specifies the name of the ObjectGrid instance. The value assigned to this attribute must correspond to a valid ObjectGrid configured in the ObjectGrid descriptor file.

```
<register
        id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsxsdelements_register id"
        gridname="ObjectGrid name"
/>
```

server element

Use the server element to define a server, which can host a container, a catalog service, or both.

- Number of occurrences: Zero to many
- Child element: None

Attributes

id
: Specifies the name of the eXtreme Scale server.

tracespec
: Indicates the type of trace and enables trace and trace specification for the server.

tracefile
: Provides the path and name of the trace file to create and use.

statspec
: Indicates the statistic specification for the server.

jmxport
: Specifies the port that is used by the high availability (HA) manager for heartbeat communication between peer container servers. The haManagerPort port is only used for peer-to-peer communication between container servers that are in same domain. If the haManagerPort property is not defined, then

an ephemeral port is used. In WebSphere® Application Server, this setting is inherited by the high availability manager port configuration.

isCatalog
: Specifies whether the particular server hosts a catalog service. The default value is `false`.

name
: Specifies the name of the server.

haManagerPort
: Specifies the port that is used by the high availability (HA) manager for heartbeat communication between peer container servers. The haManagerPort port is only used for peer-to-peer communication between container servers that are in same domain. If the haManagerPort property is not defined, then an ephemeral port is used. In WebSphere Application Server, this setting is inherited by the high availability manager port configuration.

listenerHost
: Specifies the host name to which the Object Request Broker (ORB) transport protocol binds for communication. The value must be a fully qualified domain name or IP address. If your configuration involves multiple network cards, set the listener host and port to the IP address for which to bind. By setting the listener and host port, it allows the transport mechanism in the JVM know which IP address to use. If you do not specify which IP address to use, symptoms such as connection timeouts, unusual API failures, and clients that seem to hang can occur.

listenerPort
: Specifies the port number to which the Object Request Broker transport protocol binds for communication.

maximumThreadPoolSize
: Sets the maximum number of threads in the pool.

memoryThresholdPercentage
: Sets the memory threshold (percentage of max heap) for memory-based eviction.

minimumThreadPoolSize
: Sets the minimum number of threads in the pool.

workingDirectory
: The property that defines which directory the ObjectGrid server uses for all default settings.

zoneName
: Defines the zone to which this server belongs.

enableSystemStreamToFile
: Defines whether SystemOut and SystemErr is sent to a file.

enableMBeans
: Determines whether the ObjectGrid registers MBeans in this process.

serverPropertyFile
: Loads the server properties from a file.

catalogServerProperties
: Specifies the catalog server that hosts server.

```
<server
        id="_dcs_markdown_workspace_Transform_htmlout_0_com.ibm.websphere.extremescale.doc_rxsxsdelements_server id"
        tracespec="the server trace specification"
        tracefile="the server trace file"
        statspec="the server statistic specification"
        jmxport="JMX port number"
        isCatalog="true"|"false"
        name="the server name"
        haManagerPort="the haManager port"
        listenerHost="the orb binding host name"
        listenerPort="the orb binding listener port"
        maximumThreadPoolSize="the number of maximum threads"
        memoryThresholdPercentage="the memory threshold (percentage of max heap)"
        minimumThreadPoolSize="the number of minimum threads"
        workingDirectory="location for the working directory"
        zoneName="the zone name"
        enableSystemStreamToFile="true"|"false"
        enableMBeans="true"|"false"
        serverPropertyFile="location of the server properties file."
        catalogServerProperties="the catalog server properties reference"
/>
```

catalog element

Use the catalog element to route to container servers in the data grid.

- Number of occurrences: Zero to many
- Child element: None

Attributes

host
: Specifies the host name of the workstation where the catalog service is running.

port
: Specifies the port number paired with the host name to determine the catalog service port to which the client can connect.

```
<catalog
        host="catalog service host name"
        port="catalog service port number"
/>
```

catalogServerProperties element

Use the catalog server properties element to define a catalog service.

- Number of occurrences: Zero to many
- Child element: foreignDomains element

Attributes

catalogServerEndPoint
: Specifies the connection properties for the catalog server.

enableQuorum
: Determines whether to enable quorum.

heartBeatFrequencyLevel
: Sets the heartbeat frequency level.

domainName
: Defines the domain name used to uniquely identify this catalog service domain to clients when routing to multiple catalog service domains.

clusterSecurityURL
: Sets the location of the security file specific to the catalog service.

```
<catalogServerProperties
        catalogServerEndPoint="a catalog server endpoint reference"
        enableQuorum="true"|"false"
        heartBeatFrequencyLevel="
                HEARTBEAT_FREQUENCY_LEVEL_TYPICAL|
                HEARTBEAT_FREQUENCY_LEVEL_RELAXED|
                HEARTBEAT_FREQUENCY_LEVEL_AGGRESSIVE"
        domainName="the domain name used to uniquely identify this catalog service domain"
        clusterSecurityURL="the The cluster security file location.">
        <foreignDomains>
                <foreignDomain name="name_of_foreign_domain_1">
                        <endPoint host="catalog server host 1" port="2809"/>
                        <endPoint host="catalog server host 2" port="2809"/>
                </foreignDomain>
                <foreignDomain name="name_of_foreign_domain_2">
                        <endPoint host="catalog server host 3" port="2809"/>
                        <endPoint host="catalog server host 4" port="2809"/>
                </foreignDomain>
        </foreignDomains>
</catalogServerProperties>
```

foreignDomains element

Use the foreignDomains element to connect to a list of other catalog service domains. You must include the name of each catalog service domain and the end points for the catalog servers within each catalog service domain.

- Number of occurrences: Zero to many
- Parent element: catalogServerProperties element
- Child element: foreignDomain element

foreignDomain element

Indicates the name of the catalog service domain to connect. This name is defined with the domainName attribute on the catalogServiceProperties element.

- Number of occurrences: One to many
- Parent element: foreignDomains element
- Child element: endPoint element
- **Attributes**

  name
  : Specifies the name that identifies the foreign catalog service domain.

endPoint element

Indicates a list of catalog service end points for a specified foreign catalog service domain.

- Number of occurrences: One to many
- Parent element: foreignDomain element
- Child element: None
- **Attributes**

  host
  : Specifies the host name of one of the catalog servers in the catalog service domain.

  port
  : Specifies the port of one of the catalog servers in the catalog service domain.

catalog server endpoint element

Use the catalog server endpoint element to create a catalog server endpoint to be used by a catalog server element.

- Number of occurrences: Zero to many
- Child element: None

Attributes

serverName
: Specifies the name that identifies the process that you are launching.

hostName
: Specifies the host name for the machine where the server is launched.

clientPort
: Specifies the port used for peer catalog cluster communication.

peerPort
: Specifies the port used for peer catalog cluster communication.

```
<catalogServerEndPoint
        name="catalog server endpoint name"
        host=""
        clientPort=""
        peerPort=""
/>
```

container element

Use the container element to store the data itself.

- Number of occurrences: Zero to many
- Child element: None

Attributes

objectgridxml
: Specifies the path and name of the descriptor XML file to use that specifies characteristics for the ObjectGrid, including maps, locking strategy, and plug-ins.

deploymentxml
: Specifies the path and name of the XML file that is used with the descriptor XML file. This file determines partitioning, replication, number of initial containers, and other settings.

server
: Specifies the server on which the container is hosted.

```
<server
        objectgridxml="the objectgrid descriptor XML file"
        deploymentxml ="the objectgrid deployment descriptor XML file "
        server="the server reference "
/>
```

JPALoader element

Use the JPALoader element to synchronize the ObjectGrid cache with an existing backend data store when using the ObjectMap API.

- Number of occurrences: Zero to many
- Child element: None

Attributes

entityClassName
: Enables usage of JPAs such as EntityManager.persist and EntityManager.find. The entityClassName attribute is required for the JPALoader.

preloadPartition
: Specifies the partition number at which the map preload is started. If the value is less than 0, or greater than (totalNumberOfPartition – 1), the map preload is not started.

```
<JPALoader
        entityClassName="the entity class name"
        preloadPartition ="int"
/>
```

JPATxCallback element

Use the JPATxCallback element to coordinate JPA and ObjectGrid transactions.

- Number of occurrences: Zero to many
- Child element: None

Attributes

persistenceUnitName
: Creates a JPA EntityManagerFactory and locates the JPA entity metadata in the persistence.xml file. The persistenceUnitName attribute is required.

jpaPropertyFactory
: Specifies the factory to create a persistence property map to override the default persistence properties. This attribute references a bean.

exceptionMapper
: Specifies the ExceptionMapper plug-in that can be used for JPA-specific or database-specific exception mapping functions. This attribute references a bean.

```
<JPATxCallback
        persistenceUnitName="the JPA persistence unit name"
        jpaPropertyFactory ="JPAPropertyFactory bean reference"
        exceptionMapper="ExceptionMapper bean reference"
/>
```

JPAEntityLoader element

Use the JPAEntityLoader element to synchronize the ObjectGrid cache with an existing backend data store when using the EntityManager API.

- Number of occurrences: Zero to many
- Child element: None

Attributes

entityClassName
: Enables usage of JPAs such as EntityManager.persist and EntityManager.find. The entityClassName attribute is optional for the JPAEntityLoader element. If the element is not configured, the entity class configured in the ObjectGrid entity map is used. The same class must be used for the ObjectGrid EntityManager and for the JPA provider.

preloadPartition
: Specifies the partition number at which the map preload is started. If the value is less than 0, or greater than (totalNumberOfPartition – 1) the map preload is not launched.

```
<JPAEntityLoader
        entityClassName="the entity class name"
        preloadPartition ="int"
/>
```

LRUEvictor element

Use the LRUEvictor element to decide which entries to evict when a map exceeds its maximum number of entries.

- Number of occurrences: Zero to many
- Child element: None

Attributes

maxSize
: Specifies the total entries in a queue until the evictor must intervene.

sleepTime
: Sets the time in seconds between the eviction sweep over map queues to determine any necessary actions on the map.

numberOfLRUQueues
: Specifies the setting of how many queues the evictor must scan to avoid having a single queue that is the size of the entire map.

```
<LRUEvictor
        maxSize="int"
        sleepTime ="seconds"
        numberOfLRUQueues ="int"
/>
```

LFUEvictor element

Use the LFUEvictor element to determine which entries to evict when a map exceeds its maximum number of entries.

- Number of occurrences: Zero to many
- Child element: None

Attributes

maxSize
: Specifies the total entries that are allowed in each heap until the evictor must act.

sleepTime
: Sets the time in seconds between eviction sweeps over map heaps to determine any necessary actions on the map.

numberOfHeaps
: Specifies the setting of how many heaps the evictor must scan to avoid having a single heap that is the size of the entire map.

```
<LFUEvictor
        maxSize="int"
        sleepTime ="seconds"
        numberOfHeaps ="int"
/>
```

HashIndex element

Use the HashIndex element with Java™ reflection to dynamically introspect objects stored in a map when the objects are updated.

- Number of occurrences: Zero to many
- Child element: None

Attributes

name
Specifies the name of the index, which must be unique for each map.
attributeName
Specifies the name of the attribute to index. For field-access indexes, the attribute name is equivalent to the field name. For property-access indexes, the attribute name is the JavaBeans compatible property name.
rangeIndex
Indicates whether range indexing is enabled. The default value is `false`.
fieldAccessAttribute
Used for non-entity maps. The getter method is used to access the data. The default value is `false`. If you specify the value as `true`, the object is accessed using the fields directly.
POJOKeyIndex
Used for non-entity maps. The default value is `false`. If you specify the value as `true`, the index introspects the object in the key part of the map. This inspection is useful when the key is a composite key and the value does not have an embedded key. If you do not set the value or you specify the value as `false`, the index introspects the object in the value part of the map.

```
<HashIndex
        name="index name"
        attributeName="attribute name"
        rangeIndex ="true"|"false"
        fieldAccessAttribute ="true"|"false"
        POJOKeyIndex ="true"|"false"
/>
```

- Spring objectgrid.xsd file
  Use the Spring objectgrid.xsd file to integrate eXtreme Scale with Spring to manage eXtreme Scale transactions and configure clients and servers.

Related concepts:
Spring framework overview
Spring extension beans and namespace support
Related tasks:
Developing applications with the Spring framework
Starting a container server with Spring
Managing transactions with Spring
Troubleshooting XML configuration
Related reference:
Server properties file
Client properties file
REST data service properties file
ObjectGrid descriptor XML file
Deployment policy descriptor XML file
Entity metadata descriptor XML file
Security descriptor XML file
Liberty profile configuration files

Spring objectgrid.xsd file

Use the Spring objectgrid.xsd file to integrate eXtreme Scale with Spring to manage eXtreme Scale transactions and configure clients and servers.

See the Spring descriptor XML file for descriptions of the elements and attributes defined in the Spring objectgrid.xsd file.

Spring objectgrid.xsd file

Related concepts:
Spring framework overview
Spring extension beans and namespace support
Related tasks:
Developing applications with the Spring framework
Starting a container server with Spring
Managing transactions with Spring

Liberty profile configuration files

You can specify server properties and Liberty web feature properties for WebSphere® eXtreme Scale applications that run in the Liberty profile.

Use the following files to configure eXtreme Scale servers that run in the Liberty profile:

- Liberty profile server properties
  Use the options from the server properties file to configure WebSphere eXtreme Scale servers that run in the Liberty profile.
- Liberty profile web feature properties
  Specify the web feature for your server definition to identify web-based applications and add functions, such as session replication.
- Liberty profile webApp feature properties
  Specify the webApp feature to extend the Liberty profile web application. Add the webApp feature when you want to replicate HTTP session data for fault

tolerance.
- Liberty profile xsWebGrid feature properties
  Specify the webGrid feature to automatically start a container that hosts clients for HTTP session replication.
- Liberty profile xsDynacacheApp feature properties
  Specify Liberty profile to host the data grid, which you can configure as a dynamic cache provider with this feature.

Related reference:
Server properties file
Client properties file
REST data service properties file
ObjectGrid descriptor XML file
Deployment policy descriptor XML file
Entity metadata descriptor XML file
Security descriptor XML file
Spring descriptor XML file

Liberty profile server properties

Use the options from the server properties file to configure WebSphere® eXtreme Scale servers that run in the Liberty profile.

The server properties file contains several properties that define different settings for your server, such as security configuration. The server properties file is used by both catalog service and container servers in both stand-alone servers and servers that are hosted in WebSphere Application Server.

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 com.ibm.ws.xs.server.config element. See the following example from the server.xml file:

```
<server>
...
<com.ibm.ws.xs.server.config ... serverProps="/path/to/myServerProps.properties" ... />
</server>
```

Some properties that were formerly configurable in a stand-alone environment must be configured using the Liberty profile configuration instead of the eXtreme Scale configuration mechanisms.

- Logging and tracing settings must be specified using the logging element in the server.xml file, rather than being specified in the eXtreme Scale server properties file or com.ibm.ws.xs.server.config element.
- The working directory, like logging and tracing, is a server-wide setting, and therefore, they must be specified in a server-wide way.

If the previous settings are specified incorrectly, eXtreme Scale logs a warning message, which indicates that the settings are ignored.
Related concepts:
Liberty profile
Related reference:
Liberty profile web feature properties

Liberty profile web feature properties

Specify the web feature for your server definition to identify web-based applications and add functions, such as session replication.

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

You can set the following attributes on the `xsWebAppV85` element of the server.xml file:

Parameters

objectGridType
: A string value that is set to `REMOTE`, which specifies that session data is stored outside of the server on which the web application is running.

objectGridName
: A string value that defines the name of the ObjectGrid instance used for a particular web application. The default name is session.

  This property must reflect the objectGridName in both the ObjectGrid XML and deployment XML files used to start the eXtreme Scale container servers.

catalogHostPort
: The catalog server can be contacted to obtain a client side ObjectGrid instance. The value must be of the form `host:port<,host:port>`. The host is the listener host on which the catalog server is running. The port is the listener port for that catalog server process. This list can be arbitrarily long and is used for bootstrapping only. The first viable address is used. It is optional inside WebSphere® Application Server if the catalog.services.cluster property is configured.

replicationInterval
: An integer value (in seconds) that defines the time between writing of updated sessions to ObjectGrid. The default is `10` seconds. Possible values are from 0 to 60. 0 means that updated sessions are written to the ObjectGrid at the end of servlet service method call for each request. A higher replicationInterval value improves performance because fewer updates are written to the data grid. However, a higher value makes the configuration less fault tolerant.

This setting applies only when objectGridType is set to `REMOTE`.

sessionTableSize
: An integer value that defines the number of session references kept in memory. The default is `1000`.

  This setting pertains only to a `REMOTE` topology.

  Sessions are evicted from the in-memory table based on least recently used (LRU) logic. When a session is evicted from the in-memory table, it is invalidated from the web container. However, the data is not removed from the grid, so subsequent requests for that session can still retrieve the data. This value must be set higher than the web container maximum thread pool value, which reduces contention on the session cache.

fragmentedSession
: A string value of either `true` or `false`. The default value is `true`. Use this setting to control whether the product stores session data as a whole entry, or stores each attribute separately.

  Set the fragmentedSession parameter to `true` if the web application session has many attributes or attributes with large sizes. Set fragmentedSession to `false` if a session has few attributes, because all the attributes are stored in the same key in the data grid.

  In the previous, filter-based implementation, this property was referred to as persistenceMechanism, with the possible values of ObjectGridStore (fragmented) and ObjectGridAtomicSessionStore (not fragmented).

securityEnabled
: A string value of either `true` or `false`. The default value is `false`. This setting enables eXtreme Scale client security. It must match the securityEnabled setting in the eXtreme Scale server properties file. If the settings do not match, an exception occurs.

credentialAuthentication
: Indicates if credential authentication is enforced or supported.

  Never
  : No client certificate authentication is enforced.

  Required
  : Credential authentication is always enforced. If the server does not support credential authentication, the client cannot to connect to the server.

  Supported
  : (Default) Credential authentication is enforced only if both the client and server support credential authentication.

authenticationRetryCount
: Specifies the name of the class that implements the com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator interface. This class is used to get credentials for clients. The default value is `0`.

credentialGeneratorClass
: The name of the class that implements the com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator interface. This class is used to obtain credentials for clients.

credentialGeneratorProps
: The properties for the CredentialGenerator implementation class. The properties are set to the object with the setProperties(String) method. The credentialGeneratorProps value is used only if the value of the credentialGeneratorClass property is not null.

objectGridXML
: The file location of the objectgrid.xml file.

objectGridDeploymentXML
: Specifies the location of the objectGrid deployment policy XML file.

cookieDomain
: Specifies if you require sessions to be accessible across hosts. Set the value to the name of the common domain between the hosts.

cookiePath
: The default path is `"/"`.

reuseSessionID
: Set to `true` if the underlying web container reuses session IDs across requests to different hosts. The default value is `false`. The value of this property must be the same as the value in the web container. If you are using WebSphere Application Server and configuring eXtreme Scale HTTP session persistence using the administrative console or **wsadmin** tool scripting, the web container custom property `HttpSessionIdReuse=true` is added by default. The reuseSessionID is also set to `true`. If you do not want the session IDs to be reused, set the `HttpSessionIdReuse=false` custom property on the web container custom property before you configure eXtreme Scale session persistence.

shareSessionsAcrossWebApps
: Specifies if sessions are shared across web applications, specified as a string value of either `true` or `false`. The default is `false`. The servlet specification states that HTTP Sessions cannot be shared across web applications. An extension to the servlet specification is provided to allow this sharing.

useURLEncoding
: Set to `true` if you want to enable URL rewriting. The default value is `false`, which indicates that cookies are used to store session data. The value of this parameter must be the same as the web container settings for session management.

useCookies
: The default value is `false`.

Related tasks:
Enabling the eXtreme Scale web feature in the Liberty profile
Related reference:
Liberty profile xsWebGrid feature properties
Liberty profile webApp feature properties
Liberty profile server properties

Messages

When you encounter a message in a log or other parts of the product interface, you can look up the message by its component prefix to find out more information.

Finding messages

When you encounter a message in a log, copy the message number with its letter prefix and number and search in the information center (for example, `CWOBJ1526I`). When you search for the message, you can find an additional explanation of the message and possible actions you can take to resolve the problem.

To open the information center table of contents to the location of this reference information, click the Show in Table of Contents () button on your information center border.

Related tasks:
Enabling logging
Collecting trace
Starting stand-alone servers
Administering with the xscmd utility

CWOBJ: WebSphere eXtreme Scale messages for the ObjectGrid and related components

When you encounter a message in a log or other parts of the product interface, look up the message by its message ID to find out more information.

- CWOBJ0001E
  ILLEGAL_STATE_EXCEPTION_CWOBJ0001=CWOBJ0001E: Method, {0}, was called after initialization completed.
- CWOBJ0002W
  IGNORING_UNEXPECTED_EXCEPTION_CWOBJ0002=CWOBJ0002W: ObjectGrid component, {1}, is ignoring an unexpected exception: {0}.
- CWOBJ0003W
  DEPRECATED_FUNCTION_CWOBJ0003W=CWOBJ0003W: The {0} function was deprecated in the WebSphere eXtreme Scale {1} release and will be removed in a future release. See {2} in the information center for more information.
- CWOBJ0004W
  DEPRECATED_FUNCTION_CWOBJ0004W=CWOBJ0004W: The {0} method is deprecated. The {1} function was deprecated in the WebSphere eXtreme Scale {2} release and will be removed in a future release. See {3} in the information center for more information.
- CWOBJ0005W
  INTERRUPTED_EXCEPTION_CWOBJ0005=CWOBJ0005W: The thread created an java.lang.InterruptedException: {0}
- CWOBJ0006W
  GENERAL_EXCEPTION_WARNING_CWOBJ0006=CWOBJ0006W: An exception occurred: {0}
- CWOBJ0007W
  NULL_VALUE_WARNING_CWOBJ0007=CWOBJ0007W: The invalid value of null was specified for {0}. The default value of {1} will be used instead.
- CWOBJ0008E
  INVALID_VALUE_ERROR_CWOBJ0008=CWOBJ0008E: The value {0} provided for the property {1} is not valid.
- CWOBJ0010E
  MISSING_KEY_ERROR_CWOBJ0010=CWOBJ0010E: Message key {0} is missing.
- CWOBJ0012E
  INVALID_LOGELEMENT_TYPE_CWOBJ0012=CWOBJ0012E: The LogElement type code, {0} ({1}), is not recognized for this operation.
- CWOBJ0013E
  EVICT_ENTRIES_EXCEPTION_CWOBJ0013=CWOBJ0013E: An exception occurred while attempting to evict entries from the cache: {0}
- CWOBJ0021E
  OBJECT_TRANSFORMER_NOT_FOUND_CWOBJ0021=CWOBJ0021E: A usable ObjectTransformer instance was not found during the deserialization of the LogSequence object for {0} ObjectGrid and {1} ObjectMap.
- CWOBJ0022E
  LOCK_MANAGER_INTERNAL_ERROR_CWOBJ0022=CWOBJ0022E: The caller does not own mutex: {0}.
- CWOBJ0023E
  UNRECOGNIZED_COPY_MODE_CWOBJ0023=CWOBJ0023E: The CopyMode ({0}) is not recognized for this operation.
- CWOBJ0024E
  REQUIRED_FIELD_NOT_FOUND_CWOBJ0024=CWOBJ0024E: Cannot deserialize field {0} in class {1}. Deserialization failed.
- CWOBJ0025E
  SERIALIZATION_FAILED_CWOBJ0025=CWOBJ0025E: The serialization of the LogSequence object failed. The number of serialized LogElement objects ({0}) does not match the number of read LogElement objects ({1}).
- CWOBJ0026E
  INVALID_JMX_CREDENTIAL_CWOBJ0026=CWOBJ0026E: The JMX credential type is not right. It should be of type {0}.
- CWOBJ0027E
  CLONE_METHOD_NOT_SUPPORTED_CWOBJ0027=CWOBJ0027E: Internal runtime error. Clone method not supported: {0}
- CWOBJ0030I
  OBJECTGRID_INSTRUMENTATION_ENABLED_CWOBJ0030=CWOBJ0030I: ObjectGrid entity class instrumentation is enabled. The instrumentation mode is {0}.
- CWOBJ0033I
  CLASS_NOT_IMPLEMENT_CLONE_CWOBJ0033=CWOBJ0033I: Class, {0}, does not implement the clone method. Using serialization instead for this Class in map {1}.

- CWOBJ0034I
  TARGET_MEMORY_UTILIZATION_THRESHOLD_LEVEL_CWOBJ0034=CWOBJ0034I: Memory utilization threshold percentage is set to {0} %.
- CWOBJ0035W
  MEMORY_UTILIZATION_THRESHOLD_NOT_SUPPORTED_CWOBJ0035=CWOBJ0035W: Memory utilization threshold not supported for this JVM.
- CWOBJ0036W
  CHANGING_MEMORY_UTILIZATION_THRESHOLD_CWOBJ0036=CWOBJ0036W: Changing memory utilization threshold from {0} to {1} for {2} memory pool.
- CWOBJ0037W
  CHANGING_MEMORY_COLLECTION_UTILIZATION_THRESHOLD_CWOBJ0037=CWOBJ0037W: Changing memory collection utilization threshold from {0} to {1} for {2} memory pool.
- CWOBJ0038W
  MEMORY_THRESHOLD_EXCEEDED_CWOBJ0038=CWOBJ0038W: Memory threshold exceeded. Current heap memory usage: {0}.
- CWOBJ0039W
  MEMORY_COLLECTION_THRESHOLD_EXCEEDED_CWOBJ0039=CWOBJ0039W: Memory collection threshold exceeded. Current heap memory usage: {0}.
- CWOBJ0040E
  CWOBJ0040=CWOBJ0040E: Hash based data structure over run for {0} with {1} elements in the data structure. Examine the hashCode method on this class for better distribution.
- CWOBJ0041W
  RANGE_INDEX_NOT_SUPPORTED_CWOBJ0041=CWOBJ0041W: The rangeIndex property of HashIndex plug-in cannot be set to true for a composite index: {0}. The rangeIndex property setting will be ignored.
- CWOBJ0042E
  EXCEPTION_MAPPER_THROWABLE_IGNORED_CWOBJ0042=CWOBJ0042E: The ExceptionMapper implementation class {0} threw an unexpected exception with the following message: {1}. This exception is ignored.
- CWOBJ0043W
  CATALOG_CONFIG_PROBLEM_CELL_PROPERTY_CWOBJ0043=CWOBJ0043W: The {0} is formatted improperly but was corrected: {1}
- CWOBJ0044E
  FORMAT_ERROR_INITIALIZE_CATALOG_CWOBJ0044=CWOBJ0044E: Invalid data in the {0}: {1}. The exception is: {2}
- CWOBJ0045W
  ERROR_CREATING_MBEAN_CWOBJ0045=CWOBJ0045W: An exception occurred while creating an MBean with ObjectName: {0}. The exception is: {1}.
- CWOBJ0046I
  MEMORY_THRESHOLD_DEFAULT_PERCENT_USED_CWOBJ0046=CWOBJ0046I: The Java MemoryMXPool bean was not set (current value = 0). During initialization, the memoryThresholdPercentage property is set to default value of {0}.
- CWOBJ0047I
  DEVELOPMENT_MODE_ENABLED_CWOBJ0047=CWOBJ0047I: Development mode is enabled for one or more MapSets for ObjectGrids: {0}. For a production deployment, set the development mode attribute in the deployment policy file to false.
- CWOBJ0048E
  START_PROCESS_IN_WAS_CWOBJ0048=CWOBJ0048E: Starting stand-alone WebSphere eXtreme Scale server processes in a WebSphere Application Server 6.0.x deployment is not supported.
- CWOBJ0049W
  WAS_NOT_AUGMENTED_CWOBJ0049=CWOBJ0049W: This profile is not augmented with WebSphere eXtreme Scale. WebSphere eXtreme Scale container servers will therefore not start automatically.
- CWOBJ0050W
  INVALID_PORT_BOOTSTRAP_OVERRIDE_CWOBJ0050=CWOBJ0050W: Invalid listenerPort {0} defined in the {1}. Overriding it with the bootstrap address port (BOOTSTRAP_ADDRESS) {2}.
- CWOBJ0051W
  WAS_NOT_AUGMENTED_CWOBJ0051=CWOBJ0051W: This profile is not augmented with WebSphere eXtreme Scale. A catalog service will therefore not start automatically.
- CWOBJ0052I
  ORB_CHANNELFRAMEWORK_CWOBJ0052=CWOBJ0052I: The IBM ORB TransportMode property was set to ChannelFramework.
- CWOBJ0053I
  ORB_SEVERSOCKETQUEUEDEPTH_OVERRIDE_CWOBJ0053=CWOBJ0053I: The IBM ORB ServerSocketQueueDepth property was set to {0} to run with correctly with the ChannelFramework TransportMode.
- CWOBJ0054I
  WXS_PROPERTY_CWOBJ0054=CWOBJ0054I: The value of the "{0}" property is "{1}".
- CWOBJ0055W
  ORB_CHANNELFRAMEWORK_CWOBJ0055=CWOBJ0055W: The IBM ORB TransportMode property was set to ChannelFramework in the server properties file, but the existing orb.properties file already had a TransportMode set. The TransportMode will not be overridden.
- CWOBJ0056I
  ORB_PROPERTY_OVERRIDE_CWOBJ0056=CWOBJ0056I: The Object Request Broker (ORB) property, {0}, with the value, {1}, is being overridden with the value, {2}.
- CWOBJ0057E
  CATALOG_VERSION_DOWN_LEVEL_CWOBJ0057=CWOBJ0057E: The WebSphere eXtreme Scale catalog server version is {0}, and the client or container server version is {1}.
- CWOBJ0058I
  SAME_GRID_DIFFERENT_MAPSETS_CWOBJ0058=CWOBJ0058I: A deployment policy conflict was detected. Additional mapsets were found for ObjectGrid {0}.
- CWOBJ0059I
  DEFAULT_TRANSACTION_TIMEOUT_CWOBJ0059=CWOBJ0059I: The transaction time out value was not configured or was set to 0 for ObjectGrid {0}. With this configuration, transactions never time out. The transaction time out is being set to 600 seconds.
- CWOBJ0060W
  JVM_SHUTDOWN_HOOK_NOT_ORDERED_CWOBJ0060=CWOBJ0060W: The JVM shutdown hook is not ordered. The ORB shutdown hook might execute before eXtreme Scale shutdown hook executes. This may cause connectivity problem during the XS shutdown process.
- CWOBJ0061W
  TRANSACTION_ROLLED_BACK_CWOBJ0061W=CWOBJ0061W: The transaction with TxID, {0}, that was last running on thread, {1}, on shard {2} has

exceeded the configured transaction timeout value and was been marked rollback-only. This might be caused by lock contention or application deadlock, or your transaction timeout value is set too small.

- CWOBJ0062I
  ORB_PROPERTY_CWOBJ0062=CWOBJ0062I: The value of the "{0}" ORB property is "{1}".
- CWOBJ0063I
  ORB_DEFAULT_PROPERTY_SET_CWOBJ0063=CWOBJ0063I: The {0} property was not configured. The {0} property is being set to {1}.
- CWOBJ0064I
  MEMORY_THRESHOLD_USER_OVERRIDE_CWOBJ0064=CWOBJ0064I: The memoryThresholdPercentage property is provided in a server properties file, which overrides any previously set values.
- CWOBJ0065W
  HASHINDEX_ATTRIBUTE_NOT_FOUND_IN_SERIALIZER_METADATA_CWOBJ0065=CWOBJ0065W: The HashIndex, "{0}", for map "{1}" is enabled for multi-type access. The {2} attribute "{3}" was not defined in the {4} descriptor for the configured DataSerializer plug-in.
- CWOBJ0066W
  TRANSACTION_ROLLED_BACK_STATE_CWOBJ0066W=CWOBJ0066W: The {0} transaction has been marked rollback-only due to a state change of ObjectGrid {1} on shard {2} that forced transaction completion. This could be caused by an administrator changing the availability state of the ObjectGrid instance or termination of the ObjectGrid instance.
- CWOBJ0067W
  JMX_SSL_ENABLED_WITHOUT_PORT_CWOBJ0067W=CWOBJ0067W: SSL is enabled for JMX connections to this server. However, the JMXServicePort property was not provided.
- CWOBJ0068I
  JMX_SERVICE_URL_CWOBJ0068I=CWOBJ0068I: MBeanServer started with JMX URL {0}.
- CWOBJ0069W
  GRID_NOT_OFFHEAP_ELIGIBLE_CWOBJ0069=CWOBJ0069W: OffHeap is enabled but one of the BackingMaps for ObjectGrid "{0}" does not have a CopyMode of COPY_TO_BYTES or COPY_TO_BYTES_RAW. All BackingMaps for an ObjectGrid must be configured with either COPY_TO_BYTES or COPY_TO_BYTES_RAW to use OffHeap.
- CWOBJ0070W
  GC_POLICY_WARNING_CWOBJ0070W=CWOBJ0070W: The IBM implementation of the JVM is using a garbage collection policy that might affect performance.
- CWOBJ0071W
  MAX_HEAP_WARNING_CWOBJ0071W=CWOBJ0071W: The maximum heap size of {1} bytes surpasses the recommended maximum heap size of {0} bytes.
- CWOBJ0072I
  COMMAND_RUNAS_SUBJECT_CWOBJ0072I=CWOBJ0072I: The WebSphere eXtreme Scale command runtime is using the {0} Subject RunAs type.
- CWOBJ0080W
  NO_OBJECTGRID_XML_SPECIFIED_CWOBJ0080W=CWOBJ0080W: No objectgrid.xml specified for container, {0}. Will not process.
- CWOBJ0081I
  STARTING_CONTAINER_CWOBJ0081I=CWOBJ0081I: Starting container specified at: {0} with name, "{1}".
- CWOBJ0082W
  UNABLE_TO_LOAD_SERVER_PROPS_FILE_CWOBJ0082W=CWOBJ0082W: Unable to load properties from server properties file, {0}.
- CWOBJ0083W
  UNRECOGNIZED_SERVER_PROPERTY_CWOBJ0083W=CWOBJ0083W: Unrecognized XS server property specified: {0} Ignoring.
- CWOBJ0084W
  UNSUPPORTED_MULTI_PARAMETER_PROPERTY_CWOBJ0084W=CWOBJ0084W: Unsupported multi-parameter property: {0} - ignoring.
- CWOBJ0085W
  UNKNOWN_PARAMETER_TYPE_CWOBJ0085W=CWOBJ0085W: Unknown parameter type ({0}) while setting config property, {1} - ignoring.
- CWOBJ0086W
  FAILED_TO_SET_PROPERTY_CWOBJ0086W=CWOBJ0086W: Failed to set user property ({0}) : {1}.
- CWOBJ0087W
  PROPERTY_CHANGE_SERVER_RESTART_CWOBJ0087W=CWOBJ0087W: Property, {0}, has changed from {1} to {2} but will not take effect until the server is restarted.
- CWOBJ0088W
  UNRECOGNIZED_WEB_APP_CONFIG_PROPERTY_CWOBJ0088W=CWOBJ0088W: Unrecognized XS Web App Configuration property specified: {0} Ignoring.
- CWOBJ0900I
  CWOBJ0900=CWOBJ0900I: The ObjectGrid runtime component is started for server {0}.
- CWOBJ0901E
  CWOBJ0901=CWOBJ0901E: {0} system property is required to start ObjectGrid runtime component for server: {1}.
- CWOBJ0902W
  CWOBJ0902=CWOBJ0902W: An error prevented the ObjectGrid runtime component from starting for server: {0}.
- CWOBJ0903I
  INTERNAL_OBJECTGRID_VERSION_CWOBJ0903=CWOBJ0903I: The internal version of WebSphere eXtreme Scale is {0}.
- CWOBJ0904E
  FILES_DO_NOT_EXIST_CWOBJ0904=CWOBJ0904E: {0} exists but the following file or files are missing: {1}. Cannot start the ObjectGrid runtime component for server: {2}.
- CWOBJ0905I
  FILES_NOT_FOUND_CWOBJ0905=CWOBJ0905I: WebSphere eXtreme Scale did not find object grid configuration files packaged with application {0}.
- CWOBJ0910I
  CWOBJ0910=CWOBJ0910I: The ObjectGrid runtime component is stopped for server {0}.
- CWOBJ0912E
  CWOBJ0912=CWOBJ0912E: The application {0} has ObjectGrid configuration files that will not be used because application {1} is currently running an ObjectGrid server instance.
- CWOBJ0913I
  LOADED_PROPERTY_FILES_CWOBJ0913=CWOBJ0913I: Server property files have been loaded: {0}.
- CWOBJ0915I
  ORB_VERSION_USED_CWOBJ0915=CWOBJ0915I: ORB version used is {0}.

- CWOBJ0917I
  ORB_LISTENING_CWOBJ0917=CWOBJ0917I: {0} ORB is listening on host and port {1}:{2}.
- CWOBJ0918W
  NON_OBJECTGRID_CONFIG_OBJECT_CWOBJ0918=CWOBJ0918W: The list that was supplied to override client-side ObjectGrid settings for domain/cluster {0} contains an element that is not an ObjectGridConfiguration object. This element will be removed from the List: {1}
- CWOBJ0919W
  SERVER_PROPERTY_NOT_FOUND_CWOBJ0919=CWOBJ0919W: The server property file {0} cannot be found. All server properties are set to the default values.
- CWOBJ0920I
  CATALOG_SERVER_NOT_STARTED_FOR_PROCESS_CWOBJ0920=CWOBJ0920I: The catalog service was not started in this process: {0}. The {1} is: {2}
- CWOBJ0921W
  OLD_CLIENT_PROP_FILE_USED_CWOBJ0921W=CWOBJ0921W: Using the Java Virtual Machine system property name "com.ibm.websphere.objectgrid.ClientProperties" to set the ObjectGrid client property file is deprecated. Use the property "objectgrid.client.props" instead.
- CWOBJ0922W
  CLIENT_PROP_FILE_NOT_FOUND_CWOBJ0922W=CWOBJ0922W: The ObjectGrid client property file {0} cannot be found.
- CWOBJ0923W
  DEPRECATED_SERVER_SECURITY_PROP_FILE_USED_CWOBJ0923W=CWOBJ0923W: Using the Java Virtual Machine system property name "objectgrid.security.server.props" to set the ObjectGrid server security properties is deprecated. Use the property "objectgrid.server.props" instead.
- CWOBJ0924I
  LOADED_CLIENT_PROPERTY_FILES_CWOBJ0924I=CWOBJ0924I: The client property file {0} has been loaded.
- CWOBJ0925E
  AUTO_START_PROP_NOT_FOUND_CWOBJ0925E=CWOBJ0925E: A container autostart file {0} was found in the classpath, but the {1} property was not specified.
- CWOBJ1001I
  OPEN_FOR_BUSINESS_CWOBJ1001=CWOBJ1001I: ObjectGrid Server {0} is ready to process requests.
- CWOBJ1003I
  DCS_CWOBJ1003=CWOBJ1003I: DCS Adapter service is disabled by configuration, to enable it, please change your configuration with an endpoint defined.
- CWOBJ1004E
  SERVER_TOPIC_CWOBJ1004=CWOBJ1004E: Server topic is null
- CWOBJ1005E
  CLIENT_REQUESTQ_CWOBJ1005=CWOBJ1005E: The incoming request queue is null.
- CWOBJ1006E
  CLIENT_RESULTQ_CWOBJ1006=CWOBJ1006E: The outgoing result queue is null.
- CWOBJ1007E
  CLIENT_REQUEST_CWOBJ1007=CWOBJ1007E: ObjectGrid client request is null.
- CWOBJ1008E
  TXID_CWOBJ1008=CWOBJ1008E: ObjectGrid client request TxID is null.
- CWOBJ1013W
  EXCEPTION_ON_SERVER_CWOBJ1013=CWOBJ1013W: An exception occurred on a remote server: {0}
- CWOBJ1014I
  CLASSPATH_PROBLEM_CWOBJ1014=CWOBJ1014I: Preceding {0} message may be caused by application classes missing from the classpath on the server.
- CWOBJ1015I
  OBJECTTRANSFORMER_PROBLEM_CWOBJ1015=CWOBJ1015I: Preceeding {0} message may be caused by an incorrect application implementation of the ObjectTransformer or Serializable interface
- CWOBJ1016E
  PROPERTY_FILE_DOES_NOT_EXIST_CWOBJ1016E=CWOBJ1016E: The property file {0} does not exist: {1}.
- CWOBJ1118I
  DCS_CWOBJ1118=CWOBJ1118I: ObjectGrid Server Initializing [Cluster: {0} Server: {1}].
- CWOBJ1119I
  CLIENT_CWOBJ1119=CWOBJ1119I: ObjectGrid client failed to connect to host: {0} port: {1}.
- CWOBJ1120I
  CLIENT_CWOBJ1120=CWOBJ1120I: ObjectGrid Client connected successfully to host: {0} port: {1}.
- CWOBJ1121W
  TIMEOUT_DURING_SHUTDOWN_WORK_LEFT_CWOBJ1121W=CWOBJ1121W: Timeout during shutdown while waiting for work items to complete. Work items left: {0}
- CWOBJ1122W
  TIMEOUT_DURING_SHUTDOWN_SHARDS_LEFT_CWOBJ1122W=CWOBJ1122W: Time out while waiting for shards to be moved off server. Shards left: {0}
- CWOBJ1123W
  SERVER_DISCONNECTED_FROM_CATALOG_SERVER_CWOBJ1123W=CWOBJ1123W: Server was disconnected from the primary catalog service and cannot be reconnected.
- CWOBJ1124W
  DIFFERENT_CATALOG_SERVER_TIMESTAMPS_CWOBJ1124W=CWOBJ1124W: The container server database timestamps are normalized with different catalog servers {0} and {1}. Make sure the clocks of these two servers are synchronized.
- CWOBJ1125W
  CONTAINER_NOT_REGISTERED_CWOBJ1125W=CWOBJ1125W: The server was unable to register container, {0}, with the catalog server due to an exception. {1}
- CWOBJ1126I
  CLIENT_CONNECT_CWOBJ1126=CWOBJ1126I: The ObjectGrid client has connected to the {0} grid in the {1} domain using connection {2}.
- CWOBJ1127I
  CLIENT_DISCONNECT_CWOBJ1127=CWOBJ1127I: The ObjectGrid client connection {0} has disconnected from the {1} domain. ObjectGrids used by this connection were {2}.

- CWOBJ1128I
  CLIENT_CACHE_MAPS_CWOBJ1128=CWOBJ1128I: The client cache is enabled for maps {0} on the {1} ObjectGrid.
- CWOBJ1129W
  SHARDS_LEFT_ON_TERMINATE_CWOBJ1129=CWOBJ1129W: Some of the shards were not removed before the container terminate completed on {0} container. Shards left: {1}
- CWOBJ1130W
  COMM_ERROR_WITH_SHARD_CWOBJ1130=CWOBJ1130W: Communication with the partition with the domain:grid:mapSet:partitionId {0} failed with an Object Request Broker (ORB) exception communicating with {1} at {2}.
- CWOBJ1131I
  RECEIVED_ROUTE_UPDATE_FROM_CONTAINER_CWOBJ1131=CWOBJ1131I: An updated routing entry for the partition with domain:grid:mapSet:partitionId:epoch {0} was sent to this client.
- CWOBJ1132I
  RECEIVED_ROUTE_UPDATE_FROM_CATALOG_CWOBJ1132=CWOBJ1132I: An updated routing entry for domain:grid:epoch {0} was obtained from the catalog server.
- CWOBJ1133W
  RETRY_COMM_WITH_SHARD_CWOBJ1133=CWOBJ1133W: After an initial communication failure, an attempt to access the partition for domain:grid:mapSet:partitionId {0} on the {1} container server at {2} was successful.
- CWOBJ1203W
  CLIENT_RESPONSE_TIMEOUT_CWOBJ1203W=CWOBJ1203W: Received a timeout event from the server for transaction: {0}
- CWOBJ1204W
  UNKNOWN_MESSAGE_TYPE_CWOBJ1204W=CWOBJ1204W: Received a message of unknown message type. The message is: {0}
- CWOBJ1207W
  CONFIG_PROPERTY_UNSUPPORTED_CWOBJ1207W=CWOBJ1207W: The property {0} on plug-in {1} is using an unsupported property type.
- CWOBJ1208W
  CONFIG_PLUGIN_UNSUPPORTED_CWOBJ1208W=CWOBJ1208W: The specified plug-in type {0} is not supported.
- CWOBJ1209E
  UNABLE_TO_ACTIVATE_SHARD_CWOBJ1209E=CWOBJ1209E: Unable to activate shard for ObjectGrid {0}, domain {1}, map set {2}, partition {3}, shard type {4} ({0}:{2}:{3}) due to exception: {5}
- CWOBJ1210E
  UNABLE_TO_RETURN_SHARD_CWOBJ1210E=CWOBJ1210E: Unable to return shard for ObjectGrid {0}, domain {1}, map set {2}, partition {3}, shard type {4} ({1}:{0}:{2}:{3}) due to exception: {5}
- CWOBJ1211E
  ERROR_OG_PMI_CREATE_FAILED_CWOBJ1211E=CWOBJ1211E: The Performance Monitoring Infrastructure (PMI) creation of {0} failed. The exception is {1}.
- CWOBJ1212I
  PMI_NOT_FOUND=CWOBJ1212I: The WebSphere Application Server Performance Monitoring Infrastructure (PMI) cannot be found.
- CWOBJ1213I
  SERVER_RECONNECTED_WITH_CATALOG_SERVER_CWOBJ1213I=CWOBJ1213I: Server was disconnected from the primary catalog server but was able to reconnect.
- CWOBJ1214I
  PLACEMENT_BALANCE_STATUS_CWOBJ1214I=CWOBJ1214I: Shard balancing for ObjectGrid {0}:{1} is {2}.
- CWOBJ1215I
  TP_CWOBJ1215=CWOBJ1215I: ObjectGrid Transaction Propagation Event Listener is initializing [ObjectGrid {0}].
- CWOBJ1216I
  TP_CWOBJ1216=CWOBJ1216I: ObjectGrid Transaction Propagation Event Listener is initialized [ObjectGrid {0}].
- CWOBJ1217I
  TP_CWOBJ1217=CWOBJ1217I: ObjectGrid Transaction Propagation Service Point Initialized [ObjectGrid {0}].
- CWOBJ1218E
  TP_CWOBJ1218=CWOBJ1218E: ObjectGrid Transaction Propagation Event Listener failure occurred [ObjectGrid {0} Exception message {1}].
- CWOBJ1219E
  TP_CWOBJ1219=CWOBJ1219E: ObjectGrid Transaction Propagation Service End Point failure occurred [ObjectGrid {0} Exception message {1}].
- CWOBJ1220E
  TRANPROPLISTENER_UNSUPPORTED_CWOBJ1220=CWOBJ1220E: ObjectGrid Transaction Propagation Service is not supported in this environment.
- CWOBJ1221E
  PLUGIN_FAILED_CWOBJ1221=CWOBJ1221E: The plug-in implemented by class {0} failed during a call to method {1}, the exception is: {2}
- CWOBJ1222E
  PLUGIN_INCORRECT_CWOBJ1222=CWOBJ1222E: The plug-in implemented by class {0} is in an incorrect state or has an incorrect status as indicated by method {1}.
- CWOBJ1223E
  INVALID_QUERY_SCHEMA_CONFIG_CWOBJ1223=CWOBJ1223E: An invalid ObjectQuery schema configuration is defined. The following maps have both an ObjectQuery configuration and a serializer or an entity configuration: {0}
- CWOBJ1224I
  RESTART_EXITING_JVM_CWOBJ1224I=CWOBJ1224I: The JVM process is ending because a replacement JVM has started.
- CWOBJ1225E
  RESTART_JVM_FAILED_CWOBJ1225E=CWOBJ1225E: The JVM did not restart.
- CWOBJ1226E
  RESTART_PARENT_TIMEOUT_CWOBJ1226E=CWOBJ1226E: The parent JVM did not terminate within the timeout period ({0} ms). Discontinuing with startup of the child JVM.
- CWOBJ1227I
  SERVER_REBOOT_TO_CONNECT_WITH_CATALOG_SERVER_CWOBJ1227I=CWOBJ1227I: Server was disconnected from the primary catalog server so we will restart it to reconnect.
- CWOBJ1228I
  CONTAINER_RECONNECT_IGNORED_CWOBJ1228I=CWOBJ1228I: The server ignored a request to reconnect its containers because a previous reconnect request was just completed.

- CWOBJ1250W
  UPGRADE_CATALOG_CWOBJ1250=CWOBJ1250W: A client with a version greater than or equal to {0} is connecting to a catalog service that has a version less than {0}. The catalog service must be upgraded before clients are upgraded.
- CWOBJ1251I
  QUORUM_ENABLED_CWOBJ1251I=CWOBJ1251I: Quorum is enabled for the catalog service.
- CWOBJ1252I
  QUORUM_DISABLED_CWOBJ1252I=CWOBJ1252I: Quorum is disabled for the catalog service.
- CWOBJ1253I
  QUORUM_OVERRIDE_CWOBJ1253I=CWOBJ1253I: Quorum has been overridden for the catalog service.
- CWOBJ1300I
  PMA_CWOBJ1300=CWOBJ1300I: Adapter successfully initialized ObjectGrid.
- CWOBJ1301E
  PMA_CWOBJ1301=CWOBJ1301E: Adapter failed to initialize ObjectGrid. Exception occurred {0}.
- CWOBJ1302I
  PMA_CWOBJ1302=CWOBJ1302I: Adapter stopped.
- CWOBJ1303I
  PMA_CWOBJ1303=CWOBJ1303I: Adapter started.
- CWOBJ1304I
  SECURITY_ENABLED_CWOBJ1304=CWOBJ1304I: Security is enabled.
- CWOBJ1305I
  SECURITY_DISABLED_CWOBJ1305=CWOBJ1305I: Security is disabled.
- CWOBJ1306W
  CANNOT_RETRIEVE_CLIENT_CERTS_CWOBJ1306=CWOBJ1306W: Cannot retrieve the client certificates from the SSL socket.
- CWOBJ1307I
  OBJECTGRID_SECURITY_ENABLED_CWOBJ1307=CWOBJ1307I: Authorization security for ObjectGrid {0} is enabled.
- CWOBJ1308I
  OBJECTGRID_SECURITY_DISABLED_CWOBJ1308=CWOBJ1308I: Security of the ObjectGrid instance {0} is disabled.
- CWOBJ1309E
  OBJECTGRID_CONNECT_TOKEN_CREATION_CWOBJ1309=CWOBJ1309E: Unexpected error occured in the connect token creation: {0}
- CWOBJ1310E
  OBJECTGRID_CONNECT_TOKEN_VALIDATION_CWOBJ1310=CWOBJ1310E: An attempt by another process to connect to this process through the core group transport has been rejected. The connecting process provided a source core group name of {0}, a target of {1}, a member name of {2} and an IP address of {3}. The error message is {4}.
- CWOBJ1311W
  IGNORE_CREDENTIAL_GENERATOR_PROPS=CWOBJ1311W: The credentialGeneratorProps setting is ignored since the credentialGeneratorClass value is not provided.
- CWOBJ1312W
  EXPIRED_CREDENTIAL_EXCEPTION=CWOBJ1312W: The credential expired. The exception message is {0}.
- CWOBJ1313W
  CUSTOM_SECURE_TOKEN_MANAGER_CLASS_IGNORED=CWOBJ1313W: The customSecureTokenManagerClass setting is ignored since the provided customSecureTokenManagerType value is not "custom".
- CWOBJ1314W
  IGNORE_CREDENTIAL_GENERATOR_CLASS_CWOBJ1314W=CWOBJ1314W: The credentialGeneratorClass property with value "{0}" is being overridden.
- CWOBJ1315I
  DYNAMIC_CREDENTIAL_GENERATOR_CLASS_CWOBJ1315I=CWOBJ1315I: The credentialGeneratorClass property was set dynamically to a value of "{0}".
- CWOBJ1316W
  CLIENT_SECURITY_ENABLED_SERVER_SECURITY_DISABLED_CWOBJ1316W=CWOBJ1316W: This non-secure server received a client request containing credential information. The credential information will be ignored by this server.
- CWOBJ1317W
  UNSUPPORTED_ENCODE_ALGORITHM_CWOBJ1317W=CWOBJ1317W: The property {0} is encoded with an unsupported encoding algorithm "{1}". The property will be ignored.
- CWOBJ1318I
  SSL_TRANSPORT_TYPE_ENABLED_CWOBJ1318I=CWOBJ1318I: Transport layer security configuration is set to {0}.
- CWOBJ1319E
  SECURITY_NOT_PROVIDED_ON_STOP_CWOBJ1319=CWOBJ1319E: The exception {0} occured attempting to stop the server. Verify that a client property file was provided with the stop command including the required security settings.
- CWOBJ1320E
  FILEPASSWORDENCODER_ERROR_CWOBJ1320E=CWOBJ1320E: An error occured while processing the FilePasswordEncoder request: {0} {1}
- CWOBJ1321I
  FILEPASSWORDENCODER_INFO_CWOBJ1321I=CWOBJ1321I: FilePasswordEncoder informational message: {0} {1}
- CWOBJ1322E
  CLIENTSECURITYCONTEXT_ERROR_CWOBJ1322E=CWOBJ1322E: Internal runtime error occured for client request on thread {0}. Security Context Map information is {1}
- CWOBJ1400W
  MULTIPLE_JAR_FILE_CWOBJ1400W=CWOBJ1400W: Detected multiple ObjectGrid runtime JAR files in the JVM. Using multiple ObjectGrid runtime JAR files may cause problems.
- CWOBJ1401E
  WRONG_JAR_FILE_CWOBJ1401E=CWOBJ1401E: Detected a wrong ObjectGrid runtime JAR file for this configuration. Detected configuration is {0}. Expected JAR file is {1}.
- CWOBJ1402E
  MISSING_CONNECTION_LINK_CALLBACK_CWOBJ1402E=CWOBJ1402E: ObjectGrid connection link callback not found for id: {0}
- CWOBJ1403E
  INVALID_RESOURCE_CWOBJ1403E=CWOBJ1403E: The resource specified is invalid: {0}

- CWOBJ1504E
  CANNOT_PROCESS_REPLICA_CHANGES_CWOBJ1504=CWOBJ1504E: An exception occurred when attempting to process the LogSequences for replica ({0}): {1}.
- CWOBJ1505E
  MORE_THAN_ONE_PRIMARY_RESPONSE_CWOBJ1505=CWOBJ1505E: More than one replication group member reported back as the primary. Only one primary can be active. ({0}).
- CWOBJ1506E
  POSSIBLE_NETWORK_PARTITION_CWOBJ1506=CWOBJ1506E: More than one primary replication group member exists in this group ({1}). Only one primary can be active. ({0}).
- CWOBJ1508E
  CANNOT_SEND_MESSAGE_CWOBJ1508=CWOBJ1508E: An exception occurred when attempting to send message ({0}) from sender ({1}) to receiver ({2}): {3}.
- CWOBJ1509E
  CANNOT_SERIALIZE_MESSAGE_CWOBJ1509=CWOBJ1509E: An exception occurred when attempting to serialize message ({0}): {1}.
- CWOBJ1510E
  CANNOT_DESERIALIZE_MESSAGE_CWOBJ1510=CWOBJ1510E: An exception occurred when attempting to inflate message ({0}): {1}.
- CWOBJ1511I
  OPEN_FOR_BUSINESS_CWOBJ1511=CWOBJ1511I: {0} ({1}) is open for business.
- CWOBJ1513E
  SYNCH_REPLICATION_FAILED_CWOBJ1513=CWOBJ1513E: Synchronous replication failed on {0} ({1}). This member is no longer active.
- CWOBJ1514I
  PRIMARY_DOWNGRADED_CWOBJ1514=CWOBJ1514I: Primary ({0}) is being downgraded to either a replica or standby.
- CWOBJ1515I
  MIN_CONFIG_NOT_MET_CWOBJ1515=CWOBJ1515I: Minimum configuration requirements not satisfied for replication group ({0}).
- CWOBJ1516E
  CANNOT_ACTIVATE_OBJECTGRID_CWOBJ1516=CWOBJ1516E: An exception occurred when attempting to activate the replication process for ObjectGrid ({0}): {1}.
- CWOBJ1518E
  CANNOT_COMMIT_REPLICA_CHANGES_CWOBJ1518=CWOBJ1518E: An exception occurred when attempting to commit replica transaction ({0}) for primary transaction ({1}) on Replica ({2}): {3}.
- CWOBJ1519E
  CANNOT_ROLLBACK_REPLICA_CHANGES_CWOBJ1519=CWOBJ1519E: An exception occurred when attempting to rollback the LogSequences for replica ({0}): {1}
- CWOBJ1524I
  LISTENER_REREGISTER_CWOBJ1524=CWOBJ1524I: Replica listener {0} must re-register with the primary. Reason: {1}
- CWOBJ1525I
  CHECKPRELOADSTATE_EXCEPTION_CWOBJ1525=CWOBJ1525I: A ReplicaPreloadController ({0}) for map {1} threw an unexpected exception in method checkPreloadState {2}
- CWOBJ1526I
  ENTERING_PEER_MODE_CWOBJ1526=CWOBJ1526I: Replica {0} entering peer mode after {1} seconds, replicating from primary on {2}
- CWOBJ1527W
  FAILED_ENTERING_PEER_MODE_CWOBJ1527=CWOBJ1527W: Replica {0} failed to enter peer mode after {1} seconds
- CWOBJ1531I
  CLOSED_FOR_BUSINESS_CWOBJ1531=CWOBJ1531I: {0} ({1}) stopped on this server.
- CWOBJ1532I
  SHARD_TRANSITION_CWOBJ1532=CWOBJ1532I: {0} (moving from server {1} ({2}), promoting {3} to {4}) in transition.
- CWOBJ1533E
  REPLICA_IN_PEER_MODE_CWOBJ1533=CWOBJ1533E: {0}:{1} is already in peer mode.
- CWOBJ1535E
  REPLICA_FAIL_ON_PEER_MODE_BAD_TRAN_CWOBJ1535=CWOBJ1535E: {0}:{1} Replica map failed to enter peer mode. A transaction threw an error while copying data from the primary. Error received: {2}
- CWOBJ1536E
  REPLICA_FAIL_ON_PEER_MODE_ORDERING_CWOBJ1536=CWOBJ1536E: {0}:{1} Replica map failed to enter peer mode. Received incorrect ordering data from the primary, data copy cannot complete.
- CWOBJ1537E
  REPLICA_FAIL_TO_REREGISTER_CWOBJ1537=CWOBJ1537E: {0} exceeded the maximum number of times to reregister ({1}) without successful transactions.
- CWOBJ1538E
  PRIMARY_REJECT_SAME_REPLICA_CWOBJ1538=CWOBJ1538E: {0} received a {1} replica with the same container name as the local container. The replica will not be placed. Container: {2}.
- CWOBJ1539W
  CLIENT_REPLICA_SERIALIZATION_ERROR_CWOBJ1539=CWOBJ1539W: Unable to serialize client listener data to send to replica, the client listener may not failover: {0}
- CWOBJ1540E
  CREATING_CLIENT_REPLICA_TIMED_OUT_CWOBJ1540=CWOBJ1540E: Creating a client replica map times out after {0} ms.
- CWOBJ1541E
  REPLICA_FAIL_ON_PRIOR_MAP_CWOBJ1541=CWOBJ1541E: {0}:{1} Replica map failed to enter peer mode because a previous map failed to enter peer mode. The entire replica cannot continue to enter peer mode. The prior exception was {2}.
- CWOBJ1542I
  FOREIGN_PRIMARY_REPLICATING_CWOBJ1542=CWOBJ1542I: Primary {0} started or continued replicating from {3} primary ({1}). Replicating for maps: {2}
- CWOBJ1543I
  REPLICA_REPLICATING_CWOBJ1543=CWOBJ1543I: The {0} {1} started or continued replicating from the primary on {3}. Replicating for maps: {2}
- CWOBJ1544I
  FOREIGN_PRIMARY_REPLICATING_CWOBJ1544=CWOBJ1544I: Primary {0} stopped replicating from {2} primary ({1}).

- CWOBJ1545W
  REPLICA_NOT_PLACED_CWOBJ1545=CWOBJ1545W: Primary {0} could not place a ({1}) on {2}. The remote container did not respond or returned an error.
- CWOBJ1546W
  FOREIGN_PRIMARY_NOT_ADDED_CWOBJ1546=CWOBJ1546W: Primary {0} could not initiate replication to a {2} primary on {1}. The remote container did not respond or returned an error.
- CWOBJ1547I
  SHARD_DEMOTION_CWOBJ1547=CWOBJ1547I: {0} (demoting {1} to {2}) in transition.
- CWOBJ1548W
  ENTERING_PEER_MODE_CWOBJ1548=CWOBJ1548W: Replica shard {0} did not enter peer mode, and the replication from the primary shard in the {1} container failed.
- CWOBJ1549I
  COPY_FROM_REPLICA_CWOBJ1549=CWOBJ1549I: The transitioning primary shard ({0}) did not finish copying data from the previous primary shard on the {1} primary container. The transitioning primary shard will replicate from the existing {2} shard on the {3} replica container.
- CWOBJ1550W
  ERROR_REPLICATING_FROM_PRIMARY_CWOBJ1550=CWOBJ1550W: The {1} ({0}) shard received exceptions while replicating from the primary shard on the {2} primary container. The {1} shard will continue to poll the primary shard. Exception received: {3}
- CWOBJ1551I
  RECOVERED_REPLICATING_FROM_PRIMARY_CWOBJ1551=CWOBJ1551I: The {1} ({0}) shard successfully recovered and replicated after several exceptions from the primary shard on the {2} primary container.
- CWOBJ1552W
  REPLICA_NOT_REPLICATING_CWOBJ1552=CWOBJ1552W: The {0} {1} could not start replicating from the primary on {3}. Could not replicate for maps, {2}, because {4}.
- CWOBJ1553I
  REPLICA_IDLE_TIME_CWOBJ1553=CWOBJ1553I: The maximum replication idle level is set to {0} ({1}).
- CWOBJ1554E
  REPLICATION_IDLE_INCORRECT_CWOBJ1554=CWOBJ1554E: The replication idle level was set to an invalid value of {0}. Valid levels are {1}.
- CWOBJ1555E
  REPLICA_INITIALIZATION_FAIL_CWOBJ1555=CWOBJ1555E: The {0} ({1}) shard failed to initialize. The shard was added by a primary shard on the {2} primary container. The initialization exception is {3}
- CWOBJ1556I
  REPLICA_ORPHANED_CWOBJ1556=CWOBJ1556I: The {0} ({1}) shard is orphaned and does not have a valid primary shard. The last primary shard for this {0} shard is on {2} primary container. This shard will be stopped.
- CWOBJ1557W
  REPLICA_RECEIVED_OLD_PRIMARY_TRAN_CWOBJ1557=CWOBJ1557W: The {0} ({1}) shard received a transaction from a primary shard on the {2} primary container. The current primary shard is on the {3} primary container. The primary on the {2} primary container is an old primary and should be stopped.
- CWOBJ1558E
  PRIMARY_FAILED_ACTIVATION_CWOBJ1558=CWOBJ1558E: The {0} ({1}) shard failed to activate. The exception that occurred is {2}.
- CWOBJ1559I
  SCHEDULE_ROUTE_UPDATE_CWOBJ1559=CWOBJ1559I: The routing update for grid:mapSet:partitionId:epoch {0} is scheduled for transfer to the catalog server.
- CWOBJ1560I
  ROUTE_UPDATE_SENT_CWOBJ1560=CWOBJ1560I: The next set of routing updates to transfer has been sent to the catalog server.
- CWOBJ1561I
  ROUTE_TRANSFERS_TO_CLIENT_START_CWOBJ1561=CWOBJ1561I: Routing updates are beginning to directly transfer to eXtreme Scale clients.
- CWOBJ1562I
  ROUTE_TRANSFERS_TO_CLIENT_STOP_CWOBJ1562=CWOBJ1562I: Routing updates are no longer being directly transferred to eXtreme Scale clients.
- CWOBJ1600I
  GATEWAY_STARTED_CWOBJ1600=CWOBJ1600I: ManagementGateway service started on port ({0}).
- CWOBJ1601E
  GATEWAY_SERVICE_FAILED_CWOBJ1601=CWOBJ1601E: ManagementGateway service failed to start on port ({0}).
- CWOBJ1602E
  GATEWAY_CLIENT_CONNECT_FAILED_CWOBJ1602=CWOBJ1602E: ManagementGateway service failed to connect to server at ({0}):({1}).
- CWOBJ1603E
  MANAGEMENT_SERVICE_RESPONSE_FAILED_CWOBJ1603=CWOBJ1603E: Management service failed to respond to ({0}) remote request: {1}.
- CWOBJ1604I
  MANAGEMENT_GATEWAY_STOP_FAILED_CWOBJ1604=CWOBJ1604I: ManagementGateway service failed to stop connector due to Throwable {0}. Exiting.
- CWOBJ1605I
  MANAGEMENT_GATEWAY_REFRESH_FAILED_CWOBJ1605=CWOBJ1605I: ManagementGateway caught Throwable {0} while refreshing attributes. Exiting.
- CWOBJ1606I
  NO_RESPONSE_FROM_SERVER_CWOBJ1606=CWOBJ1606I: {0} - Unable to get response from server {1}. Returning false.
- CWOBJ1607I
  USE_WSADMIN_CWOBJ1607=CWOBJ1607I: {0} - When an ObjectGrid server is colocated with a WebSphere Application Server, use WSADMIN to stop server {1}. Returning false.
- CWOBJ1608I
  SERVER_NOT_RESPONDING_NULL_CWOBJ1608=CWOBJ1608I: {0} - Unable to get response from server {1}. Ensure server is running. Returning null.
- CWOBJ1609I
  NO_ROUTING_TABLE_CWOBJ1609=CWOBJ1609I: {0} - Unable to get routing table. Wait several seconds and retry operation. Returning null.
- CWOBJ1610W
  RESET_NULL_CLUSTER_CWOBJ1610=CWOBJ1610W: Try to reset a null cluster for {0}.
- CWOBJ1616I
  JMX_SECURITY_NOT_FOUND_CWOBJ1616=CWOBJ1616I: JMX Security implementation not found.

- CWOBJ1617E
  NO_JNDI_NAME_SUPPLIED_BIND_CWOBJ1617=CWOBJ1617E: No JNDI name was supplied while doing a bind. The bind will not complete.
- CWOBJ1619E
  NO_JNDI_NAME_SUPPLIED_RESOLVE_CWOBJ1619=CWOBJ1619E: No name was supplied while doing a JNDI resolve.
- CWOBJ1620I
  REPLACE_SERVER_CWOBJ1620=CWOBJ1620I: Replacing target for wrongly routed request due to changes in the server. The new target is {0}.
- CWOBJ1621E
  UNABLE_TO_RESOVLE_JNDI_CWOBJ1621=CWOBJ1621E: Unable to resolve JNDI name {0}.
- CWOBJ1630I
  DOMINO_MODE_CWOBJ1630=CWOBJ1630I: Replication group cannot serve this request {0}.
- CWOBJ1632E
  NULL_ID_CWOBJ1632=CWOBJ1632E: Original request does not have a valid ID; no way to forward this request.
- CWOBJ1634I
  BLIND_FORWARD_CWOBJ1634=CWOBJ1634I: Router cannot find the forwarding target; using blind forwarding.
- CWOBJ1660I
  SERVER_NOT_RIGHT_CWOBJ1660=CWOBJ1660I: Replication group member has changed. This server does not host what is requested anymore. The original request is {0}.
- CWOBJ1663E
  VERIFY_NULL_CLUSTER_CWOBJ1663=CWOBJ1663E: Server router cannot verify server routing for {0}, because cluster data for this replication group are null in the server.
- CWOBJ1668W
  NOT_STARTED_CWOBJ1668=CWOBJ1668W: A client is making a request to a server that has not completed starting.
- CWOBJ1700I
  STANDLAONE_HAMANAGER_INITIALIZED_CWOBJ1700=CWOBJ1700I: Standalone HAManager initialized with coregroup {0}.
- CWOBJ1701I
  STANDLAONE_HAMANAGER_ALREADY_INITIALIZED_CWOBJ1701=CWOBJ1701I: Standalone HAManager is already initialized.
- CWOBJ1702E
  STANDLAONE_HAMANAGER_NOT_INITIALIZED_CWOBJ1702=CWOBJ1702E: Standalone HAManager is not initialized, so it cannot be started.
- CWOBJ1710I
  STANDLAONE_HAMANAGER_STARTED_CWOBJ1710=CWOBJ1710I: Standalone HAManager started successfully.
- CWOBJ1711I
  STANDLAONE_HAMANAGER_ALREADY_STARTED_CWOBJ1711=CWOBJ1711I: Standalone HAManager already started successfully.
- CWOBJ1712E
  STANDLAONE_HAMANAGER_NOT_STARTED_CWOBJ1712=CWOBJ1712E: Standalone HAManager is not started.
- CWOBJ1713E
  STANDLAONE_HAMANAGER_START_FAIL_CWOBJ1713=CWOBJ1713E: Standalone HAManager failed to start.
- CWOBJ1767I
  DCS_SLIDEBAR_SET_CWOBJ1767=CWOBJ1767I: The DCS heartbeating interval is {0}.
- CWOBJ1768I
  DCS_SLIDEBAR_SET_CWOBJ1768=CWOBJ1768I: The DCS heartbeating timeout is {0}.
- CWOBJ1769I
  DCS_SLIDEBAR_SET_CWOBJ1769=CWOBJ1769I: The number of DCS heartbeating threads is {0}.
- CWOBJ1770I
  HA_NOW_COREGROUP_LEADER_CWOBJ1770=CWOBJ1770I: This process is now the core group leader for the {0} core group.
- CWOBJ1771I
  HA_NOT_COREGROUP_LEADER_CWOBJ1771=CWOBJ1771I: This process is no longer the core group leader for the {0} core group.
- CWOBJ1772I
  HA_MEMBERS_CHANGED_CWOBJ1772I=CWOBJ1772I: The high availability (HA) manager and Distribution and Consistency Services (DCS) have notified eXtreme Scale that the list of servers that are running in this core group has changed to {0}.
- CWOBJ1773I
  HA_STANDALONE_DEFINED_SET_IMPORTED_CWOBJ1773I=CWOBJ1773I: This server has received an updated high availabliity (HA) defined set from the catalog server, version {0} with the set now containing the servers: {1}
- CWOBJ1790I
  HAMANAGER_CONTROLLER_NEED_STANDALONE_HAM_CWOBJ1790=CWOBJ1790I: Need to initialize and start the standalone high availability (HA) manager.
- CWOBJ1810I
  CLIENT_FORWARDING_CWOBJ1810=CWOBJ1810I: Forwarding is required for response {0}.
- CWOBJ1811E
  FORWARDING_NOT_FOUND_REQUEST_CWOBJ1811=CWOBJ1811E: Forwarding is required, but the original request cannot be found.
- CWOBJ1870I
  CLIENT_DOMINO_CWOBJ1870=CWOBJ1870I: Server service is not available for response {0}.
- CWOBJ1871E
  NULL_DOMINO_CWOBJ1871=CWOBJ1871E: Detected unavailable service, received null response, no way to retry.
- CWOBJ1872I
  CLIENT_DOMINO_TIMEOUT_CWOBJ1872=CWOBJ1872I: Service is unavailable with response of {0}.
- CWOBJ1890I
  DEAD_SERVER_REROUTING_CWOBJ1890=CWOBJ1890I: Re-routing request {0} due to an un-responsive server.
- CWOBJ1891E
  NO_SERVER_REROUTING_CWOBJ1891=CWOBJ1891E: All servers are not available in replication group {0}.
- CWOBJ1899W
  FORWARD_NULL_RGID_CWOBJ1899=CWOBJ1899W: Forwarding is required, but router cannot find right replication group for response {0}
- CWOBJ1900I
  RPC_SERVICE_INIT_CWOBJ1900=CWOBJ1900I: Client server remote procedure call service is initialized.
- CWOBJ1901I
  RPC_SERVICE_START_CWOBJ1901=CWOBJ1901I: Client server remote procedure call service is started.

- CWOBJ1902I
  RPC_HANDLER_THREADS_START_CWOBJ1902=CWOBJ1902I: Client server remote procedure call handler threads are started.
- CWOBJ1903I
  CONFIG_NETWORK_SERVICE_INIT_CWOBJ1903=CWOBJ1903I: Configuration network service is initialized.
- CWOBJ1904I
  CONFIG_NETWORK_SERVICE_START_CWOBJ1904=CWOBJ1904I: Configuration network service is started.
- CWOBJ1905I
  CONFIG_NETWORK_HANDLER_START_CWOBJ1905=CWOBJ1905I: Configuration handler is started.
- CWOBJ1921W
  Cannot_Find_host_name=CWOBJ1921W: Cannot find host name {0}.
- CWOBJ1922E
  Cannot_Lookup_IP=CWOBJ1922E: Cannot lookup the IP address for this host {0}.
- CWOBJ1929W
  LocalHostUsed=CWOBJ1929W: LOCALHOST is used in the configuration which may lose server identity in multiple machine environment.
- CWOBJ1931I
  ServerSupport=CWOBJ1931I: The configuration for {0} does not support either an ObjectGrid replication group member or a client-server transaction processor. This server will provide bootstrap support to peer ObjectGrid servers and clients only.
- CWOBJ1932I
  ThreadPoolMinMax=CWOBJ1932I: Client thread pool minimum size is {0} maximum size {1}.
- CWOBJ2000E
  NO_RGM_CWOBJ2000=CWOBJ2000E: No member in this replication group {0}.
- CWOBJ2002W
  NO_AVAILABLE_RT_CWOBJ2002=CWOBJ2002W: No available routing table for this replication group {0}.
- CWOBJ2010E
  NULL_TARGET_CWOBJ2010=CWOBJ2010E: Target for this request is null.
- CWOBJ2020I
  ClientProperty_CWOBJ2020=CWOBJ2020I: Client properties are: {0}.
- CWOBJ2024E
  FAILED_TO_LOAD_CLIENT_RETRY_CWOBJ2024E=CWOBJ2024E: Failed to process the value for the client retry property {0}. Value supplied: {1}. The correct value type is an integer up to a long indicating the timeout.
- CWOBJ2060I
  NEW_RT_CHANGE_CWOBJ2060=CWOBJ2060I: Client received new version of replication group cluster {0}.
- CWOBJ2100I
  STALECONN_CWOBJ2100=CWOBJ2100I: Connection ({0}) is stale, it cannot be reused.
- CWOBJ2400E
  INVALID_MAP_SET_CONFIGURATION_CWOBJ2400=CWOBJ2400E: Invalid Configuration: backingMap {0} is a member of more than one mapSet.
- CWOBJ2401E
  BACKING_MAP_WO_MAPSET_CWOBJ2401=CWOBJ2401E: Invalid Configuration: backingMap {0} in distributed ObjectGrid {1} is not in a mapSet.
- CWOBJ2402E
  MAPSET_REF_NONEXISTENT_BMAP_CWOBJ2402=CWOBJ2402E: Invalid Configuration: mapSet has a reference to a {0} map. This backingMap does not exist in the ObjectGrid XML file.
- CWOBJ2403E
  INVALID_XML_FILE_CWOBJ2403=CWOBJ2403E: The XML file is invalid. A problem has been detected with {0} at line {1}. The error message is {2}.
- CWOBJ2404W
  INVALID_CONFIG_VALUE_CWOBJ2404=CWOBJ2404W: The value specified for {0} is {1}. This is an invalid value. {0} will not be set.
- CWOBJ2405E
  OG_BINDING_REF_NONEXISTENT_OG_CWOBJ2405=CWOBJ2405E: The objectgridBinding ref {0} in the cluster XML file does not reference a valid ObjectGrid from the ObjectGrid XML file.
- CWOBJ2406W
  INVALID_XML_FILE_CWOBJ2406=CWOBJ2406W: The XML file is invalid. A problem has been detected with {0} at line {1}. The error message is {2}.
- CWOBJ2407W
  PLUGIN_PROPERTY_INVALID_CWOBJ2407=CWOBJ2407W: The {0} property on the {1} plug-in class could not be set. The exception is {2}.
- CWOBJ2408E
  INVALID_ARGUMENT_CWOBJ2408=CWOBJ2408E: The argument {0} is invalid.
- CWOBJ2409E
  SERVER_STARTUP_EXCEPTION_CWOBJ2409=CWOBJ2409E: The exception {0} occurred during server startup.
- CWOBJ2410E
  ACTIVATION_FAILURE_CWOBJ2410=CWOBJ2410E: The server failed to activate.
- CWOBJ2411E
  INITIALIZATION_FAILURE_CWOBJ2411=CWOBJ2411E: The server failed to initialize.
- CWOBJ2412E
  BOOTSTRAP_FAILURE_CWOBJ2412=CWOBJ2412E: The server failed to bootstrap.
- CWOBJ2413E
  SERVER_STOP_UNSUCCESSFUL_CWOBJ2413=CWOBJ2413E: The attempt to stop the server was unsuccessful.
- CWOBJ2414E
  FORCEFUL_TERMINATION_CWOBJ2414=CWOBJ2414E: The server will be terminated.
- CWOBJ2415I
  SCRIPT_CREATION_CWOBJ2415=CWOBJ2415I: Creating script file {0}.
- CWOBJ2416E
  PLUGIN_INSTANTIATION_ERROR_CWOBJ2416=CWOBJ2416E: Plug-in {0} could not be instantiated and is not configured. The exception is {1}.
- CWOBJ2417W
  DEPRECATED_CLUSTER_XML_ATTRIBUTE_CWOBJ2417=CWOBJ2417W: The {0} attribute on the objectgridBinding element is deprecated in the cluster XML. Use the {0} attribute on the serverDefinition element.
- CWOBJ2418E
  SERVER_LAUNCH_FAILED_CWOBJ2418=CWOBJ2418E: The server failed to launch.

- CWOBJ2419W
  MIN_THREADPOOL_SIZE_WARNING_CWOBJ2419=CWOBJ2419W: The minThreadPoolSize cannot be less than 1. The default value of {0} will be used.
- CWOBJ2420W
  MAX_THREADPOOL_SIZE_WARNING_CWOBJ2420=CWOBJ2420W: The minThreadPoolSize is set to {0} and maxThreadPoolSize is set to {1}. The maxThreadPoolSize must be greater than minThreadPoolSize. The default values will be used: minThreadPoolSize = {2}, maxThreadPoolSize = {3}.
- CWOBJ2421W
  OVERRIDE_WARNING_CWOBJ2421=CWOBJ2421W: The java.util.List supplied to override client-side ObjectGrid settings for cluster {0} contains an element that is not an ObjectGridConfiguration object. This element will be removed from the java.util.List: {1}.
- CWOBJ2422I
  CHECKSUM_DIFFERENCE_CWOBJ2422=CWOBJ2422I: Configuration version on client might not be the same as configuration version used by this server. Client side: host = {0}, , port = {1}, , Server side: host = {2}, port = {3}.
- CWOBJ2423I
  CLIENT_OVERRIDE_URL_CWOBJ2423=CWOBJ2423I: Client-side ObjectGrid settings will be overridden for cluster {0} using the URL {1}.
- CWOBJ2424I
  CLIENT_OVERRIDE_MAP_CWOBJ2424=CWOBJ2424I: Client-side ObjectGrid settings will be overridden for cluster {0} using an entry supplied in the overrideMap.
- CWOBJ2425E
  CLIENT_OVERRIDE_MAP_ERROR_CWOBJ2425=CWOBJ2425E: The Map provided to override client-side ObjectGrid settings for cluster {0} contains a value that is not of type java.util.List. Client-side ObjectGrid settings will not be overridden for this cluster.
- CWOBJ2426E
  CONTAINER_WITHOUT_ZONE_INVALID_CWOBJ2426=CWOBJ2426E: This container has been started without a zone association. This container must be started within a zone since one or more containers in the domain already have been started within one or more zones.
- CWOBJ2427E
  CONTAINER_WITH_ZONE_INVALID_CWOBJ2427=CWOBJ2427E: This container has been started with a zone association. This container must be started without a zone since one or more containers in the domain already have been started without a zone.
- CWOBJ2428W
  ZONE_CONFIG_DEFAULT_INVALID_CWOBJ2428=CWOBJ2428W: The container {0} has started without an association to a zone, but other containers have already started within zones. {0} will be deactivated.
- CWOBJ2429W
  ZONE_CONFIG_CUSTOM_INVALID_CWOBJ2429=CWOBJ2429W: The container {0} started with an association to a zone, but other containers have already started without zone associations. {0} will be deactivated.
- CWOBJ2430E
  ZONE_RULE_TOO_FEW_ZONES_CWOBJ2430=CWOBJ2430E: The zoneRule {0} contains an insufficient number of zones ({1}) for the number of shardMapping entries ({2}) using zoneRule {0}.
- CWOBJ2431E
  MAP_SET_NOT_CONFIGURED_FOR_ZONE_CWOBJ2431=CWOBJ2431E: The container was started in zone {0}, but mapSet {1} for ObjectGrid {2} is not configured to run in zone {0}.
- CWOBJ2432E
  WRONG_NUMBER_SHARD_MAPPINGS_CWOBJ2432=CWOBJ2432E: The wrong number of {0} shardMappings were found for the {1} mapSet in the {2} ObjectGrid. Expected {3} shardMappings, but found {4}.
- CWOBJ2433I
  CLIENT_OVERRIDE_URL_CWOBJ2433=CWOBJ2433I: Client-side ObjectGrid settings will be overridden for domain {0} using the URL {1}.
- CWOBJ2500E
  SERVER_STARTUP_ERROR_CWOBJ2500=CWOBJ2500E: Failed to start ObjectGrid server {0}.
- CWOBJ2501I
  LAUNCHING_SERVER_CWOBJ2501=CWOBJ2501I: Launching ObjectGrid server {0}.
- CWOBJ2502I
  LAUNCHING_SERVER_XML_CWOBJ2502=CWOBJ2502I: Starting ObjectGrid server using ObjectGrid XML file URL {0} and Cluster XML file URL {1}.
- CWOBJ2504I
  SERVER_BOOTSTRAP_LIST_CWOBJ2504=CWOBJ2504I: Attempting to bootstrap to a peer ObjectGrid server using the following host(s) and port(s) {0}.
- CWOBJ2506I
  COMMAND_LINE_TRACE_FILE_CWOBJ2506=CWOBJ2506I: Trace is being logged to {0}.
- CWOBJ2507I
  COMMAND_LINE_TRACE_SPEC_CWOBJ2507=CWOBJ2507I: Trace specification is set to {0}.
- CWOBJ2508I
  LAUNCHING_SERVER_SECURITY_CWOBJ2508=CWOBJ2508I: A security properties file {0} has been specified and will be used to start the server.
- CWOBJ2509E
  SERVER_STARTUP_TIMEOUT_CWOBJ2509=CWOBJ2509E: Timed out after waiting {0} seconds for the server to start.
- CWOBJ2510I
  SERVER_STOP_CWOBJ2510=CWOBJ2510I: Stopping ObjectGrid server {0}.
- CWOBJ2512I
  SERVER_STOPPED_CWOBJ2512=CWOBJ2512I: ObjectGrid server {0} stopped.
- CWOBJ2514I
  SERVER_START_WAITING_CWOBJ2514=CWOBJ2514I: Waiting for ObjectGrid server activation to complete.
- CWOBJ2515E
  INVALID_ARGS_CWOBJ2515=CWOBJ2515E: The arguments provided are invalid. The following are valid arguments: {0}{1}
- CWOBJ2518I
  LAUNCHING_CATALOG_SERVICE_CWOBJ2518I=CWOBJ2518I: Launching ObjectGrid catalog service: {0}.
- CWOBJ2519I
  CWOBJ2519=CWOBJ2519I: The client interceptor has not been registered. Security will not be enabled.
- CWOBJ2520E
  SERVER_NAME_NOT_FOUND_CWOBJ2520=CWOBJ2520E: Server reference for {0} not found in the supplied configuration, please verify provided server name.
- CWOBJ2521I
  CATALOG_CLUSTER_BOOTSTRAP_CHANGED_CWOBJ2521I=CWOBJ2521I: The catalog server bootstrap addresses changed from {0} to {1}.

- CWOBJ2522I
  STATSSPEC_CHANGED_CWOBJ2522I=CWOBJ2522I: The statistics specification has changed to: {0}.
- CWOBJ2601I
  ADD_SUFFIX_TO_VIEW_NAME=CWOBJ2601I: Add suffix {0} to stream query views deployed in partition {1}.
- CWOBJ2602W
  VIEW_TRANSFORMER_EXISTS=CWOBJ2602W: The view transformer {0} already exists.
- CWOBJ2604I
  STREAM_QUERY_JAR_NOT_IN_CLASSPATH=CWOBJ2604I: The stream query jar file is not in the class path.
- CWOBJ2605E
  STREAM_QUERY_LOGGER_ERROR=CWOBJ2605E: The stream query logger setting method introspection or invocation error: {0}
- CWOBJ2606W
  VIEW_REMOVE_NON_EXISTING_ENTRY=CWOBJ2606W: Try to remove a non-existing entry for key {0}.
- CWOBJ2607E
  STREAM_QUERY_SET_ACROSS_MAP_SET=CWOBJ2607E: The stream query set with name {0} contains maps from different map sets.
- CWOBJ2608E
  EXCEEDED_RETRY_UNPROJECT_CWOBJ2608=CWOBJ2608E: Exceeded retry attempts to publish the message, exception: {0}
- CWOBJ2609E
  CONTAINER_SCOPE_PER_CONTAINER_STRATEGY_ERROR=CWOBJ2609E: The combination of container scope and per container strategy were specified for map set {0}.
- CWOBJ2610W
  CONTAINER_SCOPE_REPLICA_WARNING=CWOBJ2610W: A replica setting greater than zero is specified with container scope for map set {0}.
- CWOBJ2611W
  CONTAINER_SCOPE_PARTITION_COUNT_WARNING=CWOBJ2611W: A partition count greater than one was specified on the container scope map set {0}.
- CWOBJ3001I
  EM_SERVICE_STARTED_CWOBJ3001I=CWOBJ3001I: The ObjectGrid EntityManager service is available to process requests for ObjectGrid: {0} and container or server: {1}
- CWOBJ3002I
  EM_INIT_ENTITIES_CWOBJ3002I=CWOBJ3002I: Initializing entity metadata for ObjectGrid: {0}
- CWOBJ3003I
  EM_REGISTERED_CWOBJ3003I=CWOBJ3003I: Entity registered: {0}
- CWOBJ3004E
  EM_REGISTER_EXCEPTION_CWOBJ3004E=CWOBJ3004E: An exception occurred while registering an entity: {0}
- CWOBJ3005I
  EM_CREATING_INDEX_CWOBJ3005I=CWOBJ3005I: Creating index {0} for entity BackingMap {1}, attribute {2}.
- CWOBJ3006E
  EM_UNSUPPORTED_INDEX_TYPE_CWOBJ3006E=CWOBJ3006E: The defined MapIndexPlugin type is not supported for index {0} on BackingMap {1} for attribute {2}.
- CWOBJ3007E
  EM_LATE_REGISTRATION_CWOBJ3007E=CWOBJ3007E: Unable to register new entity {0} after ObjectGrid initialization has completed.
- CWOBJ3008E
  EM_BACKINGMAP_REASSOCIATION_CWOBJ3008E=CWOBJ3008E: BackingMap {0} is associated with entity {1} and cannot be reassociated with entity {2}.
- CWOBJ3009E
  EM_REPOSITORY_EXCEPTION_CWOBJ3009E=CWOBJ3009E: The exception {0} occurred while communicating with the entity metadata repository.
- CWOBJ3010E
  EM_INVALID_MAPSET_CWOBJ3010E=CWOBJ3010E: All entity BackingMaps must be members of a MapSet with the name: "ENTITY_MAPSET".
- CWOBJ3011E
  EM_METADATALISTENER_EXCEPTION_CWOBJ3011E=CWOBJ3011E: Error creating entity metadata for entity {0} ({1}): {2}
- CWOBJ3013E
  EM_MULTIPLE_MAPSETS_CWOBJ3013E=CWOBJ3013E: The EntityMetadata repository is not available. Timeout threshold reached when trying to register the entity: {0}.
- CWOBJ3014I
  AVAILABILITY_STATE_CHANGED_CWOBJ3014=CWOBJ3014I: The availability state has changed for {0}. The state is now {1}. It was previously {2}.
- CWOBJ3015E
  EM_MISSING_MAPSET_CWOBJ3015E=CWOBJ3015E: Invalid entity MapSet configuration. Unable to find MapSet that contains a BackingMap for {0}.
- CWOBJ3016E
  EM_SCHEMA_MAPSET_CROSSOVER_CWOBJ3016E=CWOBJ3016E: Invalid entity MapSet configuration. Entity {0} should be present in MapSet {1} but is already exists in MapSet {2}.
- CWOBJ3017E
  FAILED_TO_VERIFY_ENTITY_CWOBJ3017E=CWOBJ3017E: An entity {0} has been defined in the entity descriptor XML file, but does not have an associated backing map of the same name defined.
- CWOBJ3018E
  FAILED_TO_INIT_ENTITIES_CWOBJ3018E=CWOBJ3018E: Failed to initialize the entities in ObjectGrid {0}.
- CWOBJ3019E
  FAILED_TO_LOAD_OG_CLASSES_CWOBJ3019E=CWOBJ3019E: The class {0} cannot be found for ObjectGrid {1}.
- CWOBJ3101E
  WB_LOADER_INITIALIZATION_FAILED_CWOBJ3101E=CWOBJ3101E: The write-behind loader for map {0} on partition {1} failed to initialize with exception {2}.
- CWOBJ3102E
  WB_LOADER_FAILED_CWOBJ3102E=CWOBJ3102E: Loader failed to do a write-behind update to the database for map {0} on partition {1}. A failed update is logged in the failed update map. The failed update index is {2}, and the failed map key is {3}. The exception causing the failed update was {4}.
- CWOBJ3103E
  WB_LOADER_FAILED_CWOBJ3103E=CWOBJ3103E: The write-behind loader for map {0} on partition {1} failed to complete a transaction. The exception is {2}.

- CWOBJ3104W
  WB_LOADER_LOCKTIMEOUT_CWOBJ3104W=CWOBJ3104W: The write-behind loader for map {0} on partition {1} gets a lock timeout exception, {2}, when trying to switch the queue maps.
- CWOBJ3105E
  WB_LOADER_LOADER_NOT_AVAILABLE_CWOBJ3105E=CWOBJ3105E: The write-behind loader for ObjectGrid {0}, map {1} on partition {2} received an {3} error.
- CWOBJ3106W
  WB_LOADER_LONG_TRAN_CWOBJ3106W=CWOBJ3106W: The write-behind loader for ObjectGrid {0}, map {1} on partition {2} {3} a {4} ms long transaction when removing the data from the queue map and batch updating to the loader. Within this eXtreme Scale transaction, the loader batch update takes {5} ms. The possible causes are: 1) The data store backend cannot keep up. Considering tuning database and using connection pool. 2) The write-behind parameter update count is too big or update time is too long. Consider decreasing the write-behind parameter value.
- CWOBJ3107W
  WB_LOADER_SMALL_TRAN_TIMEOUT_CWOBJ3107W=CWOBJ3107W: The write-behind loader for ObjectGrid {0}, map {1} on partition {2} {3} a {4} ms long transaction, which is approaching to the transaction timeout value {5} ms. Within this eXtreme Scale transaction, the loader batch update takes {6} ms. The transaction timeout value is probably too small. Consider increasing the transction timeout value.
- CWOBJ3108E
  WB_LOADER_REPLICA_UNAVAILABLE_CWOBJ3108E=CWOBJ3108E: The write-behind loader of ObjectGrid {0}, map {1} on partition {2} received a ReplicationVotedToRollbackTransactionException: {3}
- CWOBJ3111E
  CLIENT_LOADER_AGENT_FAIL_CWOBJ3111E=CWOBJ3111E: The client loader agent {0} execution fails with exception: {1}.
- CWOBJ3112I
  DEFAULT_PERSISTENCE_UNIT_CWOBJ3112I=CWOBJ3112I: A JPA persistence unit name was not specified. The first persistence unit {0} defined in the persistence.xml will used as the default persistence unit.
- CWOBJ3113E
  AGENT_FAIL_CWOBJ3113E=CWOBJ3113E: The DataGrid agent {0} execution failed with a exception {1}.
- CWOBJ3114E
  AGENT_FAIL_RETRYABLE_CWOBJ3114E=CWOBJ3114E: The DataGrid agent {0} execution failed with a retryable exception {1}.
- CWOBJ3115E
  UNEXPECTED_SHARD_STATE_CWOBJ3115E=CWOBJ3115E: The shard is expected to be in the {0} state, but currently it is in the {1} state. If you have already set the shard to the {0} state, it might take a while for it to move to the target state. If you have not set the shard to the {0} state, revise your application to do so.
- CWOBJ3116I
  PRELOAD_STARTS_CWOBJ3116I=CWOBJ3116I: Preloading ObjectGrid {0} Map {1} at partition {2} started.
- CWOBJ3117I
  PRELOAD_FINISHES_CWOBJ3117I=CWOBJ3117I: Preloading ObjectGrid {0} Map {1} at partition {2} finished.
- CWOBJ3118E
  PRELOAD_FAILS_CWOBJ3118E=CWOBJ3118E: Failed to preload ObjectGrid {0} Map {1} at partition {2} with the exception {3}.
- CWOBJ3121E
  TIME_BASED_DBUPDATE_AGENT_FAIL_CWOBJ3121E=CWOBJ3121E: The time-based database update agent fails with exception {0}.
- CWOBJ3122E
  TIME_BASED_DBUPDATE_FAIL_CWOBJ3122E=CWOBJ3122E: The time-based database update fails with exception {0}.
- CWOBJ3131E
  JPA_TX_CALLBACK_NOT_FOUND_CWOBJ3131E=CWOBJ3131E: The JPATxCallback transaction callback plug-in cannot be found.
- CWOBJ3132E
  EMF_NOT_FOUND_CWOBJ3132E=CWOBJ3132E: The JPA EntityManagerFactory with persistence unit name {0} and property map {1} could not be found.
- CWOBJ3133E
  OBJECTGRID_CACHE_INITIALIZE_OBJECTGRID_FAILED_CWOBJ3133E=CWOBJ3133E: ObjectGrid cache plug-in initialization with ObjectGrid {1} failed with exception {0}.
- CWOBJ3134I
  OBJECTGRID_CACHE_TYPE_EMBEDDED_CWOBJ3134I=CWOBJ3134I: The ObjectGrid type is {0} and the default maximum number of replicas is {1}.
- CWOBJ3135I
  OBJECTGRID_CACHE_TYPE_EMBEDDED_PARTITION_CWOBJ3135I=CWOBJ3135I: The ObjectGridType is {0} and the default number of partitions is {1}. The number of partitions must be less than or equal to the number of JVMs in the system.
- CWOBJ3136I
  OBJECTGRID_CACHE_TYPE_EMBEDDED_INTERDOMAIN_CWOBJ3136I=CWOBJ3136I: The ObjectGrid type is {0}. The placement scope is {1} and the scope topology is {2}.
- CWOBJ3141W
  NODEGROUP_NOT_SET_FOR_ZONE_SUPPORT_CWOBJ3141W=CWOBJ3141W: This WebSphere Application Server is not associated with a WebSphere eXtreme Scale zone. In order to start the server in a zone, ensure that the server's node is within a node group whose name begins with the string ReplicationZone.
- CWOBJ3142I
  NODEGROUP_NOT_SET_FOR_ZONE_SUPPORT_CWOBJ3142I=CWOBJ3142I: This WebSphere Application Server is not associated with a WebSphere eXtreme Scale zone. In order to start the server in a zone, ensure that the server's node is within a node group whose name begins with the string ReplicationZone.
- CWOBJ3150E
  CLEAR_TIMED_OUT_CWOBJ3150=CWOBJ3150E: The clear operation timed-out after {0} ms.
- CWOBJ3175E
  FAILED_TO_REGISTER_BEAN_FACTORY_CWOBJ3175E=CWOBJ3175E: Exception {2} occurred when registering Spring bean factory {0} with the ObjectGrid {1}.
- CWOBJ3176E
  FAILED_TO_GET_BEAN_FACTORY_CWOBJ3176E=CWOBJ3176E: Exception {1} occurred when loading Spring bean factory with the ObjectGrid {0}.
- CWOBJ3177E
  FAILED_TO_LOCATE_OBJECTGRID_XML_FILE_CWOBJ3177E=CWOBJ3177E: Failed to locate the ObjectGrid XML file: {0}.
- CWOBJ3178E
  BACKING_MAP_NOT_FOUND_IN_OBJECTGRID_XML_CWOBJ3178E=CWOBJ3178E: The map {1} in ObjectGrid {0} referenced in the ObjectGrid XML was

not found in the deployment descriptor file.

- CWOBJ3179E
  INVALID_BACKING_MAP_IN_DEPLOYMENT_XML_CWOBJ3179E=CWOBJ3179E: The map {0} reference in the mapSet {1} of ObjectGrid {2} deployment descriptor file does not reference a valid backing map from the ObjectGrid XML.
- CWOBJ3180E
  INVALID_OBJECTGRID_IN_DEPLOYMENT_XML_CWOBJ3180E=CWOBJ3180E: The ObjectGrid {0} specified in the deployment descriptor file is not defined in the ObjectGrid XML file.
- CWOBJ3181E
  INVALID_SERVER_SECURITY_FILE_OPTION_CWOBJ3181E=CWOBJ3181E: The command-line option -serverSecurityFile is invalid for ObjectGrid container servers.
- CWOBJ3182E
  XERCES_IMPLEMENTATION_NOT_IN_CLASSPATH_CWOBJ3182E=CWOBJ3182E: Apache Xerces2 was not found in the classpath.
- CWOBJ3183W
  CONTAINER_PLACEMENT_SCOPE_IN_DEPLOYMENT_XML_IGNORE_REPLICA_CWOBJ3183W=CWOBJ3183W: When the container placement scope of CONTAINER_SCOPE setting is specified, any replica setting must be zero.
- CWOBJ3184W
  CONTAINER_PLACEMENT_SCOPE_IN_DEPLOYMENT_XML_NON_ONE_PARTITION_COUNT_CWOBJ3184W=CWOBJ3184W: When the container placement scope of CONTAINER_SCOPE setting is specified, the number of partitions setting must be one.
- CWOBJ3185E
  INVALID_PLACEMENT_SCOPE_IN_DEPLOYMENT_XML_CWOBJ3185E=CWOBJ3185E: The placement strategy of per container and the container placement scope of CONTAINER_SCOPE can not be used together.
- CWOBJ3186I
  CONTAINER_PLACEMENT_SCOPE_IN_DEPLOYMENT_XML_DEFAULT_COLLISION_ARBITER_CWOBJ3186I=CWOBJ3186I: No custom COLLISION_ARBITER is defined. The default arbitration will be use.
- CWOBJ3187E
  ERROR_IN_ARBITER_CWOBJ3187E=CWOBJ3187E: The collision arbiter implementation on {1} generate exception, {0}, which will result in a halt to replication.
- CWOBJ3188E
  CONTAINER_PLACEMENT_SCOPE_IN_DOWNLEVEL_CONTAINER_CWOBJ3188E=CWOBJ3188E: A map set with a container placement scope of CONTAINER_SCOPE can not be deployed to a pre 7.1.1 WebSphere eXtreme Scal container.
- CWOBJ3189I
  CONTAINER_PLACEMENT_SCOPE_HUB_CONTAINER_CWOBJ3189I=CWOBJ3189I: The hub container for the container scope placement scope map set {1} is container {0}.
- CWOBJ3190E
  INVALID_CONTAINER_PLACEMENT_SCOPE_IN_DEPLOYMENT_XML_CWOBJ3190E=CWOBJ3190E: When the container placement scope of CONTAINER_SCOPE setting is specified in the deployment descriptor file, the Loader class in the object grid file can not be specified.
- CWOBJ4000I
  RESTSERVICE_STARTED_CWOBJ4000I=CWOBJ4000I: The WebSphere eXtreme Scale REST data service has been started.
- CWOBJ4001I
  RESTSERVICE_VERSION_CWOBJ4001I=CWOBJ4001I: The WebSphere eXtreme Scale REST data service version is {0}.
- CWOBJ4002E
  RESTSERVICE_STARTUPFAILURE_CWOBJ4002E=CWOBJ4002E: The WebSphere eXtreme Scale REST data service could not be started.
- CWOBJ4003E
  RESTSERVICE_MISSINGCATALOGSERVICE_CWOBJ4003E=CWOBJ4003E: Unable to connect to the catalog service. The catalog service endpoints were not specified.
- CWOBJ4004I
  RESTSERVICE_PROPSLOADED_CWOBJ4004I=CWOBJ4004I: The eXtreme Scale REST data service properties files were loaded: {0}
- CWOBJ4005E
  RESTSERVICE_MISSINGCLIENTOGXML_CWOBJ4005E=CWOBJ4005E: The client ObjectGrid descriptor XML file "{0}" could not be found in the classpath.
- CWOBJ4006I
  RESTSERVICE_CONNECTIONENDPOINTS_CWOBJ4006I=CWOBJ4006I: Connecting to eXtreme Scale catalog service endpoints: {0}
- CWOBJ4007E
  RESTSERVICE_CONNECTFAILURE_CWOBJ4007E=CWOBJ4007E: The WebSphere eXtreme Scale REST data service was unable to connect to the eXtreme Scale grid: {0}
- CWOBJ4008W
  RESTSERVICE_ATOM_WRONG_ELEMENT_NAMESPACE_WARNING_CWOBJ4008W=CWOBJ4008W: The XML element "{0}" specified in the AtomPub format XML has a wrong namespace prefix "{1}". The valid namespace prefix should be resolved to "{2}".
- CWOBJ4010I
  RESTSERVICE_GRIDSAVAILABLE_CWOBJ4010I=CWOBJ4010I: The following ObjectGrids can now be accessed from the REST data service: {0}
- CWOBJ4011E
  RESTSERVICE_METADATA_INVALID_CWOBJ4011E=CWOBJ4011E: The entity metadata for ObjectGrid "{0}" is configured incorrectly.
- CWOBJ4012E
  RESTSERVICE_METADATA_INVALID_UNIASSOC_CWOBJ4012E=CWOBJ4012E: The association "{0}" defined for entity "{1}" is not mapped to a target association on entity "{2}". All associations must be bi-directional and have the mapped-by attribute defined.
- CWOBJ4013E
  RESTSERVICE_METADATA_AUTOGEN_KEY_COLLISION_CWOBJ4013E=CWOBJ4013E: An automatically generated key association name resulted in a duplicate attribute "{0}" for entity "{1}".
- CWOBJ4014E
  RESTSERVICE_METADATA_NOROOTPATH_CWOBJ4014E=CWOBJ4014E: The partitioned entity "{0}" must be defined as a schema root or must have a key relationship to the schema root.
- CWOBJ4015E
  RESTSERVICE_METADATA_INVALIDNAME_ATTR_CWOBJ4015E=CWOBJ4015E: The attribute name "{0}" is invalid for entity "{1}". Attributes cannot begin with characters: $_
- CWOBJ4016E
  RESTSERVICE_METADATA_INVALIDNAME_ENTITY_CWOBJ4016E=CWOBJ4016E: The entity name "{0}" is invalid. Entity names cannot begin with

characters: $_
- CWOBJ4017E
  RESTSERVICE_INCOMPATIBLE_VERSION_CWOBJ4017E=CWOBJ4017E: Version {0} of WebSphere eXtreme Scale runtime is incompatible with version {1} of the REST data service.
- CWOBJ4018E
  RESTSERVICE_METADATA_INVALID_CASCADE_REMOVE_CWOBJ4018E=CWOBJ4018E: The association "{0}" is is invalid for entity "{1}". Many-to-one and many-to-many associations cannot be configured to cascade remove operations.
- CWOBJ4019E
  RESTSERVICE_METADATA_INVALID_CASCADE_MULTI_CWOBJ4019E=CWOBJ4019E: The association "{0}" for entity "{1}" and the association "{2}" for entity "{3}" are invalid. A cascade remove can only be configured on one end of a bi-directional association.
- CWOBJ4020E
  RESTSERVICE_CONFIG_INVALID_OBJECTGRID_NAME_CWOBJ4020E=CWOBJ4020E: The "{0}" property in the REST service properties file contains an incorrect value. At least one ObjectGrid name must be specified.
- CWOBJ4021E
  RESTSERVICE_CONFIG_INVALID_PROPERTY_FILE_CWOBJ4021E=CWOBJ4021E: The REST service properties file "{0}" was not found on the file system or the class path.
- CWOBJ4022E
  RESTSERVICE_CONFIG_INVALID_OBJECT_GRID_NAME_CWOBJ4022E=CWOBJ4022E: The ObjectGrid "{0}" does not exist or is not started and will not be exposed via the REST service.
- CWOBJ4023E
  RESTSERVICE_CONFIG_EXCEP_CLIENT_OBJECT_GRID_XML_CWOBJ4023E=CWOBJ4023E: The exception "{0}" was encountered when loading the client ObjectGrid override XML file "{1}" from the class path.
- CWOBJ4024E
  RESTSERVICE_CLIENT_SECURITY_CONFIG_GEN_PROPS_CWOBJ4024E=CWOBJ4024E: The eXtreme Scale REST service is configured to use ObjectGrid client security but the "credentialGeneratorProps" property is not defined in the "{0}" file.
- CWOBJ4025E
  RESTSERVICE_REST_SECURITY_LOGINTYPE_CWOBJ4025E=CWOBJ4025E: The eXtreme Scale REST service is configured to use REST security with an incorrect "{0}" property of "{1}". A "{0}" of "{2}" will be used.
- CWOBJ4026E
  RESTSERVICE_REST_SECURITY_INCORRECT_MAXRESULTS_CWOBJ4026E=CWOBJ4026E: The eXtreme Scale REST service "maxResultsForCollection" config property has an incorrect value of"{0}". The default value of unlimited will be used.
- CWOBJ4027W
  RESTSERVICE_MBEAN_INCORRECT_OPERATION_PARAM_CWOBJ4027W=CWOBJ4027W: The eXtreme Scale REST service MBean operation "{0}" was invoked with an incorrect paramter of "{1}". The current value will be used.
- CWOBJ4028I
  RESTSERVICE_MBEAN_TRACE_DYNAMIC_CWOBJ4028I=CWOBJ4028I: The eXtreme Scale REST service debug tracing was set to "{0}" dynamically.
- CWOBJ4029W
  RESTSERVICE_MBEAN_INCORRECT_TRACESPEC_CWOBJ4029W=CWOBJ4029W: The eXtreme Scale REST service MBean operation "{0}" was invoked with an incorrect paramter of "{1}". The current value will be used.
- CWOBJ4030W
  RESTSERVICE_CONFIG_MALFORMED_LINE_CWOBJ4030W=CWOBJ4030W: The malformed line: "{0}" was encountered when loading the REST service properties file: "{1}".
- CWOBJ4031W
  RESTSERVICE_CONFIG_INVALID_VALUE_CWOBJ4031W=CWOBJ4031W: Invalid value: "{0}" was encountered on line: "{1}" when loading the rest service properties file: "{2}". Expected value is: "{3}".
- CWOBJ4032E
  RESTSERVICE_CONFIG_EXCEPTION_PARSING_FILE_CWOBJ4032E=CWOBJ4032E: The exception: "{0}" was encountered when loading the REST service properties file: "{1}".
- CWOBJ4033W
  RESTSERVICE_CONFIG_INVALID_GRIDNAME_CWOBJ4033W=CWOBJ4033W: Invalid grid name: "{0}" was encountered on line: "{1}" when loading the rest service properties file: "{2}". Please specify name of an existing ObjectGrid.
- CWOBJ4034W
  RESTSERVICE_CONFIG_INVALID_ENTITYNAME_CWOBJ4034W=CWOBJ4034W: Invalid entity name: "{0}" was encountered on line: "{1}" when loading the rest service properties file: "{2}". Please specify name of an existing entity within ObjectGrid: "{3}".
- CWOBJ4500I
  DYNACACHE_PROVIDER_INITIALIZED_CWOBJ4500=CWOBJ4500I: WebSphere eXtreme Scale Dynamic Cache provider is successfully initialized.
- CWOBJ4501E
  DYNACACHE_CREATION_FAILURE_CWOBJ4501=CWOBJ4501E: The WebSphere eXtreme Scale Dynamic Cache provider encountered an error while creating the following cache instance: {0}.
- CWOBJ4502E
  MISSING_REQUIRED_CONFIGURATION_PARAMETER_CWOBJ4502=CWOBJ4502E: Missing the following required configuration parameter: {0}.
- CWOBJ4503E
  DYNACACHE_PROVIDER_FAILED_INIT_CWOBJ4503=CWOBJ4503E: WebSphere eXtreme Scale Dynamic Cache provider failed to initialize successfully.
- CWOBJ4504W
  DYNACACHE_UNEXPECTED_SPECIAL_VALUE_CWOBJ4504=CWOBJ4504W: Cache Entry is tagged as a Special Value. Value is being ignored.
- CWOBJ4505W
  DYNACACHE_CONFIG_MISMATCH_CWOBJ4505=CWOBJ4505W: Dynamic Cache configuration sent from provider does not match currently stored configuration for cache {0}. Stored configuration is {1}. Received configuration is {2}.
- CWOBJ4506I
  DYNACACHE_EVICTOR_FAILOVER_CWOBJ4506=CWOBJ4506I: Configuration found in map. ObjectGrid shard is becoming primary after a failover. Setting Dynamic Cache Evictor configuration. Configuration: {0}
- CWOBJ4507E
  DYNACACHE_INCORRECT_PARAMETER_FORMAT_CWOBJ4507=CWOBJ4507E: The value {1} set for an optional configuration parameter {0} is invalid.
- CWOBJ4508I
  DYNACACHE_CREATED_CWOBJ4508=CWOBJ4508I: The WebSphere eXtreme Scale provider has created a Dynamic Cache instance with name {0} using

topology {1}.
- CWOBJ4509E
  DYNACACHE_UNSUPPORTED_REPLICATION_POLICY_CWOBJ4509=CWOBJ4509E: The WebSphere eXtreme Scale Dynamic Cache provider does not support the {0} replication policy for Cache {1} with key {2}.
- CWOBJ4510E
  DYNACACHE_REQUIRES_SERVER_CWOBJ4510=CWOBJ4510E: WebSphere eXtreme Scale Server is required to create Dynamic Cache instances with topology {0}. Cache name is {1}.
- CWOBJ4511E
  DYNACACHE_GRID_DISCONNECTED_CWOBJ4511=CWOBJ4511E: The WebSphere eXtreme Scale Dynamic Cache provider has become disconnected from {0} WebSphere eXtreme Scale grid and {1} map: {2}
- CWOBJ4512I
  DYNACACHE_GRID_RECONNECTED_CWOBJ4512=CWOBJ4512I: The WebSphere eXtreme Scale Dynamic Cache provider has reconnected with {0} WebSphere eXtreme Scale grid and {1} map.
- CWOBJ4541I
  MEMORYSTATS_ENHANCED_SIZING_IN_USE_CWOBJ4541=CWOBJ4541I: Enhanced BackingMap memory sizing is enabled.
- CWOBJ4542I
  MEMORYSTATS_DEFAULT_SIZING_IN_USE_CWOBJ4542=CWOBJ4542I: Basic BackingMap memory sizing is enabled.
- CWOBJ4543W
  MEMORYSTATS_OBJECT_TOO_COMPLEX_CWOBJ4543=CWOBJ4543W: The size of an object of type {0} is not accurate.
- CWOBJ4551E
  EVICTION_TRIGGER_NOT_SUPPORTED_CWOBJ4551=CWOBJ4551E: The eviction trigger {0} cannot be used with the current Java Virtual Machine configuration {1}.
- CWOBJ4552W
  EVICTION_TRIGGER_NOT_STABLE_CWOBJ4552=CWOBJ4552W: The eviction trigger {0} might not behave as expected when used with the Java Virtual Machine setting {1}.
- CWOBJ4560W
  QUERY_CACHE_MAX_ENTRIES_CWOBJ4560=CWOBJ4560W: The query queue cache of ObjectGrid {0} reached the maximum size of {1}. Eviction of the query queues will occur based on the Least Recently Used rule. This message will only be logged for the first eviction.
- CWOBJ4561W
  QUERY_CACHE_MAX_ENTRIES_CWOBJ4561=CWOBJ4561W: The query queue cache of ObjectGrid {0} for partition {1} reached the maximum size of {2}. Eviction of the query queues will occur based on the Least Recently Used rule. This message will only be logged for the first eviction.
- CWOBJ4600E
  GET_ATTRIBUTES_EXCEPTION_CWOBJ4600=CWOBJ4600E: Exception {1} occurred on the getAttribute for {0}. Continuing to create attribute list.
- CWOBJ4601E
  SET_ATTRIBUTES_EXCEPTION_CWOBJ4601=CWOBJ4601E: Exception {1} occurred on the setAttribute for {0}. Continuing to set other attributes.
- CWOBJ4650I
  DISK_EVICTOR_DETECTED_CWOBJ4650=CWOBJ4650I: The Evictor is using disk based persistance at the following URI {0}.
- CWOBJ4651W
  DISK_OFFLOAD_CWOBJ4651=CWOBJ4651W: Persistence directory {0} already exists but does not contain valid data. It will be cleared.
- CWOBJ4652W
  DISK_OFFLOAD_CWOBJ4652=CWOBJ4652W: Persistence directory {0} cannot be opened because it is in use by another process.
- CWOBJ4700I
  DYNAMIC_MAP_CREATED_CWOBJ4700=CWOBJ4700I: The map name {0} matched the regular expression of template map {1}. The {0} map has been created for ObjectGrid {2}.
- CWOBJ4701I
  TEMPLATE_MAP_CONFIGURED_CWOBJ4701=CWOBJ4701I: Template map {0} is configured in ObjectGrid {1}.
- CWOBJ4702E
  DYNAMIC_MAP_CREATION_ERROR_CWOBJ4702=CWOBJ4702E: Dynamic creation failed for map {0} due to the following exception {1}.
- CWOBJ4800E
  SHARD_ALREADY_RESERVED_ERROR_CWOBJ4800=CWOBJ4800E: Could not reserve shard {0} on container {1} because this shard is already reserved by container {2}.
- CWOBJ4801W
  PARTITION_NOT_FOUND_CWOBJ4801=CWOBJ4801W: No shard for partition {0} was released from container {1} because this container has not reserved a shard from this partition.
- CWOBJ4802E
  RESERVE_PARTITION_NON_EXISTENT_CWOBJ4802=CWOBJ4802E: Attempt to reserve shard {0} on container {1} has failed because the partition does not exist.
- CWOBJ4803E
  PER_CONTAINER_UNSUPPORTED_CWOBJ4803=CWOBJ4803E: The shard reservation feature is not supported with PER_CONTAINER placement strategy or scope.
- CWOBJ4804I
  SUCCESSFUL_SHARD_RESERVATION_CWOBJ4804=CWOBJ4804I: Shard {0} was successfully reserved on container {1}.
- CWOBJ4805I
  SUCCESSFUL_SHARD_RELEASE_CWOBJ4805=CWOBJ4805I: Shard from partition {0} was successfully released from container {1}.
- CWOBJ4806I
  SHARD_RESERVED_PRIOR_INIT_PLACEMENT_CWOBJ4806=CWOBJ4806I: Shard {0} has been reserved on container {1} prior to initial placement. Shard will be placed onto this container when initial placement occurs.
- CWOBJ4807E
  RESERVE_CONTAINER_NOT_SUPPORTING_MAPSET_CWOBJ4807=CWOBJ4807E: Shard {0} cannot be reserved on container {1} because this container does not support map set {2}.
- CWOBJ4808I
  ROLE_SWAP_SUCCESSFUL_CWOBJ4808=CWOBJ4808I: Request from shard {0} to swap roles with a(n) {1} shard was processed successfully. This shard is now a(n) {1}.
- CWOBJ4809W
  ROLE_SWAP_SAME_TYPE_CWOBJ4809=CWOBJ4809W: Request from shard {0} to swap roles with a(n) {1} shard failed to execute because this shard is

already a(n) {1}
- CWOBJ4810E
  ROLE_SWAP_SHARD_TYPE_NOT_PLACED_CWOBJ4810=CWOBJ4810E: Request from shard {0} to swap roles with a(n) {1} shard has failed because no {1} from this partition is currently placed.
- CWOBJ4811E
  ROLE_SWAP_SHARD_TIMEOUT_CWOBJ4811=CWOBJ4811E: Request from shard {0} to swap roles with a(n) {1} shard has timed out. Shard {0} did not inherit its new role in the allotted time.
- CWOBJ4812E
  ROLE_SWAP_SHARD_NOT_FOUND_ON_CONTAINER_CWOBJ4812=CWOBJ4812E: Request from shard {0} to swap roles with the {1} shard on container {2} has failed. No {1} shard was found on the specified container for this partition.
- CWOBJ4813E
  ROLE_SWAP_INVALID_CONTAINER_CWOBJ4813=CWOBJ4813E: No container was found to match the name {2}. Request from shard {0} to swap roles with the {1} shard on container {2} has failed.
- CWOBJ4814E
  ROLE_SWAP_PER_CONTAINER_SCOPE_NOT_SUPPROTED_CWOBJ4814=CWOBJ4814E: The shard {0} has a placement scope of per container.
- CWOBJ4900E
  MAPSET_INCOMPATIBLE_PARTITION_NUM_CWOBJ4900=CWOBJ4900E: Domain {0} will not send updates to domain {1} for map set {2} from ObjectGrid {3} because of a mismatch in the number of partitions. The map set is configured for {4} partitions in domain {0} and {5} partitions in domain {1}.
- CWOBJ4901E
  MAPSET_INCOMPATIBLE_PLACEMENT_STRAT_CWOBJ4901=CWOBJ4901E: Domain {0} will not send updates to domain {1} for map set {2} from ObjectGrid {3} because of a mismatch in the placement strategy. The map set is configured for {4} placement strategy in domain {0} and {5} placement strategy in domain {1}.
- CWOBJ4902I
  MAPSET_COMPATIBLE_CWOBJ4902=CWOBJ4902I: This domain ({0}) received a compatible map set from domain {1}. Updates for map set {2} from ObjectGrid {3} will be sent to domain {1}.
- CWOBJ4903I
  FOREIGN_DOMAINS_FOUND_CWOBJ4903=CWOBJ4903I: The following foreign domains have been provided: {0}
- CWOBJ4904I
  FOREIGN_DOMAIN_END_POINTS_CWOBJ4904=CWOBJ4904I: The following end points have been provided for foreign domain {0}: {1}
- CWOBJ4905E
  MAPSET_WRONG_NUM_MAPS_CWOBJ4905=CWOBJ4905E: {0} from linked domains do not contain the same maps. While domain {1} contains the following maps for this map set: {2}, domain {3} contains: {4}.
- CWOBJ4906E
  MAPSET_MISSING_MAP_CWOBJ4906=CWOBJ4906E: Domain {0} will not send updates to domain {1} for map set {2} from ObjectGrid {3}. The maps in the map set are not consistent. The {4} map was found in the map set for domain {5}, but not for domain {6}.
- CWOBJ4907E
  FOREIGN_ENDPOINTS_NOT_FOUND_CWOBJ4907=CWOBJ4907E: No end points were provided for {0} foreign domain, which was expecting {2} property. No attempt will be made to establish a link between the {1} and {0} domains.
- CWOBJ4908E
  LOCAL_DOMAIN_INCLUDED_IN_FOREIGN_CWOBJ4908=CWOBJ4908E: The local domain name {0} was found in the set of foreign domains in the server properties.
- CWOBJ4909E
  WRITE_BEHIND_MAP_FOUND_CWOBJ4909=CWOBJ4909E: Domain {0} will not send updates to domain {1} for map set {2} from ObjectGrid {3} because the {4} map is configued for write-behind support in {5}.
- CWOBJ4910E
  COPY_TO_BYTES_FOUND_CWOBJ4910=CWOBJ4910E: Domain {0} will not send updates to domain {1} for {2}:{3} because the {4} map is configured as a byte array map in {5}.
- CWOBJ4911E
  ENTITY_MAP_FOUND_CWOBJ4911=CWOBJ4911E: Domain {0} will not send updates to domain {1} for {2}:{3} because the {4} map is backing an entity in {5}.
- CWOBJ4912E
  KEYTYPE_BYTES_FOUND_CWOBJ4912=CWOBJ4912E: The {0} local domain will not send updates to the {1} foreign domain for {2}:{3} because the {4} map is configured as a bytes-for-keys map in the {5} domain that contains the bytes for keys.
- CWOBJ4913I
  FOREIGN_DOMAIN_NOT_AVAILABLE_CWOBJ4913=CWOBJ4913I: During an attempt to {0} the foreign catalog service for foreign domain {1} could not be reached. The procedure completed in the local domain but was not completed in the foreign domain.
- CWOBJ4914W
  DOMAIN_PING_FAILURE_CWOBJ4914=CWOBJ4914W: Attempt to ping foreign domain, {0}, failed. The ping will be attempted again in {1} milliseconds.
- CWOBJ4915I
  DOMAIN_PING_SUCCESS_CWOBJ4915=CWOBJ4915I: This domain successfully pinged the foreign domain, {0}.
- CWOBJ4916I
  FOREIGN_DOMAIN_RECYCLED_CWOBJ4916=CWOBJ4916I: The local domain detected that domain {0} has been restarted after being unavailable for some time.
- CWOBJ4917I
  DOMAIN_PING_SUCCESSFUL_AFTER_UNSUCCESSFUL_CWOBJ4917=CWOBJ4917I: This domain successfully pinged the foreign domain, {0}, after {1} consecutive unsuccessful attempts.
- CWOBJ5000I
  RA_CONNECTED_CWOBJ5000I=CWOBJ5000I: The WebSphere eXtreme Scale resource adapter connected to the {0} grid at the following catalog server endpoints: {1}
- CWOBJ5001I
  RA_DISCONNECTED_CWOBJ5001I=CWOBJ5001I: The WebSphere eXtreme Scale resource adapter disconnected from the {0} grid at the following catalog server endpoints: {1}
- CWOBJ6400I
  OSGI_NEW_SERVICE_ADDED_CWOBJ6400=CWOBJ6400I: The {0} OSGi service with the service ranking {1} from the {2} service ID is available.

- CWOBJ6401I
  OSGI_SERVICE_REMOVED_CWOBJ6401=CWOBJ6401I: The {0} OSGi service with service ranking {1} from the {2} service ID is no longer available.
- CWOBJ6402W
  OSGI_SERVICE_NOT_FOUND_CWOBJ6402=CWOBJ6402W: The {0} OSGi service with service ranking {1} from the {2} service ID cannot be found in the eXtreme Scale runtime environment.
- CWOBJ6403I
  OSGI_SERVICE_USED_CWOBJ6403=CWOBJ6403I: The {0} OSGi service with service ranking {1} from the {2} service ID is used by the eXtreme Scale runtime.
- CWOBJ6404I
  OSGI_SERVICE_ALREADY_USED_CWOBJ6404=CWOBJ6404I: The {0} OSGi service with service ranking {1} has already been used. The service ID is {2}.
- CWOBJ6405I
  OSGI_BUNDLE_ACTIVATED_CWOBJ6405=CWOBJ6405I: The eXtreme Scale OSGi bundle with the {0} symbolic name is activated.
- CWOBJ6406I
  OSGI_BUNDLE_STOPPED_CWOBJ6406=CWOBJ6406I: The eXtreme Scale OSGi bundle with the {0} symbolic name is stopped.
- CWOBJ6407W
  OSGI_SERVICE_UPGRADE_WARNING_CWOBJ6407=CWOBJ6407W: The OSGi service upgrade did not find a client identifier for the {0} data grid.
- CWOBJ6408W
  SHARD_SCOPE_THREADLOCAL_WARNING_CWOBJ6408=CWOBJ6408W: In the Spring framework, the thread local {0} value for the custom shard scope is not null. It is {1}.
- CWOBJ6409W
  SERVICE_DESTROY_FAILED_CWOBJ6409=CWOBJ6409W: When a new OSGi service is used, the destroy() call on the old service instance {0} throws an exception with the following message: {1}
- CWOBJ6410E
  SERVICE_UPDATE_FAILED_CWOBJ6410=CWOBJ6410E: The update for the {0} OSGi service of the {1} ObjectGrid to service ranking {2} failed. The failure is logged and ignored. Error: {3}
- CWOBJ6411W
  REPOSITORY_SERVICE_RANKING_USED_CWOBJ6411=CWOBJ6411W: For the {0} ObjectGrid, the OSGi metadata repository is currently using service ranking {1} for service {2}, which is not the highest service ranking {3} for this JVM. The ObjectGrid instance uses service ranking {4} for this service.
- CWOBJ6412W
  MULTI_BLUEPRINT_SERVICE_FOUND_CWOBJ6412=CWOBJ6412W: The following OSGi blueprint container classes are found: {0}.
- CWOBJ6413W
  NUMBER_SERVICES_NOT_MATCH_AFTER_UPDATE_CWOBJ6413=CWOBJ6413W: After updating the {0} OSGi service from the old service ranking {1} to the new service ranking {2}, the number of service instances is changed from {3} to {4}.
- CWOBJ6414W
  SERVER_PROPERTIES_UPDATED_CWOBJ6414=CWOBJ6414W: The server property file is updated to {0} using OSGi Configuration Admin while the eXtreme Scale server is running. The update does not take effect until the server is restarted.
- CWOBJ6415E
  BUNDLE_RESTART_NOT_ALLOWED_CWOBJ6415=CWOBJ6415E: Restarting the eXtreme Scale (XS) bundles is not allowed.
- CWOBJ7000I
  STATS_MAP_INJECTION_GRID_ENABLED_CWOBJ7000=CWOBJ7000I: ObjectGrid {0} is enabled for storing historic statistics on container "{1}".
- CWOBJ7001I
  STATS_MAP_INJECTION_GRID_DISABLED_CWOBJ7001=CWOBJ7001I: ObjectGrid {0} is disabled for storing historic statistics on container "{1}".
- CWOBJ7002I
  STATS_MAP_INJECTION_MAPSET_ENABLED_CWOBJ7002=CWOBJ7002I: ObjectGrid:MapSet {0}:{1} is enabled for storing historic statistics on container "{2}".
- CWOBJ7003I
  STATS_MAP_INJECTION_MAPSET_DISABLED_CWOBJ7003=CWOBJ7003I: ObjectGrid:MapSet {0}:{1} is disabled for storing historic statistics on container "{2}".
- CWOBJ7006I
  DynamicPort=CWOBJ7006I: ObjectGrid server agent generated dynamic port {0}.
- CWOBJ7100E
  STATS_COLLECTOR_CWOBJ7100=CWOBJ7100E: Could not locate internal ObjectGrid information map for the following shard:{0}. Ensure that the grid and mapset is appropriately enabled for historic statistics monitoring.
- CWOBJ7101E
  STATS_COLLECTOR_CWOBJ7101=CWOBJ7101E: Statistic monitoring infrastructure could not find any maps associated with ObjectGrid {0}. No monitoring will be performed on an empty ObjectGrid
- CWOBJ7102E
  STATS_COLLECTOR_CWOBJ7102=CWOBJ7102E: Statistic monitoring infrastructure could not retrieve statistics for path {0}. Ensure statistics tracking is enabled for this process.
- CWOBJ7103I
  STATS_COLLECTOR_ROUTING_ADDITION_PROCESSED_CWOBJ7103=CWOBJ7103I: Statistic charting can now display charts using statistics from partition {0}.
- CWOBJ7104I
  STATS_COLLECTOR_ROUTING_DELETION_PROCESSED_CWOBJ7104=CWOBJ7104I: Statistic charting has been informed to remove its reference to partition {0}.
- CWOBJ7200I
  DeadServer=CWOBJ7200I: Detected the failure of server ({0}) in core group ({1}).
- CWOBJ7201I
  NewServer=CWOBJ7201I: Detected the addition of new server ({0}) in core group ({1}).
- CWOBJ7203I
  NewLeader=CWOBJ7203I: Leader changed. New leader ({0}) is elected in core group ({1}) and reported to the catalog service.
- CWOBJ7205I
  CWOBJ7205=CWOBJ7205I: Server, {0}, has sent a membership change notice that will be rejected as this member has already been removed from the core group.

- CWOBJ7211I
  CWOBJ7211=CWOBJ7211I: As a result of a heartbeat (view heartbeat type) from leader {0} for core group {1} with member list {2}, the server {3} is being removed from the core group view.
- CWOBJ7212I
  HA_STANDALONE_DEFINED_SET_EXPORTED_CWOBJ7212I=CWOBJ7212I: The catalog server has sent an updated defined set with version {0} and host:port list {1} to the following list of servers: {2}.
- CWOBJ7213W
  HA_SPLIT_PARTITION_DETECTED_CWOBJ7213W=CWOBJ7213W: The core group {0} received a heart beat notfication from the server {1} with revision {2} and a current view listing {3} and previous listing {4} - such a combination indicates a partitioned core group.
- CWOBJ7214I
  HA_DEFINED_SET_VIEW_DIFFERENCE_IGNORED_CWOBJ7214I=CWOBJ7214I: While processing a container heart beat for the core group {0}, a difference between the defined set and view was detected. However, since the previous and current views are the same, {1}, this difference can be ignored.
- CWOBJ7300W
  CONTAINER_FAILED_BOOTSTRAP_CWOBJ7300=CWOBJ7300W: This server failed to bootstrap to a catalog server at the following address(es): {0}. Will retry in {1} ms.
- CWOBJ7301E
  CONTAINER_TIMEOUT_BOOTSTRAP_CWOBJ7301=CWOBJ7301E: This server failed to start because it exceeded the maximum allowable number of retry attempts for bootstraping to a catalog server.
- CWOBJ7400E
  CATALOGSERVICE_ENDPOINTS_INCONSISTENT_ORDER_CWOBJ7400=CWOBJ7400E: The decision to honor the order of catalogServiceEndPoints must be consistent across the catalog servers in the domain. This server ({0}) will be stopped. Exception detected: {1}
- CWOBJ7401E
  CATALOGSERVICE_ENDPOINTS_STRING_INCONSISTENT_CWOBJ7401=CWOBJ7401E: A discrepancy in the catalogServiceEndPoints value was detected. The catalogServiceEndPoints value must be the same on each catalog server. This server ({0}) will be stopped. Exception detected: {1}
- CWOBJ7402I
  CATALOGSERVICE_ENDPOINTS_ORDERED_CWOBJ7402=CWOBJ7402I: This catalog server is honoring the order of catalogServiceEndPoints.
- CWOBJ7403E
  UNABLE_TO_START_EXTREME_SCALE_TRANSPORT_CWOBJ7403=CWOBJ7403E: The eXtreme Scale transport did not start.
- CWOBJ7404I
  OFFHEAP_ENABLED_CWOBJ7404=CWOBJ7404I: Off-heap memory storage is enabled for the {0} server.
- CWOBJ7405E
  FAILED_TO_GET_EVICTION_LIST_CWOBJ7405=CWOBJ7405E: Failed to get eviction list from off-heap address.
- CWOBJ7406W
  XM_NO_CONTAINER_CWOBJ7406=CWOBJ7406W: No container named {0} hosted on this server.
- CWOBJ7407W
  XM_NO_SHARD_CWOBJ7407=CWOBJ7407W: No shard named {0} hosted on this server.
- CWOBJ7408E
  ERROR_STARTING_XIO_TRANSPORT_CWOBJ7408=CWOBJ7408E: Caught exception starting eXtremeIO transport service.
- CWOBJ7409E
  ERROR_STARTING_LOADING_XM_NATIVE_LIBRARIES_CWOBJ7409=CWOBJ7409E: Caught exception starting eXtremeMemory due to missing native libraries.
- CWOBJ7500W
  ROUTE_TABLE_PARTITION_PURGE_CWOBJ7500=CWOBJ7500W: Partition {0} will be removed from the route table because that partition entry is stale.
- CWOBJ7501I
  ROUTE_TABLE_UPDATES_CWOBJ7501=CWOBJ7501I: The following partitions listed by the form grid:mapSet:partitionId:gridEpoch:partitionEpoch just had their routing entries update: {0}.
- CWOBJ7600E
  CANNOT_SERIALIZE_VALUE_CWOBJ7600=CWOBJ7600E: Cannot serialize cache entry value {0}. Serialization failed.
- CWOBJ7601E
  CANNOT_SERIALIZE_KEY_CWOBJ7601=CWOBJ7601E: Cannot serialize cache entry key {0}. Serialization failed.
- CWOBJ7602E
  CATALOG_SERVICE_DOMAIN_BEAN_INITIALIZATION_FAIL_CWOBJ7602=CWOBJ7602E: Object Grid Catalog Service Domain Bean failed to initialize. Exception occured {0}
- CWOBJ7603E
  CLIENT_BEAN_INITIALIZATION_FAIL_CWOBJ7603=CWOBJ7603E: Object Grid Client Bean failed to initialize. Exception occured {0}
- CWOBJ7700I
  PeerManagerStart=CWOBJ7700I: Peer Manager service started successfully in server ({0}) with core group ({1}).
- CWOBJ7800I
  Start_HAController=CWOBJ7800I: Start ObjectGrid HA Controller with core group ({0}), host ({1}), and port ({2}).
- CWOBJ8000I
  Register_CWOBJ8000=CWOBJ8000I: Registration is successful with zone ({0}) and coregroup of ({1}).
- CWOBJ8009E
  Failed_Register_CWOBJ8009=CWOBJ8009E: Registration failed for zone ({0})
- CWOBJ8101I
  StandbyCatalogServerCreated_CWOBJ8101=CWOBJ8101I: Notify that standby catalog service is created with domain= {0} and with IOR= {1}
- CWOBJ8102I
  MasterCatalogServerCreated_CWOBJ8102=CWOBJ8102I: Notify that master catalog service is created with domain= {0} and with IOR= {1}
- CWOBJ8106I
  MasterCatalogServerActivated_CWOBJ8106=CWOBJ8106I: The master catalog service cluster activated with cluster {0}
- CWOBJ8108I
  ResentStandbyCatalogServer_CWOBJ8108=CWOBJ8108I: Re-send standby catalog service on the request of master catalog service with domain= {0} and IOR= {1}
- CWOBJ8109I
  UpdateCatalogServerCluster_CWOBJ8109=CWOBJ8109I: Updated catalog service cluster {0} from server {1} with entry {2}

- CWOBJ8401I
  WaitForReplica_CWOBJ8401=CWOBJ8401I: Waiting for a server replica to be started. Start another server(s) immediately.
- CWOBJ8601I
  PeerServers_CWOBJ8601=CWOBJ8601I: PeerManager found peers of size {0}
- CWOBJ9000I
  ENGLISH_ONLY_INFO_MESSAGE_CWOBJ9000=CWOBJ9000I: This message is an English-only Informational message: {0}.
- CWOBJ9001W
  ENGLISH_ONLY_WARN_MESSAGE_CWOBJ9001=CWOBJ9001W: This message is an English-only Warning message: {0}.
- CWOBJ9002E
  ENGLISH_ONLY_ERROR_MESSAGE_CWOBJ9002=CWOBJ9002E: This message is an English only Error message: {0}.

Related tasks:
Configuring
Administering

CWOBJ0001E

ILLEGAL_STATE_EXCEPTION_CWOBJ0001=CWOBJ0001E: Method, {0}, was called after initialization completed.

Explanation

After initialization completes, certain method invocations are no longer accepted.

User response

Configuration methods must be called before initialization. Restructure the code so the configuration is completed before the runtime object is initialized. For example, on ObjectGrid, methods such as defineMap, setTransactionCallback, or setSecurityEnabled must be called before an ObjectGrid is used.
Related tasks:
Configuring
Administering
Controlling the lifecycle of an ObjectGrid with the ObjectGridManager interface

CWOBJ0002W

IGNORING_UNEXPECTED_EXCEPTION_CWOBJ0002=CWOBJ0002W: ObjectGrid component, {1}, is ignoring an unexpected exception: {0}.

Explanation

An exception occurred, but the runtime will ignore the exception and continue to work on the current action.

User response

Review the exception provided for any configuration errors. Look for any additional errors in the log. Also review the first failure data capture (FFDC) logs. See the Logs and Trace section in the information center for JVM log location.
Related tasks:
Configuring
Administering
Enabling logging
Analyzing log and trace data

CWOBJ0003W

DEPRECATED_FUNCTION_CWOBJ0003W=CWOBJ0003W: The {0} function was deprecated in the WebSphere eXtreme Scale {1} release and will be removed in a future release. See {2} in the information center for more information.

Explanation

The function referenced in the message was deprecrated and should not be used. While the server will continue to work, the function will be removed in the future.

User response

Refer to the information center for the new function that should be used instead of the deprecated function.
Related tasks:
Configuring
Administering
Related reference:
Deprecated properties and APIs

CWOBJ0004W

DEPRECATED_FUNCTION_CWOBJ0004W=CWOBJ0004W: The {0} method is deprecated. The {1} function was deprecated in the WebSphere eXtreme Scale {2} release and will be removed in a future release. See {3} in the information center for more information.

Explanation

The method and function referenced in the message was deprecrated and should not be used. While the server will continue to work, the function will be removed in the future.

User response

Refer to the information center for the new method and function that should be used instead of the deprecated function.
Related tasks:
Configuring
Administering
Related reference:
Deprecated properties and APIs

CWOBJ0005W

INTERRUPTED_EXCEPTION_CWOBJ0005=CWOBJ0005W: The thread created an java.lang.InterruptedException: {0}

Explanation

A java.lang.InterruptedException occurred and woke up the waiting or sleeping thread.

User response

Check the exception message to see whether this interruption is expected. For example, if the user stopped the login panel. Otherwise, check for configuration or compatibility errors.
Related tasks:
Configuring
Administering

CWOBJ0006W

GENERAL_EXCEPTION_WARNING_CWOBJ0006=CWOBJ0006W: An exception occurred: {0}

Explanation

An exception occurred during the runtime.

User response

Check the exception message to see whether this is an expected exception. If it is not expected, check for configuration errors, network problems or other previous errors in the log. Also review the first failure data capture (FFDC) logs. See the Logs and Trace section in the information center for JVM log location.
Related tasks:
Configuring
Administering
Enabling logging
Analyzing log and trace data

CWOBJ0007W

NULL_VALUE_WARNING_CWOBJ0007=CWOBJ0007W: The invalid value of null was specified for {0}. The default value of {1} will be used instead.

Explanation

The value null is invalid for the variable or property. A default value will be used.

User response

Replace null with a value appropriate for your deployment environment. Search on the variable or property name in the information center for valid values.
Related tasks:
Configuring

Administering

CWOBJ0008E

INVALID_VALUE_ERROR_CWOBJ0008=CWOBJ0008E: The value {0} provided for the property {1} is not valid.

Explanation

A not valid value was specified for the variable.

User response

Replace the current not valid value with a value appropriate for your deployment environment. Search on the variable or property name in the information center for valid values.
Related tasks:
Configuring
Administering

CWOBJ0010E

MISSING_KEY_ERROR_CWOBJ0010=CWOBJ0010E: Message key {0} is missing.

Explanation

A message key is missing in the message resource bundle and cannot be printed to the log file.

User response

Examine the first failure data capture (FFDC) logs for errors that occurred at the time as the missing message key. Not all errors are also printed to the FFDC logs. Also contact IBM Software Support with the missing message key.
Related tasks:
Configuring
Administering
Enabling logging
Analyzing log and trace data

CWOBJ0012E

INVALID_LOGELEMENT_TYPE_CWOBJ0012=CWOBJ0012E: The LogElement type code, {0} ({1}), is not recognized for this operation.

Explanation

An internal error occurred in the ObjectGrid runtime.

User response

Contact IBM Software Support.
Related tasks:
Configuring
Administering

CWOBJ0013E

EVICT_ENTRIES_EXCEPTION_CWOBJ0013=CWOBJ0013E: An exception occurred while attempting to evict entries from the cache: {0}

Explanation

There was an exception evicting entries from the cache. The exception should be handled in the custom evictor code.

User response

Check the exception message to see whether this is an expected exception handled by the custom evictor. If it is not expected, check for configuration errors. Review the custom evictor code.
Related concepts:
Evictors
Related tasks:

Configuring
Administering
Writing a custom evictor
Enabling evictors with XML configuration

CWOBJ0021E

OBJECT_TRANSFORMER_NOT_FOUND_CWOBJ0021=CWOBJ0021E: A usable ObjectTransformer instance was not found during the deserialization of the LogSequence object for {0} ObjectGrid and {1} ObjectMap.

Explanation

The receiving side of a LogSequence object does not have the proper configuration to support the required ObjectTransformer instance.

User response

Verify the configuration of the ObjectGrid instances for both the sending and receiving sides of the LogSequence object.
Related tasks:
Configuring
Administering
Tracking map updates by an application
Related reference:
LogSequence interface
Deprecated properties and APIs

CWOBJ0022E

LOCK_MANAGER_INTERNAL_ERROR_CWOBJ0022=CWOBJ0022E: The caller does not own mutex: {0}.

Explanation

An internal error occurred in the ObjectGrid runtime.

User response

Contact IBM Software Support.
Related tasks:
Configuring
Administering
Contacting IBM support

CWOBJ0023E

UNRECOGNIZED_COPY_MODE_CWOBJ0023=CWOBJ0023E: The CopyMode ({0}) is not recognized for this operation.

Explanation

An internal error occurred in the ObjectGrid runtime.

User response

Contact IBM Software Support.
Related tasks:
Configuring
Administering
Contacting IBM support

CWOBJ0024E

REQUIRED_FIELD_NOT_FOUND_CWOBJ0024=CWOBJ0024E: Cannot deserialize field {0} in class {1}. Deserialization failed.

Explanation

During deserialization of an object, a required field was not found. This problem is likely an ObjectGrid runtime error.

User response

Contact IBM Software Support.
Related tasks:
Configuring
Administering

CWOBJ0025E

SERIALIZATION_FAILED_CWOBJ0025=CWOBJ0025E: The serialization of the LogSequence object failed. The number of serialized LogElement objects ({0}) does not match the number of read LogElement objects ({1}).

Explanation

An internal error occurred in the ObjectGrid runtime.

User response

Contact IBM Software Support.
Related tasks:
Configuring
Administering
Contacting IBM support

CWOBJ0026E

INVALID_JMX_CREDENTIAL_CWOBJ0026=CWOBJ0026E: The JMX credential type is not right. It should be of type {0}.

Explanation

The JMX credential type supplied is not correct and the user cannot be authenticated.

User response

Use the right credentials. Use the type suggested. If basic authentication is used, the expected type is String[] with the first element being user name and the second being password.
Related tasks:
Configuring
Administering

CWOBJ0027E

CLONE_METHOD_NOT_SUPPORTED_CWOBJ0027=CWOBJ0027E: Internal runtime error. Clone method not supported: {0}

Explanation

An internal error occurred in the ObjectGrid runtime.

User response

Contact IBM Software Support.
Related tasks:
Configuring
Administering
Contacting IBM support

CWOBJ0030I

OBJECTGRID_INSTRUMENTATION_ENABLED_CWOBJ0030=CWOBJ0030I: ObjectGrid entity class instrumentation is enabled. The instrumentation mode is {0}.

Explanation

ObjectGrid entity class instrumentation is enabled. Java classes in the configured transformation domain may be transformed to support field-access entities.

User response

No action is required.
Related tasks:

Configuring
Administering

CWOBJ0033I

CLASS_NOT_IMPLEMENT_CLONE_CWOBJ0033=CWOBJ0033I: Class, {0}, does not implement the clone method. Using serialization instead for this Class in map {1}.

Explanation

If using the default ObjectTransformer and the object type does not implement the clone() method, the ObjectGrid is serialized and deserialized to create a copy of the object.

User response

If you are using a copy mode such as COPY_ON_READ or COPY_ON_READ_AND_COMMIT and your performance is slow, add a clone method to your object type or provide a custom ObjectTransformer implementation for your object type. See the ObjectTransformer interface best practices section in the information center for more information about the ObjectTransformer interface.
Related tasks:
Configuring
Administering
Related reference:
Deprecated properties and APIs

CWOBJ0034I

TARGET_MEMORY_UTILIZATION_THRESHOLD_LEVEL_CWOBJ0034=CWOBJ0034I: Memory utilization threshold percentage is set to {0} %.

Explanation

The target memory utilization threshold percentage is set to the target level.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ0035W

MEMORY_UTILIZATION_THRESHOLD_NOT_SUPPORTED_CWOBJ0035=CWOBJ0035W: Memory utilization threshold not supported for this JVM.

Explanation

The Java Virtual Machine (JVM) does not support memory utilization threshold. The memory utilization threshold is only supported in JVMs at version 5.0 or later.

User response

Evictors and other ObjectGrid components will not respond to memory utilization threshold events. To stop this message from occurring: Remove the server property setting for memoryThresholdPercentage (or set to -1) if it is configured. Or remove the MEMORY_USAGE_THRESHOLD trigger on the BackingMap if configured. Or do not set the Java MemoryMXPool bean programmatically.
Related tasks:
Configuring
Administering

CWOBJ0036W

CHANGING_MEMORY_UTILIZATION_THRESHOLD_CWOBJ0036=CWOBJ0036W: Changing memory utilization threshold from {0} to {1} for {2} memory pool.

Explanation

The Java heap memory utilization threshold is changed. This may have an impact on other system components that rely on the memory utilization threshold setting.

User response

Ensure the new memory utilization threshold is acceptable.
Related tasks:
Configuring
Administering

CWOBJ0037W

CHANGING_MEMORY_COLLECTION_UTILIZATION_THRESHOLD_CWOBJ0037=CWOBJ0037W: Changing memory collection utilization threshold from {0} to {1} for {2} memory pool.

Explanation

The Java heap memory collection utilization threshold is changed. This may have an impact on other system components that rely on the memory collection utilization threshold setting.

User response

Ensure the new memory collection utilization threshold is acceptable.
Related tasks:
Configuring
Administering

CWOBJ0038W

MEMORY_THRESHOLD_EXCEEDED_CWOBJ0038=CWOBJ0038W: Memory threshold exceeded. Current heap memory usage: {0}.

Explanation

The Java heap memory threshold exceeded target usage threshold.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ0039W

MEMORY_COLLECTION_THRESHOLD_EXCEEDED_CWOBJ0039=CWOBJ0039W: Memory collection threshold exceeded. Current heap memory usage: {0}.

Explanation

The Java heap memory collection threshold exceeded target usage threshold.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ0040E

CWOBJ0040=CWOBJ0040E: Hash based data structure over run for {0} with {1} elements in the data structure. Examine the hashCode method on this class for better distribution.

Explanation

The hash based data structure holding elements in the grid is getting too many collisions. This is likely because the hashCode method on the key class has not been implemented effectively.

User response

Examine the hashCode algorithm of the class to determine if a more distributed result is possible.
Related tasks:
Configuring
Administering

CWOBJ0041W

RANGE_INDEX_NOT_SUPPORTED_CWOBJ0041=CWOBJ0041W: The rangeIndex property of HashIndex plug-in cannot be set to true for a composite index: {0}. The rangeIndex property setting will be ignored.

Explanation

Composite indexes do not support range indexing.

User response

Verify that the rangeIndex property of HashIndex is not configured or is set to false and resubmit the operation.
Related tasks:
Configuring
Administering

CWOBJ0042E

EXCEPTION_MAPPER_THROWABLE_IGNORED_CWOBJ0042=CWOBJ0042E: The ExceptionMapper implementation class {0} threw an unexpected exception with the following message: {1}. This exception is ignored.

Explanation

The ExceptionMapper implementation class threw an exception. This is most likely due to an incorrect implementation.

User response

Examine the first failure data capture (FFDC) logs and the implementation class to find out why an exception was thrown, correct the problem and resubmit the operation.
Related tasks:
Configuring
Administering

CWOBJ0043W

CATALOG_CONFIG_PROBLEM_CELL_PROPERTY_CWOBJ0043=CWOBJ0043W: The {0} is formatted improperly but was corrected: {1}

Explanation

The catalog service configuration used "/" instead of "", but the data was automatically corrected.

User response

Update the catalog service configuration to use the correct formatting and restart the process or application to avoid this message. See the Starting the catalog service process in a WebSphere Application Server environment section in the information center for more information on catalog service configuration.
Related tasks:
Configuring
Administering
Configuring the catalog service in WebSphere Application Server
Creating and augmenting profiles for WebSphere eXtreme Scale

CWOBJ0044E

FORMAT_ERROR_INITIALIZE_CATALOG_CWOBJ0044=CWOBJ0044E: Invalid data in the {0}: {1}. The exception is: {2}

Explanation

The eXtreme Scale runtime environment uses the catalog service configuration to define the catalog service for the process. The data is formatted incorrectly which may result in a failure to start a catalog service, start a container or connect to a catalog service from a client.

User response

Review and fix the format of the catalog service configuration and restart the process or application. See the Starting the catalog service process in a WebSphere Application Server environment section in the information center for more information on catalog service configuration.
Related tasks:
Configuring
Administering
Configuring the catalog service in WebSphere Application Server
Creating and augmenting profiles for WebSphere eXtreme Scale

CWOBJ0045W

ERROR_CREATING_MBEAN_CWOBJ0045=CWOBJ0045W: An exception occurred while creating an MBean with ObjectName: {0}. The exception is: {1}.

Explanation

There was an exception while attempting to register the specified MBean.

User response

Review the exception, if there is a port conflict for the JMX Service, check for another service running on the JMX Service Port (the default is 1099). The JMX port may be changed using the -JMXServicePort option. If there is a security related error, see the Java Management Extensions (JMX) security section in the information center.
Related tasks:
Configuring
Administering

CWOBJ0046I

MEMORY_THRESHOLD_DEFAULT_PERCENT_USED_CWOBJ0046=CWOBJ0046I: The Java MemoryMXPool bean was not set (current value = 0). During initialization, the memoryThresholdPercentage property is set to default value of {0}.

Explanation

Memory-based eviction was enabled on the backing map, but a target usage threshold or memoryThresholdPercentage property was not set. The default value is being used.

User response

If the default value is acceptable, no action is required. You can set the memoryThresholdPercentage value in the server properties file or with the Java MemoryMXPool bean. See the information center for details about how to configure a server properties file.
Related tasks:
Configuring
Administering

CWOBJ0047I

DEVELOPMENT_MODE_ENABLED_CWOBJ0047=CWOBJ0047I: Development mode is enabled for one or more MapSets for ObjectGrids: {0}. For a production deployment, set the development mode attribute in the deployment policy file to false.

Explanation

Development mode allows primary and replica shards for the same partition to be placed on the same machine. When using one or two machines to develop code, this behavior is acceptable. However, when running in production, allowing placement of a primary and its replicas on the same machine risks data loss in case of whole machine failure.

User response

If you are in production phase or test phase consider changing development mode to false. The default is true. Otherwise, no action is required. See the Deployment policy descriptor XML file article in the information center for more information about the developmentMode setting.
Related concepts:
Shard placement
High availability
Related tasks:
Configuring
Administering
Related reference:
Development mode descriptor XML file

CWOBJ0048E

START_PROCESS_IN_WAS_CWOBJ0048=CWOBJ0048E: Starting stand-alone WebSphere eXtreme Scale server processes in a WebSphere Application Server 6.0.x deployment is not supported.

Explanation

This configuration of WebSphere eXtreme Scale and WebSphere Application Server 6.0.x is not supported.

User response

Install WebSphere eXtreme Scale outside of WebSphere Application Server or move to a version of WebSphere Application Server 6.1.x or higher.
Related tasks:
Configuring
Administering
Configuring the catalog service in WebSphere Application Server
Creating and augmenting profiles for WebSphere eXtreme Scale

CWOBJ0049W

WAS_NOT_AUGMENTED_CWOBJ0049=CWOBJ0049W: This profile is not augmented with WebSphere eXtreme Scale. WebSphere eXtreme Scale container servers will therefore not start automatically.

Explanation

The current profile is not augmented with WebSphere eXtreme Scale. WebSphere eXtreme Scale features are not available for the profile to use until it is augmented.

User response

If the profile needs to use WebSphere eXtreme Scale, augment the profile using the WebSphere Application Server profile tools. See the Creating and augmenting profiles for WebSphere eXtreme Scale section in the information center.
Related tasks:
Configuring
Administering
Configuring the catalog service in WebSphere Application Server
Creating and augmenting profiles for WebSphere eXtreme Scale

CWOBJ0050W

INVALID_PORT_BOOTSTRAP_OVERRIDE_CWOBJ0050=CWOBJ0050W: Invalid listenerPort {0} defined in the {1}. Overriding it with the bootstrap address port (BOOTSTRAP_ADDRESS) {2}.

Explanation

The listenerPort provided in the catalog service configuration was incorrect. The bootstrap port from the server will be used instead.

User response

Check the listenerPort in the catalog service configuration. See the Starting the catalog service process in a WebSphere Application Server environment section in the information center for more information on catalog service configuration.
Related tasks:
Configuring
Administering
Configuring the catalog service in WebSphere Application Server
Creating and augmenting profiles for WebSphere eXtreme Scale

CWOBJ0051W

WAS_NOT_AUGMENTED_CWOBJ0051=CWOBJ0051W: This profile is not augmented with WebSphere eXtreme Scale. A catalog service will therefore not start automatically.

Explanation

The profile is not augmented with WebSphere eXtreme Scale. WebSphere eXtreme Scale features are not available for the profile to use until it is augmented.

User response

If the profile needs to use WebSphere eXtreme Scale, augment the profile using the WebSphere Application Server profile tools. See the Creating and augmenting profiles for WebSphere eXtreme Scale section in the information center.
Related tasks:
Configuring
Administering
Configuring the catalog service in WebSphere Application Server
Creating and augmenting profiles for WebSphere eXtreme Scale

CWOBJ0052I

ORB_CHANNELFRAMEWORK_CWOBJ0052=CWOBJ0052I: The IBM ORB TransportMode property was set to ChannelFramework.

Explanation

The TransportMode was set automatically to ChannelFramework.

User response

No action is required unless you do not want to use the ChannelFramework TransportMode. The TransportMode may be set to Pluggable by setting it in the orb.properties file.
Related tasks:
Configuring
Administering
Configuring Object Request Brokers

CWOBJ0053I

ORB_SEVERSOCKETQUEUEDEPTH_OVERRIDE_CWOBJ0053=CWOBJ0053I: The IBM ORB ServerSocketQueueDepth property was set to {0} to run with correctly with the ChannelFramework TransportMode.

Explanation

The ChannelFramework TransportMode has a maximum allowed ServerSocketQueueDepth. If the current ServerSocketQueueDepth is greater than the allowed value, it will be reset to the maximum allowed value.

User response

No action is required. To stop this message from occurring, reset the ServerSocketQueueDepth to the suggested value in the orb.properties.
Related tasks:
Configuring
Administering
Configuring Object Request Brokers

CWOBJ0054I

WXS_PROPERTY_CWOBJ0054=CWOBJ0054I: The value of the "{0}" property is "{1}".

Explanation

This is a WebSphere eXtremeScale custom property.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ0055W

ORB_CHANNELFRAMEWORK_CWOBJ0055=CWOBJ0055W: The IBM ORB TransportMode property was set to ChannelFramework in the server properties file, but the existing orb.properties file already had a TransportMode set. The TransportMode will not be overridden.

Explanation

The TransportMode set in the server properties file will not override a TransportMode set in the orb.properties file.

User response

Review the TransportMode set in the orb.properties file. If it is set to Pluggable or SSL or Transport Security settings are used, the ChannelFramework Transport mode will not be used. To remove the warning, adjust the setting in one of the files.
Related tasks:
Configuring
Administering
Configuring Object Request Brokers

CWOBJ0056I

ORB_PROPERTY_OVERRIDE_CWOBJ0056=CWOBJ0056I: The Object Request Broker (ORB) property, {0}, with the value, {1}, is being overridden with the value, {2}.

Explanation

The ORB property that is defined in the orb.properties file is overridden with a new value. These properties are overridden to support WebSphere eXtreme Scale transport security.

User response

No action is required.
Related tasks:
Configuring
Administering
Configuring Object Request Brokers

CWOBJ0057E

CATALOG_VERSION_DOWN_LEVEL_CWOBJ0057=CWOBJ0057E: The WebSphere eXtreme Scale catalog server version is {0}, and the client or container server version is {1}.

Explanation

The catalog server version cannot be older than the client or container server version. This configuration is not supported.

User response

Upgrade the catalog sever to a version that is the same or newer than the client and container server versions.
Related tasks:
Configuring
Administering

CWOBJ0058I

SAME_GRID_DIFFERENT_MAPSETS_CWOBJ0058=CWOBJ0058I: A deployment policy conflict was detected. Additional mapsets were found for ObjectGrid {0}.

Explanation

An attempt was made to merge the two ObjectGridDeployments.

User response

No action is required.
Related tasks:
Configuring
Administering
Configuring deployment policies

CWOBJ0059I

DEFAULT_TRANSACTION_TIMEOUT_CWOBJ0059=CWOBJ0059I: The transaction time out value was not configured or was set to 0 for ObjectGrid {0}. With this configuration, transactions never time out. The transaction time out is being set to 600 seconds.

Explanation

eXtreme Scale does not recommend a configuration where transactions never time out. The transaction time out value is overridden as 600 seconds.

User response

No action is required, or set an ObjectGrid transaction time out value.
Related tasks:
Configuring
Administering
Configuring request retry timeout values

CWOBJ0060W

JVM_SHUTDOWN_HOOK_NOT_ORDERED_CWOBJ0060=CWOBJ0060W: The JVM shutdown hook is not ordered. The ORB shutdown hook might execute before eXtreme Scale shutdown hook executes. This may cause connectivity problem during the XS shutdown process.

Explanation

eXtreme Scale shutdown hook should be the first shutdown hook being executed. However, eXtreme Scale cannot gaurantee it by re-ordering the JVM shutdown hook execution.

User response

Gather the FFDC text files, and contact IBM Software Support
Related tasks:
Configuring
Administering

CWOBJ0061W

TRANSACTION_ROLLED_BACK_CWOBJ0061W=CWOBJ0061W: The transaction with TxID, {0}, that was last running on thread, {1}, on shard {2} has exceeded the configured transaction timeout value and was been marked rollback-only. This might be caused by lock contention or application deadlock, or your transaction timeout value is set too small.

Explanation

WebSphere eXtreme Scale automatically marks a transaction rollback-only, when it has exceeded its configured transaction timeout value. This might be caused by lock contention or application deadlock, or your transaction timeout value is set too small.

User response

Examine the application logic to determine whether a lock contention can be avoided. If a deadlock situation exists, remove that logic. If the transaction timeout value is too small and your application expects a long transaction, increase the transaction timeout value appropriately.
Related tasks:
Configuring
Administering
Configuring request retry timeout values

CWOBJ0062I

ORB_PROPERTY_CWOBJ0062=CWOBJ0062I: The value of the "{0}" ORB property is "{1}".

Explanation

This is an Object Request Broker (ORB) property that is used by the ORB.

User response

No action is required.
Related tasks:
Configuring
Administering
Configuring Object Request Brokers

CWOBJ0063I

ORB_DEFAULT_PROPERTY_SET_CWOBJ0063=CWOBJ0063I: The {0} property was not configured. The {0} property is being set to {1}.

Explanation

The listed property was not set in an orb.properties file. For some properties, if a value is not set for a timeout, it is set by the ORB to infinite and requests will not time out. The server sets a default value to allow requests to time out in the event of a problem. Other default settings are suggested starting points.

User response

No action is required, or set a specific value. See ORB properties in the information center for more information on changing the ORB properties and values.
Related tasks:
Configuring
Administering
Configuring Object Request Brokers

CWOBJ0064I

MEMORY_THRESHOLD_USER_OVERRIDE_CWOBJ0064=CWOBJ0064I: The memoryThresholdPercentage property is provided in a server properties file, which overrides any previously set values.

Explanation

The memoryThresholdPercentage property is set to {0} in a server properties file, which overrides the previous value.

User response

If the provided value is acceptable, no action is required. You can change the memoryThresholdPercentage in the server properties file or with the Java MemoryMXPool bean. See the information center for details about how to configure a server properties file.
Related tasks:
Configuring
Administering
Related reference:
Server properties file

CWOBJ0065W

HASHINDEX_ATTRIBUTE_NOT_FOUND_IN_SERIALIZER_METADATA_CWOBJ0065=CWOBJ0065W: The HashIndex, "{0}", for map "{1}" is enabled for multi-type access. The {2} attribute "{3}" was not defined in the {4} descriptor for the configured DataSerializer plug-in.

Explanation

The configured attribute was not found in the data descriptor for the configured DataSerializer plug-in, which allows multiple attribute types to be indexed with the same attribute name. For example, the attribute name is "id" and its type can be Integer or String.

User response

This message might indicate that the attribute path is defined incorrectly, or the data descriptor is missing a required attribute. Verify that the attribute path is defined correctly.
Related concepts:
Serializer programming overview
Related tasks:
Configuring
Administering

CWOBJ0066W

TRANSACTION_ROLLED_BACK_STATE_CWOBJ0066W=CWOBJ0066W: The {0} transaction has been marked rollback-only due to a state change of ObjectGrid {1} on shard {2} that forced transaction completion. This could be caused by an administrator changing the availability state of the ObjectGrid instance or termination of the ObjectGrid instance.

Explanation

eXtreme Scale automatically marks a transaction rollback-only when it does not end normally during some instances. When an administrator needs to quiesce activity to bring the data grid into the offline or preload state, if the transaction does not end normally in the allowed time, the transaction is marked rollback-only.

User response

The administrator should wait for a period of inactivity to change the state of the data grid. Alternatively, you can manually end or put client applications and their transactions into an inactive state before you change the state of the data grid.
Related tasks:
Configuring
Administering
Controlling the lifecycle of an ObjectGrid with the ObjectGridManager interface

CWOBJ0067W

JMX_SSL_ENABLED_WITHOUT_PORT_CWOBJ0067W=CWOBJ0067W: SSL is enabled for JMX connections to this server. However, the JMXServicePort property was not provided.

Explanation

If SSL is configured for JMX, JMX communication will not work when communicating with this server over SSL if the JMXServicePort is not specified.

User response

Either provide the JMXServicePort and JMXConnectorPort properties when starting the server or disable SSL for the JMX protocol. To disable SSL for the JMX protocol, either disable SSL for the server, or specify the JVM argument -Dcom.sun.management.jmxremote.ssl=false during server startup to disable SSL only for JMX communication.
Related tasks:
Configuring
Administering
Planning for network ports

CWOBJ0068I

JMX_SERVICE_URL_CWOBJ0068I=CWOBJ0068I: MBeanServer started with JMX URL {0}.

Explanation

The eXtreme Scale server can be contacted using the provided JMX URL.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ0069W

GRID_NOT_OFFHEAP_ELIGIBLE_CWOBJ0069=CWOBJ0069W: OffHeap is enabled but one of the BackingMaps for ObjectGrid "{0}" does not have a CopyMode of COPY_TO_BYTES or COPY_TO_BYTES_RAW. All BackingMaps for an ObjectGrid must be configured with either COPY_TO_BYTES or COPY_TO_BYTES_RAW to use OffHeap.

Explanation

All BackingMaps for an ObjectGrid must be configured with either COPY_TO_BYTES or COPY_TO_BYTES_RAW to use OffHeap.One of the maps in the MapSet has a copy mode other than COPY_TO_BYTES or COPY_TO_BYTES_RAW

User response

Configure all BackingMaps with CopyModes of either: COPY_TO_BYTES or COPY_TO_BYTES_RAW
Related tasks:
Configuring
Administering
Configuring IBM eXtremeMemory and IBM eXtremeIO

CWOBJ0070W

GC_POLICY_WARNING_CWOBJ0070W=CWOBJ0070W: The IBM implementation of the JVM is using a garbage collection policy that might affect performance.

Explanation

The IBM implementation of the JVM provides garbage collection policies that might provide enhanced performance.

User response

Specify one of the following JVM arguments: -Xgcpolicy:gencon or -Xgcpolicy:balanced. Not all versions of the JVM support each of the previous policies. See the IBM Developer Kit and Runtime Environment documentation for details.
Related tasks:
Configuring
Administering

CWOBJ0071W

MAX_HEAP_WARNING_CWOBJ0071W=CWOBJ0071W: The maximum heap size of {1} bytes surpasses the recommended maximum heap size of {0} bytes.

Explanation

As the heap size increases, garbage collection might cause performance degradation.

User response

Reduce the maximum heap size allocated by using the JVM argument, -Xmx. The limit is 8 GB when you specify -Xgcpolicy:balanced in the JVM arguments and 5 GB for all other garbage collection policies.
Related tasks:
Configuring
Administering

CWOBJ0072I

COMMAND_RUNAS_SUBJECT_CWOBJ0072I=CWOBJ0072I: The WebSphere eXtreme Scale command runtime is using the {0} Subject RunAs type.

Explanation

When you run multiple partition commands, such as those used by AgentManager, the configured RunAs type is used.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ0080W

NO_OBJECTGRID_XML_SPECIFIED_CWOBJ0080W=CWOBJ0080W: No objectgrid.xml specified for container, {0}. Will not process.

Explanation

No objectgrid.xml file was specified for the container so the container could not be started.

User response

Make sure an objectgrid.xml file is specified for the container.
Related tasks:
Configuring
Administering

CWOBJ0081I

STARTING_CONTAINER_CWOBJ0081I=CWOBJ0081I: Starting container specified at: {0} with name, "{1}".

Explanation

The specified container is being started.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ0082W

UNABLE_TO_LOAD_SERVER_PROPS_FILE_CWOBJ0082W=CWOBJ0082W: Unable to load properties from server properties file, {0}.

Explanation

There was an exception loading the server properties file.

User response

Verfiy the format of the file and try again.
Related tasks:
Configuring
Administering

CWOBJ0083W

UNRECOGNIZED_SERVER_PROPERTY_CWOBJ0083W=CWOBJ0083W: Unrecognized XS server property specified: {0} Ignoring.

Explanation

There is an unrecognized property found while processing the server properties.

User response

Remove the incorrect property or replace it with the correct value.
Related tasks:
Configuring
Administering

CWOBJ0084W

UNSUPPORTED_MULTI_PARAMETER_PROPERTY_CWOBJ0084W=CWOBJ0084W: Unsupported multi-parameter property: {0} - ignoring.

Explanation

Multi-parameter properties are unsupported.

User response

Use a sinlge parameter on the property.
Related tasks:
Configuring
Administering

CWOBJ0085W

UNKNOWN_PARAMETER_TYPE_CWOBJ0085W=CWOBJ0085W: Unknown parameter type ({0}) while setting config property, {1} - ignoring.

Explanation

An unknown parameter type was encountered while processing the properties.

User response

If the problem persists, see problem determination information on the WebSphere Application Server Support page at http://www.ibm.com/software/webservers/appserv/was/support/.
Related tasks:
Configuring
Administering

CWOBJ0086W

FAILED_TO_SET_PROPERTY_CWOBJ0086W=CWOBJ0086W: Failed to set user property ({0}) : {1}.

Explanation

The property failed to be set.

User response

Fix the exception.
Related tasks:
Configuring
Administering

CWOBJ0087W

PROPERTY_CHANGE_SERVER_RESTART_CWOBJ0087W=CWOBJ0087W: Property, {0}, has changed from {1} to {2} but will not take effect until the server is restarted.

Explanation

The property has been changed but the new value will not be used until the server is restarted.

User response

Restart the server if the new property value is required.
Related tasks:
Configuring
Administering

CWOBJ0088W

UNRECOGNIZED_WEB_APP_CONFIG_PROPERTY_CWOBJ0088W=CWOBJ0088W: Unrecognized XS Web App Configuration property specified: {0} Ignoring.

Explanation

here is an unrecognized property found while processing the server properties.

User response

Remove the incorrect property or replace it with the correct value.
Related tasks:
Configuring
Administering

CWOBJ0900I

CWOBJ0900=CWOBJ0900I: The ObjectGrid runtime component is started for server {0}.

Explanation

The ObjectGrid component is started.

User response

No action is required.
Related tasks:
Configuring
Administering
Controlling the lifecycle of an ObjectGrid with the ObjectGridManager interface

CWOBJ0901E

CWOBJ0901=CWOBJ0901E: {0} system property is required to start ObjectGrid runtime component for server: {1}.

Explanation

ObjectGrid runtime component is missing a required Java Virtual Machine system property.

User response

See the Administering WebSphere eXtreme Scale with WebSphere Application Server section in the information center.
Related tasks:
Configuring
Administering
Configuring the catalog service in WebSphere Application Server
Creating and augmenting profiles for WebSphere eXtreme Scale

CWOBJ0902W

CWOBJ0902=CWOBJ0902W: An error prevented the ObjectGrid runtime component from starting for server: {0}.

Explanation

A prior error prevented the ObjectGrid component from starting.

User response

See prior error messages to determine what prevented ObjectGrid component from starting.
Related tasks:
Configuring
Administering
Controlling the lifecycle of an ObjectGrid with the ObjectGridManager interface

CWOBJ0903I

INTERNAL_OBJECTGRID_VERSION_CWOBJ0903=CWOBJ0903I: The internal version of WebSphere eXtreme Scale is {0}.

Explanation

Displays the internal version of WebSphere eXtreme Scale for use by IBM Software Support.

User response

No action is required.
Related tasks:
Configuring
Administering
Contacting IBM support

CWOBJ0904E

FILES_DO_NOT_EXIST_CWOBJ0904=CWOBJ0904E: {0} exists but the following file or files are missing: {1}. Cannot start the ObjectGrid runtime component for server: {2}.

Explanation

WebSphere eXtreme Scale requires one or more missing files in order to start the runtime component for this server.

User response

Ensure the required file or files are present and perform this operation again. See the Administering WebSphere eXtreme Scale with WebSphere Application Server section in the information center for more details.
Related tasks:
Configuring
Administering
Configuring the catalog service in WebSphere Application Server
Creating and augmenting profiles for WebSphere eXtreme Scale

CWOBJ0905I

FILES_NOT_FOUND_CWOBJ0905=CWOBJ0905I: WebSphere eXtreme Scale did not find object grid configuration files packaged with application {0}.

Explanation

WebSphere eXtreme Scale requires configuration files in order to start the runtime components for this server.

User response

If the application is intended to provide object grid configuration, ensure the files are present and appropriately named, otherwise no action is neccessary.
Related tasks:
Configuring
Administering
Controlling the lifecycle of an ObjectGrid with the ObjectGridManager interface

CWOBJ0910I

CWOBJ0910=CWOBJ0910I: The ObjectGrid runtime component is stopped for server {0}.

Explanation

The ObjectGrid component is stopped.

User response

No action is required.
Related tasks:
Configuring
Administering
Controlling the lifecycle of an ObjectGrid with the ObjectGridManager interface

CWOBJ0912E

CWOBJ0912=CWOBJ0912E: The application {0} has ObjectGrid configuration files that will not be used because application {1} is currently running an ObjectGrid server instance.

Explanation

Two applications with ObjectGrid server configuration files with this server name are deployed to this application server. Only one ObjectGrid server configuration is allowed in a WebSphere Application Server.

User response

Ensure there is only one ObjectGrid server application deployed on this server.
Related tasks:
Configuring
Administering
Configuring the catalog service in WebSphere Application Server
Creating and augmenting profiles for WebSphere eXtreme Scale

CWOBJ0913I

LOADED_PROPERTY_FILES_CWOBJ0913=CWOBJ0913I: Server property files have been loaded: {0}.

Explanation

One or more server properties files were loaded. If multiple files are displayed, then the properties are loaded in the order displayed.

User response

No action is required.
Related tasks:
Configuring
Administering
Related reference:
Server properties file

CWOBJ0915I

ORB_VERSION_USED_CWOBJ0915=CWOBJ0915I: ORB version used is {0}.

Explanation

The Object Request Broker (ORB) version being used is listed here using com.ibm.rmi.util.Version.

User response

No action is required.
Related tasks:
Configuring
Administering
Configuring Object Request Brokers

CWOBJ0917I

ORB_LISTENING_CWOBJ0917=CWOBJ0917I: {0} ORB is listening on host and port {1}:{2}.

Explanation

Lists the hostname and the port that the ORB uses to listen for connections.

User response

No action is required.
Related tasks:
Configuring
Administering
Configuring Object Request Brokers

CWOBJ0918W

NON_OBJECTGRID_CONFIG_OBJECT_CWOBJ0918=CWOBJ0918W: The list that was supplied to override client-side ObjectGrid settings for domain/cluster {0} contains an element that is not an ObjectGridConfiguration object. This element will be removed from the List: {1}

Explanation

The specified object will be removed and not used to override any client-side settings.

User response

Review the object listed and remove it from the list of ObjectGrid override configurations supplied to the ObjectGridManager and resubmit the operation.
Related tasks:
Configuring
Administering

CWOBJ0919W

SERVER_PROPERTY_NOT_FOUND_CWOBJ0919=CWOBJ0919W: The server property file {0} cannot be found. All server properties are set to the default values.

Explanation

A ServerProperties instance was created programmatically, but the property file name supplied does not exist.

User response

Review the server property file path name, correct the error and resubmit the operation.
Related tasks:
Configuring
Administering
Related reference:
Server properties file

CWOBJ0920I

CATALOG_SERVER_NOT_STARTED_FOR_PROCESS_CWOBJ0920=CWOBJ0920I: The catalog service was not started in this process: {0}. The {1} is: {2}

Explanation

This server process is not listed in the catalog service endpoints provided in the catalog service configuration. This indicates that the catalog service is not required for this process.

User response

If the process listed should have a catalog service started, review the catalog service configuration and verify that the hostname and port for the process is included and is correct. See the Starting the catalog service process in a WebSphere Application Server environment section in the information center for more information on catalog service configuration.
Related tasks:
Configuring
Administering
Configuring the catalog service in WebSphere Application Server
Creating and augmenting profiles for WebSphere eXtreme Scale
Controlling the lifecycle of an ObjectGrid with the ObjectGridManager interface

CWOBJ0921W

OLD_CLIENT_PROP_FILE_USED_CWOBJ0921W=CWOBJ0921W: Using the Java Virtual Machine system property name "com.ibm.websphere.objectgrid.ClientProperties" to set the ObjectGrid client property file is deprecated. Use the property "objectgrid.client.props" instead.

Explanation

Using the Java Virtual Machine system property name "com.ibm.websphere.objectgrid.ClientProperties" to set the ObjectGrid client property file is deprecated. The new property "objectgrid.client.props" is recommended for setting the client property file.

User response

Use the recommended property name.
Related tasks:
Configuring
Administering
Related reference:
Deprecated properties and APIs

CWOBJ0922W

CLIENT_PROP_FILE_NOT_FOUND_CWOBJ0922W=CWOBJ0922W: The ObjectGrid client property file {0} cannot be found.

Explanation

The ObjectGrid client property file {0} cannot be found.

User response

Ensure that the file path name is correct and that the file is in the classpath. See the Client properties file article in the information center for more information about using the client properties file.
Related tasks:
Configuring
Administering
Related reference:
Client properties file

CWOBJ0923W

DEPRECATED_SERVER_SECURITY_PROP_FILE_USED_CWOBJ0923W=CWOBJ0923W: Using the Java Virtual Machine system property name "objectgrid.security.server.props" to set the ObjectGrid server security properties is deprecated. Use the property "objectgrid.server.props" instead.

Explanation

Using the Java Virtual Machine system property name "objectgrid.security.server.props" to set the ObjectGrid server security properties is deprecated. The new property "objectgrid.server.props" is recommended for setting the server properties.

User response

Use the recommended property.
Related tasks:
Configuring

Administering
Related reference:
Deprecated properties and APIs

CWOBJ0924I

LOADED_CLIENT_PROPERTY_FILES_CWOBJ0924I=CWOBJ0924I: The client property file {0} has been loaded.

Explanation

The client property file is loaded.

User response

No action is required.
Related tasks:
Configuring
Administering
Related reference:
Client properties file

CWOBJ0925E

AUTO_START_PROP_NOT_FOUND_CWOBJ0925E=CWOBJ0925E: A container autostart file {0} was found in the classpath, but the {1} property was not specified.

Explanation

The listed property could not be found while automatically starting the container.

User response

Review the autostart file and fix any formatting or syntax error or add the property.
Related tasks:
Configuring
Administering
Configuring the catalog service in WebSphere Application Server
Creating and augmenting profiles for WebSphere eXtreme Scale

CWOBJ1001I

OPEN_FOR_BUSINESS_CWOBJ1001=CWOBJ1001I: ObjectGrid Server {0} is ready to process requests.

Explanation

ObjectGrid Server is ready to process requests.

User response

The services for this ObjectGrid Server are available.
Related tasks:
Configuring
Administering
Controlling the lifecycle of an ObjectGrid with the ObjectGridManager interface

CWOBJ1003I

DCS_CWOBJ1003=CWOBJ1003I: DCS Adapter service is disabled by configuration, to enable it, please change your configuration with an endpoint defined.

Explanation

DCS adapter is turned off.

User response

Users can turn on DCS adapter by changing the configuration.
Related tasks:

Configuring
Administering

CWOBJ1004E

SERVER_TOPIC_CWOBJ1004=CWOBJ1004E: Server topic is null

Explanation

Server topic is null

User response

If the problem persists, see problem determination information on the WebSphere Application Server Support page at http://www.ibm.com/software/webservers/appserv/was/support/.
Related tasks:
Configuring
Administering

CWOBJ1005E

CLIENT_REQUESTQ_CWOBJ1005=CWOBJ1005E: The incoming request queue is null.

Explanation

Client request handler cannot retrieve requests.

User response

Contact IBM Software Support.
Related tasks:
Configuring
Administering

CWOBJ1006E

CLIENT_RESULTQ_CWOBJ1006=CWOBJ1006E: The outgoing result queue is null.

Explanation

Client request handler cannot give result to client.

User response

Contact IBM Software Support.
Related tasks:
Configuring
Administering

CWOBJ1007E

CLIENT_REQUEST_CWOBJ1007=CWOBJ1007E: ObjectGrid client request is null.

Explanation

Client request handler cannot handle request that does not contain any information about the request.

User response

Contact IBM Software Support.
Related tasks:
Configuring
Administering
Contacting IBM support

CWOBJ1008E

TXID_CWOBJ1008=CWOBJ1008E: ObjectGrid client request TxID is null.

Explanation

The TXID is necessary to match connections and for pooling. The TXID cannot be null.

User response

Contact IBM Software Support.
Related tasks:
Configuring
Administering
Contacting IBM support

CWOBJ1013W

EXCEPTION_ON_SERVER_CWOBJ1013=CWOBJ1013W: An exception occurred on a remote server: {0}

Explanation

An exception occurred during the server runtime processing of a request from the client.

User response

Check the exception message to see whether this is an expected exception. Resolve any configuration errors, network problems or security errors.
Related tasks:
Configuring
Administering

CWOBJ1014I

CLASSPATH_PROBLEM_CWOBJ1014=CWOBJ1014I: Preceding {0} message may be caused by application classes missing from the classpath on the server.

Explanation

If an application class is not in the classpath on the server a serialization error will occur on the server when processing a message from a client.

User response

Check the exception message to determine which class is missing on the server. Confirm that the class is included on the classpath. See the Class loader and classpath considerations section in the information center for more information the classpath.
Related concepts:
Class loader and classpath considerations
Related tasks:
Configuring
Administering

CWOBJ1015I

OBJECTTRANSFORMER_PROBLEM_CWOBJ1015=CWOBJ1015I: Preceeding {0} message may be caused by an incorrect application implementation of the ObjectTransformer or Serializable interface

Explanation

If an application implementation of ObjectTransformer or Serializable interface is incorrect, a serialization error will occur on the server when processing a message from a client.

User response

Check the exception message to determine the problem. See the ObjectTransformer interface best practices section in the information center for more information the ObjectTransformer interface.
Related concepts:
Serializer programming overview
Related tasks:

Configuring
Administering

CWOBJ1016E

PROPERTY_FILE_DOES_NOT_EXIST_CWOBJ1016E=CWOBJ1016E: The property file {0} does not exist: {1}.

Explanation

The property file does not exist in the system. It will be ignored.

User response

Specify a valid property file.
Related tasks:
Configuring
Administering
Related reference:
Server properties file
Client properties file

CWOBJ1118I

DCS_CWOBJ1118=CWOBJ1118I: ObjectGrid Server Initializing [Cluster: {0} Server: {1}].

Explanation

The ObjectGrid cluster member is initializing.

User response

No action is required.
Related tasks:
Configuring
Administering
Controlling the lifecycle of an ObjectGrid with the ObjectGridManager interface

CWOBJ1119I

CLIENT_CWOBJ1119=CWOBJ1119I: ObjectGrid client failed to connect to host: {0} port: {1}.

Explanation

ObjectGrid client failed to connect.

User response

No action is required.
Related tasks:
Configuring
Administering
Controlling the lifecycle of an ObjectGrid with the ObjectGridManager interface

CWOBJ1120I

CLIENT_CWOBJ1120=CWOBJ1120I: ObjectGrid Client connected successfully to host: {0} port: {1}.

Explanation

ObjectGrid Client connected successfully.

User response

No action is required.
Related tasks:
Configuring

Administering
Controlling the lifecycle of an ObjectGrid with the ObjectGridManager interface

CWOBJ1121W

TIMEOUT_DURING_SHUTDOWN_WORK_LEFT_CWOBJ1121W=CWOBJ1121W: Timeout during shutdown while waiting for work items to complete. Work items left: {0}

Explanation

During shutdown, the server attempts to wait for work items to complete. Work items could include shard movement off of the server. If they do not complete before timeout, the server will complete shutdown and any remaining work items will failover instead.

User response

The user should verify that the other servers are running normally and that shard placement is correct after the server completes shutdown.
Related tasks:
Configuring
Administering
Controlling the lifecycle of an ObjectGrid with the ObjectGridManager interface

CWOBJ1122W

TIMEOUT_DURING_SHUTDOWN_SHARDS_LEFT_CWOBJ1122W=CWOBJ1122W: Time out while waiting for shards to be moved off server. Shards left: {0}

Explanation

During shutdown, the servers coordinate moving shards similar to when servers are added and shards are rebalanced (if running FIXED_PARTITIONS placement). If all the shards do not move before the timeout, they will failover instead.

User response

The user should verify that the other servers are running normally and that shard placement is correct after the server completes shutdown.
Related tasks:
Configuring
Administering
Controlling the lifecycle of an ObjectGrid with the ObjectGridManager interface

CWOBJ1123W

SERVER_DISCONNECTED_FROM_CATALOG_SERVER_CWOBJ1123W=CWOBJ1123W: Server was disconnected from the primary catalog service and cannot be reconnected.

Explanation

The server was either rejected from the catalog service or received an error from the primary catalog service and cannot recover. It will be isolated from the catalog service and other servers. This is usually due to network problems.

User response

Restart the server.
Related tasks:
Configuring
Administering
Controlling the lifecycle of an ObjectGrid with the ObjectGridManager interface

CWOBJ1124W

DIFFERENT_CATALOG_SERVER_TIMESTAMPS_CWOBJ1124W=CWOBJ1124W: The container server database timestamps are normalized with different catalog servers {0} and {1}. Make sure the clocks of these two servers are synchronized.

Explanation

The container server database timestamps are normalized with different catalog servers. This normally happens when a catalog server fails over when container servers are started.

User response

Synchronize the time clocks between all the catalog servers. If the clocks are far apart, an out of date primary shard might be chosen.
Related tasks:
Configuring
Administering

CWOBJ1125W

CONTAINER_NOT_REGISTERED_CWOBJ1125W=CWOBJ1125W: The server was unable to register container, {0}, with the catalog server due to an exception. {1}

Explanation

The server timed out or received an error from the primary catalog service while starting containers to initiate shard placement. This is usually due to network problems or heavy workload on the catalog server.

User response

Restart the server.
Related tasks:
Configuring
Administering
Controlling the lifecycle of an ObjectGrid with the ObjectGridManager interface

CWOBJ1126I

CLIENT_CONNECT_CWOBJ1126=CWOBJ1126I: The ObjectGrid client has connected to the {0} grid in the {1} domain using connection {2}.

Explanation

A client ObjectGrid instance connected successfully.

User response

No action is required.
Related tasks:
Configuring
Administering
Controlling the lifecycle of an ObjectGrid with the ObjectGridManager interface

CWOBJ1127I

CLIENT_DISCONNECT_CWOBJ1127=CWOBJ1127I: The ObjectGrid client connection {0} has disconnected from the {1} domain. ObjectGrids used by this connection were {2}.

Explanation

The ObjectGrid client disconnected successfully.

User response

No action is required.
Related tasks:
Configuring
Administering
Controlling the lifecycle of an ObjectGrid with the ObjectGridManager interface

CWOBJ1128I

CLIENT_CACHE_MAPS_CWOBJ1128=CWOBJ1128I: The client cache is enabled for maps {0} on the {1} ObjectGrid.

Explanation

The provided list of maps has a client-side cache enabled.

User response

No action is required.
Related tasks:
Configuring
Administering
Controlling the lifecycle of an ObjectGrid with the ObjectGridManager interface

CWOBJ1129W

SHARDS_LEFT_ON_TERMINATE_CWOBJ1129=CWOBJ1129W: Some of the shards were not removed before the container terminate completed on {0} container. Shards left: {1}

Explanation

A call to terminate the container attempted to destroy all of the shards on the container, but shards still remained on the container. A shard destroy could have failed or a new shard could have been placed during the terminate. Any shards placed on the terminating container will be failed over to other containers.

User response

Verify the placement using the administrative tools, for example review placementStatus data.
Related tasks:
Configuring
Administering
Enabling logging
Analyzing log and trace data

CWOBJ1130W

COMM_ERROR_WITH_SHARD_CWOBJ1130=CWOBJ1130W: Communication with the partition with the domain:grid:mapSet:partitionId {0} failed with an Object Request Broker (ORB) exception communicating with {1} at {2}.

Explanation

The shard access could not complete successfully because of a CORBA-level exception.

User response

If ORB and eXtreme Scale tuning parameters are specified and allowable retries succeed, no action is required. Otherwise, an exception results. The application must perform some sort alternative data access or general failure handling.
Related tasks:
Configuring
Administering

CWOBJ1131I

RECEIVED_ROUTE_UPDATE_FROM_CONTAINER_CWOBJ1131=CWOBJ1131I: An updated routing entry for the partition with domain:grid:mapSet:partitionId:epoch {0} was sent to this client.

Explanation

Shard movement between container servers resulted in updates to the routing entry for the partition.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ1132I

RECEIVED_ROUTE_UPDATE_FROM_CATALOG_CWOBJ1132=CWOBJ1132I: An updated routing entry for domain:grid:epoch {0} was obtained from the catalog server.

Explanation

Some form of shard movement between container servers resulted in updates to the routing table for the data grid.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ1133W

RETRY_COMM_WITH_SHARD_CWOBJ1133=CWOBJ1133W: After an initial communication failure, an attempt to access the partition for domain:grid:mapSet:partitionId {0} on the {1} container server at {2} was successful.

Explanation

After a communication exception, shard access was tried again and succeeded.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ1203W

CLIENT_RESPONSE_TIMEOUT_CWOBJ1203W=CWOBJ1203W: Received a timeout event from the server for transaction: {0}

Explanation

Client did not receive expected response message from the server within a configured timeout limit.

User response

Look for prior messages that may explain the timeout. If none found, try increasing the timeout limit.
Related tasks:
Configuring
Administering
Configuring request retry timeout values

CWOBJ1204W

UNKNOWN_MESSAGE_TYPE_CWOBJ1204W=CWOBJ1204W: Received a message of unknown message type. The message is: {0}

Explanation

An unexpected internal error was detected and a message cannot be processed. There may be an interoperability problem.

User response

Review the WebSphere eXtreme Scale internet support web site for a similar problem or contact IBM Software Support.
Related tasks:
Configuring
Administering
Contacting IBM support

CWOBJ1207W

CONFIG_PROPERTY_UNSUPPORTED_CWOBJ1207W=CWOBJ1207W: The property {0} on plug-in {1} is using an unsupported property type.

Explanation

Java primitives and their java.lang counterparts are the only supported property types. This includes java.lang.String.

User response

Check the property type and change it to one of the supported types.

Related concepts:
System APIs and plug-ins
Plug-ins overview
Related tasks:
Configuring
Administering

CWOBJ1208W

CONFIG_PLUGIN_UNSUPPORTED_CWOBJ1208W=CWOBJ1208W: The specified plug-in type {0} is not supported.

Explanation

This type of plug-in is unsupported.

User response

Add one of the supported plug-in types. See the Introduction to plug-ins section in the information center for more information on plug-ins.
Related concepts:
System APIs and plug-ins
Plug-ins overview
Related tasks:
Configuring
Administering

CWOBJ1209E

UNABLE_TO_ACTIVATE_SHARD_CWOBJ1209E=CWOBJ1209E: Unable to activate shard for ObjectGrid {0}, domain {1}, map set {2}, partition {3}, shard type {4} ({0}:{2}:{3}) due to exception: {5}

Explanation

The ObjectGrid container attempted to activate a shard, but encountered an unexpected exception. The catalog service will automatically try to start the shard on another container, if available.

User response

Examine the exception included in this message and any first failure data capture (FFDC) logs, correct the error, and restart the ObjectGrid container.
Related tasks:
Configuring
Administering
Enabling logging
Analyzing log and trace data

CWOBJ1210E

UNABLE_TO_RETURN_SHARD_CWOBJ1210E=CWOBJ1210E: Unable to return shard for ObjectGrid {0}, domain {1}, map set {2}, partition {3}, shard type {4} ({1}:{0}:{2}:{3}) due to exception: {5}

Explanation

The ObjectGrid container attempted to return the requested shard reference, but encountered an unexpected exception.

User response

Examine the exception included in this message and any first failure data capture (FFDC) logs, correct the error, and restart the ObjectGrid container.
Related tasks:
Configuring
Administering
Enabling logging
Analyzing log and trace data

CWOBJ1211E

ERROR_OG_PMI_CREATE_FAILED_CWOBJ1211E=CWOBJ1211E: The Performance Monitoring Infrastructure (PMI) creation of {0} failed. The exception is {1}.

Explanation

An attempt to create ObjectGrid PMI failed.

User response

Examine the exception message and the first failure data capture (FFDC) log for configuration problems and restart the server.
Related tasks:
Configuring
Administering
Enabling logging
Analyzing log and trace data

CWOBJ1212I

PMI_NOT_FOUND=CWOBJ1212I: The WebSphere Application Server Performance Monitoring Infrastructure (PMI) cannot be found.

Explanation

The WebSphere Application Server PMI cannot be found. This is expected if eXtreme Scale is not running in WebSphere Application Server.

User response

If running standalone eXtreme Scale, no action is required. If eXtreme Scale is running in WebSphere Application Server, look for any configuration errors. Otherwise, contact IBM Software Support.
Related tasks:
Configuring
Administering
Monitoring

CWOBJ1213I

SERVER_RECONNECTED_WITH_CATALOG_SERVER_CWOBJ1213I=CWOBJ1213I: Server was disconnected from the primary catalog server but was able to reconnect.

Explanation

Usually because of network problems or garbage collection issues, the server was removed from the list of servers managed by the primary catalog server. However, the network recovered in a timely enough fashion for the server to be reincluded. It is no longer isolated from the catalog servers and other servers.

User response

No action is required. However, verify the integrity of your Java virtual machine (JVM) garbage collection.
Related tasks:
Configuring
Administering
Controlling the lifecycle of an ObjectGrid with the ObjectGridManager interface

CWOBJ1214I

PLACEMENT_BALANCE_STATUS_CWOBJ1214I=CWOBJ1214I: Shard balancing for ObjectGrid {0}:{1} is {2}.

Explanation

The state of shard balancing and placement for the listed grid.

User response

No action is requred. Use the suspendBalance or resumeBalance mBean or xscmd interface to change the state of the grid.
Related tasks:
Configuring
Administering
Enabling logging
Analyzing log and trace data

CWOBJ1215I

TP_CWOBJ1215=CWOBJ1215I: ObjectGrid Transaction Propagation Event Listener is initializing [ObjectGrid {0}].

Explanation

This informational message indicates that the ObjectGrid Transaction Propagation Event Listener is initializing.

User response

No action is required.
Related concepts:
JMS event listener
Related tasks:
Configuring
Administering

CWOBJ1216I

TP_CWOBJ1216=CWOBJ1216I: ObjectGrid Transaction Propagation Event Listener is initialized [ObjectGrid {0}].

Explanation

ObjectGrid Transaction Propagation Event Listener Initialized.

User response

No action is required.
Related concepts:
JMS event listener
Related tasks:
Configuring
Administering

CWOBJ1217I

TP_CWOBJ1217=CWOBJ1217I: ObjectGrid Transaction Propagation Service Point Initialized [ObjectGrid {0}].

Explanation

This informational message indicates that the ObjectGrid Transaction Propagation Event Listener is initialized.

User response

No action is required.
Related concepts:
JMS event listener
Related tasks:
Configuring
Administering

CWOBJ1218E

TP_CWOBJ1218=CWOBJ1218E: ObjectGrid Transaction Propagation Event Listener failure occurred [ObjectGrid {0} Exception message {1}].

Explanation

ObjectGrid runtime environment encountered an ObjectGrid Transaction Propagation failure.

User response

Review the exception, resolve the error and retry the operation.
Related tasks:
Configuring
Administering

CWOBJ1219E

TP_CWOBJ1219=CWOBJ1219E: ObjectGrid Transaction Propagation Service End Point failure occurred [ObjectGrid {0} Exception message {1}].

Explanation

ObjectGrid runtime environment encountered an ObjectGrid Transaction Propagation Service End Point failure.

User response

Review the exception, resolve the error and retry the operation.
Related tasks:
Configuring
Administering

CWOBJ1220E

TRANPROPLISTENER_UNSUPPORTED_CWOBJ1220=CWOBJ1220E: ObjectGrid Transaction Propagation Service is not supported in this environment.

Explanation

ObjectGrid Transaction Propagation Service is not supported on z/OS or the standalone ObjectGrid server environment.

User response

Do not use ObjectGrid Transaction Propagation Service on z/OS or in the standalone ObjectGrid server environment
Related tasks:
Configuring
Administering

CWOBJ1221E

PLUGIN_FAILED_CWOBJ1221=CWOBJ1221E: The plug-in implemented by class {0} failed during a call to method {1}, the exception is: {2}

Explanation

The plug-in handles all exceptions during processing.

User response

Review and repair the plug-in implementation, or remove it from the grid configuration.
Related concepts:
System APIs and plug-ins
Plug-ins overview
Related tasks:
Configuring
Administering

CWOBJ1222E

PLUGIN_INCORRECT_CWOBJ1222=CWOBJ1222E: The plug-in implemented by class {0} is in an incorrect state or has an incorrect status as indicated by method {1}.

Explanation

The plug-in records its state or status based on the life cycle events that are delivered to it, and the plug-in retains the state as appropriate.

User response

Review and repair the plug-in implementation, or remove it from the grid configuration.
Related concepts:
System APIs and plug-ins
Plug-ins overview
Related tasks:
Configuring
Administering

CWOBJ1223E

INVALID_QUERY_SCHEMA_CONFIG_CWOBJ1223=CWOBJ1223E: An invalid ObjectQuery schema configuration is defined. The following maps have both an ObjectQuery configuration and a serializer or an entity configuration: {0}

Explanation

A query schema can only be defined for maps that do not have an entity or a MapSerializerPlugin plug-in associated with it.

User response

Complete one or more of the following actions: Remove the entity definition from the entity configuration file, remove the MapSerializerPlugin plug-in from the specified backing map configurations, remove the ObjectQuery map definitions from the query schema in the ObjectGrid descriptor XML file, or remove the QueryConfig configuration object.
Related tasks:
Configuring
Administering

CWOBJ1224I

RESTART_EXITING_JVM_CWOBJ1224I=CWOBJ1224I: The JVM process is ending because a replacement JVM has started.

Explanation

A new, replacement JVM was requested. After the replacement JVM starts, the previous JVM processes end.

User response

No user action is required.
Related tasks:
Configuring
Administering
Controlling the lifecycle of an ObjectGrid with the ObjectGridManager interface

CWOBJ1225E

RESTART_JVM_FAILED_CWOBJ1225E=CWOBJ1225E: The JVM did not restart.

Explanation

A new, replacement JVM could not be started.

User response

See the logs to determine what exceptions were created.
Related tasks:
Configuring
Administering
Enabling logging
Analyzing log and trace data

CWOBJ1226E

RESTART_PARENT_TIMEOUT_CWOBJ1226E=CWOBJ1226E: The parent JVM did not terminate within the timeout period ({0} ms). Discontinuing with startup of the child JVM.

Explanation

The child JVM waited for the parent JVM to terminate but the parent JVM took longer than expected.

User response

No user action is required.
Related tasks:
Configuring
Administering
Configuring request retry timeout values

CWOBJ1227I

SERVER_REBOOT_TO_CONNECT_WITH_CATALOG_SERVER_CWOBJ1227I=CWOBJ1227I: Server was disconnected from the primary catalog server so we will restart it to reconnect.

Explanation

Usually because of network problems or garbage collection issues, the server was removed from the list of servers managed by the primary catalog server. The server is going to restart itself in an attempt to reconnect with the catalog server.

User response

No action required, however verification of your network health and JVM GC health could be wise.
Related tasks:
Configuring
Administering
Controlling the lifecycle of an ObjectGrid with the ObjectGridManager interface

CWOBJ1228I

CONTAINER_RECONNECT_IGNORED_CWOBJ1228I=CWOBJ1228I: The server ignored a request to reconnect its containers because a previous reconnect request was just completed.

Explanation

This can be the result of both the local and a remote server concurrently trying to recover from a network outage.

User response

No action required, however verifying the containers residing on this server are operational could be wise.
Related tasks:
Configuring
Administering

CWOBJ1250W

UPGRADE_CATALOG_CWOBJ1250=CWOBJ1250W: A client with a version greater than or equal to {0} is connecting to a catalog service that has a version less than {0}. The catalog service must be upgraded before clients are upgraded.

Explanation

The upgrade path for WebSphere eXtreme Scale requires the catalog service to be upgrade first. The client connecting to the objectGrid is at a higher level than the catalog service.

User response

Refer to the eXtreme Scale documentation on upgrading the product and upgrade the catalog service.
Related tasks:
Configuring
Administering

CWOBJ1251I

QUORUM_ENABLED_CWOBJ1251I=CWOBJ1251I: Quorum is enabled for the catalog service.

Explanation

The catalog service has quorum detection enabled and has successfully reached a quorum. The catalog service can successfully process requests.

User response

No action is required. This is the normal state of a quorum-enabled catalog service grid.
Related tasks:
Configuring
Administering
Configuring the quorum mechanism

CWOBJ1252I

QUORUM_DISABLED_CWOBJ1252I=CWOBJ1252I: Quorum is disabled for the catalog service.

Explanation

The catalog service does not have quorum enabled. The catalog service will continue to process requests if the catalog service grid is partitioned because of a network failure.

User response

No action is required.
Related tasks:
Configuring
Administering
Configuring the quorum mechanism

CWOBJ1253I

QUORUM_OVERRIDE_CWOBJ1253I=CWOBJ1253I: Quorum has been overridden for the catalog service.

Explanation

The catalog service has quorum enabled and did not have all servers available to make a quorum. The quorum has been overridden to force a quorum.

User response

Restart the failed catalog service as soon as practical to bring the quorum state back to normal.
Related tasks:
Configuring
Administering
Configuring the quorum mechanism

CWOBJ1300I

PMA_CWOBJ1300=CWOBJ1300I: Adapter successfully initialized ObjectGrid.

Explanation

Adapter successfully initialized ObjectGrid.

User response

No action is required.
Related tasks:
Configuring
Administering
Controlling the lifecycle of an ObjectGrid with the ObjectGridManager interface

CWOBJ1301E

PMA_CWOBJ1301=CWOBJ1301E: Adapter failed to initialize ObjectGrid. Exception occurred {0}.

Explanation

The adapter's attempt to initialize ObjectGrid failed.

User response

Review the exception, resolve the error and retry the operation. If the com.ibm.ws.pmcache.config property has a badly formed URL, fix the property. If there is an ObjectGridException, review the exception message or see the chained exception in the exception stack for the root cause.
Related tasks:
Configuring
Administering
Enabling logging
Analyzing log and trace data

CWOBJ1302I

PMA_CWOBJ1302=CWOBJ1302I: Adapter stopped.

Explanation

Adapter stopped.

User response

No action is required.
Related tasks:
Configuring
Administering
Controlling the lifecycle of an ObjectGrid with the ObjectGridManager interface

CWOBJ1303I

PMA_CWOBJ1303=CWOBJ1303I: Adapter started.

Explanation

Adapter started.

User response

No action is required.
Related tasks:
Configuring
Administering
Controlling the lifecycle of an ObjectGrid with the ObjectGridManager interface

CWOBJ1304I

SECURITY_ENABLED_CWOBJ1304=CWOBJ1304I: Security is enabled.

Explanation

Security is enabled.

User response

No action is required.
Related tasks:
Configuring
Administering
Security

CWOBJ1305I

SECURITY_DISABLED_CWOBJ1305=CWOBJ1305I: Security is disabled.

Explanation

Security is disabled.

User response

No action is required.
Related tasks:
Configuring
Administering
Security

CWOBJ1306W

CANNOT_RETRIEVE_CLIENT_CERTS_CWOBJ1306=CWOBJ1306W: Cannot retrieve the client certificates from the SSL socket.

Explanation

The runtime cannot retrieve the client certificates from the SSL socket.

User response

If you are running eXtreme Scale in WebSphere Application Server, use the administrative console to review your SSL configuration. If you are running eXtreme Scale standalone, review your SSL configuration in the property files for your servers and clients. Verify the SSL configuration settings, including the location of the key store, trust store, passwords and transportType. Check the client and server side to ensure that they match and have complementary transportTypes. See the Transport layer security and secure sockets layer section in the information center for more information.
Related tasks:
Configuring
Administering
Security

CWOBJ1307I

OBJECTGRID_SECURITY_ENABLED_CWOBJ1307=CWOBJ1307I: Authorization security for ObjectGrid {0} is enabled.

Explanation

Authorization or Java 2 security is enabled for the specified ObjectGrid instance.

User response

No action is required.
Related tasks:
Configuring
Administering
Security

CWOBJ1308I

OBJECTGRID_SECURITY_DISABLED_CWOBJ1308=CWOBJ1308I: Security of the ObjectGrid instance {0} is disabled.

Explanation

Security is disabled for the specified ObjectGrid instance.

User response

No action is required.
Related tasks:
Configuring
Administering
Security

CWOBJ1309E

OBJECTGRID_CONNECT_TOKEN_CREATION_CWOBJ1309=CWOBJ1309E: Unexpected error occured in the connect token creation: {0}

Explanation

An unexpected error occured in the connection token creation.

User response

Check the security configuration and verify the secureToken settings and the authenticationSecret. Verify that they match on the servers and the catalog service.
Related tasks:
Configuring
Administering
Security

CWOBJ1310E

OBJECTGRID_CONNECT_TOKEN_VALIDATION_CWOBJ1310=CWOBJ1310E: An attempt by another process to connect to this process through the core group transport has been rejected. The connecting process provided a source core group name of {0}, a target of {1}, a member name of {2} and an IP address of {3}. The error message is {4}.

Explanation

The High Availability Manager has rejected a connection attempt.

User response

This may be a connection attempt from an unauthorized party.
Related tasks:
Configuring
Administering

CWOBJ1311W

IGNORE_CREDENTIAL_GENERATOR_PROPS=CWOBJ1311W: The credentialGeneratorProps setting is ignored since the credentialGeneratorClass value is not provided.

Explanation

The credentialGeneratorProps setting is only used if the credentialGeneratorClass value is provided.

User response

Set the credentialGeneratorClass in the client property file if you plan to customize the credential generator.
Related tasks:
Configuring
Administering
Security

CWOBJ1312W

EXPIRED_CREDENTIAL_EXCEPTION=CWOBJ1312W: The credential expired. The exception message is {0}.

Explanation

The credential expired. Check the exception message for the reason that it expired.

User response

ObjectGrid will try to re-generate a credential. If the problem persists, check the exception messages for the reason that the credential expired.
Related tasks:
Configuring
Administering
Security

CWOBJ1313W

CUSTOM_SECURE_TOKEN_MANAGER_CLASS_IGNORED=CWOBJ1313W: The customSecureTokenManagerClass setting is ignored since the provided customSecureTokenManagerType value is not "custom".

Explanation

In order to use the custom secure token manager, the customSecureTokenManagerType property has to be set to "custom".

User response

Set the customSecureTokenManagerType value to "custom" to use the custom secure token manager. To avoid this warning, remove the customSecureTokenManagerClass value.
Related tasks:
Configuring
Administering
Security

CWOBJ1314W

IGNORE_CREDENTIAL_GENERATOR_CLASS_CWOBJ1314W=CWOBJ1314W: The credentialGeneratorClass property with value "{0}" is being overridden.

Explanation

The credentialGeneratorClass property can be overridden dynamically by certain eXtreme Scale components that require specific credentialGeneratorClass implementations.

User response

No action is required.
Related tasks:
Configuring
Administering
Security

CWOBJ1315I

DYNAMIC_CREDENTIAL_GENERATOR_CLASS_CWOBJ1315I=CWOBJ1315I: The credentialGeneratorClass property was set dynamically to a value of "{0}".

Explanation

The credentialGeneratorClass property can be set dynamically by certain eXtreme Scale components that require specific credentialGeneratorClass implementations.

User response

No action is required.
Related tasks:
Configuring
Administering
Security

CWOBJ1316W

CLIENT_SECURITY_ENABLED_SERVER_SECURITY_DISABLED_CWOBJ1316W=CWOBJ1316W: This non-secure server received a client request containing credential information. The credential information will be ignored by this server.

Explanation

The server started in this JVM is a non-secure server. Any credential information sent from a client or connecting server is ignored. This normally indicates a security mismatch between the client and server.

User response

Make sure the server is intended to be non-secure, or the client is intended to be secure. Correct the client or server security configuration if necessary.
Related tasks:
Configuring
Administering
Security

CWOBJ1317W

UNSUPPORTED_ENCODE_ALGORITHM_CWOBJ1317W=CWOBJ1317W: The property {0} is encoded with an unsupported encoding algorithm "{1}". The property will be ignored.

Explanation

Currently, only XOR encoding algorithm is supported.

User response

Use the XOR encoding algorithm. Use FilePasswordEncoder.bat or FilePasswordEncoder.sh to encode the properties.
Related tasks:
Configuring
Administering
Security

CWOBJ1318I

SSL_TRANSPORT_TYPE_ENABLED_CWOBJ1318I=CWOBJ1318I: Transport layer security configuration is set to {0}.

Explanation

The current setting for transport layer security. Available settings are listed in the property file articles or the SecurityConstants API in the information center.

User response

No action is required.
Related tasks:
Configuring
Administering
Security

CWOBJ1319E

SECURITY_NOT_PROVIDED_ON_STOP_CWOBJ1319=CWOBJ1319E: The exception {0} occured attempting to stop the server. Verify that a client property file was provided with the stop command including the required security settings.

Explanation

To stop a server, the correct SSL configuration properties and Credential authentication configuration properties as needed.

User response

If security was enabled to start start the server, provide a client property file to stop the server. See the Starting and stopping secure eXtreme Scale servers and Client properties file articles in the information center for more information.
Related tasks:
Configuring
Administering
Security

CWOBJ1320E

FILEPASSWORDENCODER_ERROR_CWOBJ1320E=CWOBJ1320E: An error occured while processing the FilePasswordEncoder request: {0} {1}

Explanation

An error occured which prevents the FilePasswordEncoder operation from completing successfully.

User response

Review the provided error message and perform any recommended actions.
Related tasks:
Configuring
Administering
Security

CWOBJ1321I

FILEPASSWORDENCODER_INFO_CWOBJ1321I=CWOBJ1321I: FilePasswordEncoder informational message: {0} {1}

Explanation

This is an informational message pertaining to a FilePasswordEncoder operation.

User response

No action is required.
Related tasks:
Configuring
Administering
Security

CWOBJ1322E

CLIENTSECURITYCONTEXT_ERROR_CWOBJ1322E=CWOBJ1322E: Internal runtime error occured for client request on thread {0}. Security Context Map information is {1}

Explanation

Internal error in ObjectGrid runtime.

User response

Contact IBM Software Support.
Related tasks:
Configuring
Administering

CWOBJ1400W

MULTIPLE_JAR_FILE_CWOBJ1400W=CWOBJ1400W: Detected multiple ObjectGrid runtime JAR files in the JVM. Using multiple ObjectGrid runtime JAR files may cause problems.

Explanation

Usually only one ObjectGrid runtime JAR should be found in a JVM.

User response

Use the appropriate ObjectGrid runtime JAR file for your configuration. Leaving other JAR files could have unpredictable results such as old classes being used at runtime.
Related tasks:
Configuring
Administering
Installing

CWOBJ1401E

WRONG_JAR_FILE_CWOBJ1401E=CWOBJ1401E: Detected a wrong ObjectGrid runtime JAR file for this configuration. Detected configuration is {0}. Expected JAR file is {1}.

Explanation

Each ObjectGrid runtime JAR file corresponds to a particular supported configuration.

User response

Use the appropriate ObjectGrid runtime JAR for your configuration. Review the article Installing stand-alone WebSphere eXtreme Scale in the information center for more information.
Related tasks:
Configuring
Administering
Installing

CWOBJ1402E

MISSING_CONNECTION_LINK_CALLBACK_CWOBJ1402E=CWOBJ1402E: ObjectGrid connection link callback not found for id: {0}

Explanation

Internal error in ObjectGrid runtime.

User response

Contact IBM Software Support.
Related tasks:
Configuring
Administering

CWOBJ1403E

INVALID_RESOURCE_CWOBJ1403E=CWOBJ1403E: The resource specified is invalid: {0}

Explanation

The specified resource may not exist, may not be accessible, or it may not be in the appropriate format.

User response

Verify that the resource exists, is accessible, and is in the correct format.
Related tasks:
Configuring
Administering

CWOBJ1504E

CANNOT_PROCESS_REPLICA_CHANGES_CWOBJ1504=CWOBJ1504E: An exception occurred when attempting to process the LogSequences for replica ({0}): {1}.

Explanation

An exception occurred on the replica while it was processing data.

User response

If the exception indicates that there is a configuration problem on the replica, correct the problem and restart the replica server. Otherwise, the replica will rollback the failed transaction.
Related tasks:
Configuring
Administering
Tracking map updates by an application
Related reference:
LogSequence interface
Deprecated properties and APIs

CWOBJ1505E

MORE_THAN_ONE_PRIMARY_RESPONSE_CWOBJ1505=CWOBJ1505E: More than one replication group member reported back as the primary. Only one primary can be active. ({0}).

Explanation

While pushing out the routing table, there is more than one replication group member reported as primary.

User response

The routing table will not be updated at this time. Check the current list of primaries to ensure there are no duplicates using the logs. See the Logs and Trace section in the information center for JVM log location, or the ManagementGateway Interface.
Related tasks:
Configuring
Administering
Enabling logging
Analyzing log and trace data

CWOBJ1506E

POSSIBLE_NETWORK_PARTITION_CWOBJ1506=CWOBJ1506E: More than one primary replication group member exists in this group ({1}). Only one primary can be active. ({0}).

Explanation

There may have been a temporary network partition or brown out condition and now there is more than one primary active.

User response

No additional placement or routing updates will be done. Restart the servers that were network partitioned.
Related tasks:
Configuring
Administering

CWOBJ1508E

CANNOT_SEND_MESSAGE_CWOBJ1508=CWOBJ1508E: An exception occurred when attempting to send message ({0}) from sender ({1}) to receiver ({2}): {3}.

Explanation

A problem occurred while attempting to send a message between replication group members.

User response

Review the error message listed. If there is a configuration problem, correct it and restart the replica server. Otherwise, verify the correct placement of primaries and replicas using the logs. See the Logs and Trace section in the information center for JVM log location, or the ManagementGateway Interface.
Related tasks:
Configuring
Administering
Enabling logging
Analyzing log and trace data

CWOBJ1509E

CANNOT_SERIALIZE_MESSAGE_CWOBJ1509=CWOBJ1509E: An exception occurred when attempting to serialize message ({0}): {1}.

Explanation

An exception occurred while serialing an internal message.

User response

Review the error message listed. If there is a configuration problem, correct it and restart the replica server.
Related tasks:
Configuring
Administering

CWOBJ1510E

CANNOT_DESERIALIZE_MESSAGE_CWOBJ1510=CWOBJ1510E: An exception occurred when attempting to inflate message ({0}): {1}.

Explanation

An exception occurred while inflating an internal message.

User response

Review the error message listed. If there is a configuration problem, correct it and restart the replica server.
Related tasks:
Configuring
Administering
Controlling the lifecycle of an ObjectGrid with the ObjectGridManager interface

CWOBJ1511I

OPEN_FOR_BUSINESS_CWOBJ1511=CWOBJ1511I: {0} ({1}) is open for business.

Explanation

Specified replication group member is now ready to accept requests.

User response

No action is required.
Related tasks:
Configuring
Administering
Controlling the lifecycle of an ObjectGrid with the ObjectGridManager interface

CWOBJ1513E

SYNCH_REPLICATION_FAILED_CWOBJ1513=CWOBJ1513E: Synchronous replication failed on {0} ({1}). This member is no longer active.

Explanation

A problem was encountered that prevented synchronous replication from successfully completing.

User response

Review previous messages in the log. See the Logs and Trace section in the information center for JVM log location to help diagnose the problem. Restart the specified server to replace the removed replica.
Related tasks:
Configuring
Administering
Enabling logging
Analyzing log and trace data

CWOBJ1514I

PRIMARY_DOWNGRADED_CWOBJ1514=CWOBJ1514I: Primary ({0}) is being downgraded to either a replica or standby.

Explanation

This is not a normal operation, but ObjectGrid processing can continue.

User response

Verify the correct placement of primaries and replicas using the logs. See the Logs and Trace section in the information center for JVM log location, or the ManagementGateway Interface.
Related tasks:
Configuring
Administering
Enabling logging
Analyzing log and trace data

CWOBJ1515I

MIN_CONFIG_NOT_MET_CWOBJ1515=CWOBJ1515I: Minimum configuration requirements not satisfied for replication group ({0}).

Explanation

The necessary primary and replica configuration requirements were not met with the recent replication group member change.

User response

Wait for additional resources to be started and recognized for this configuration.
Related concepts:
Shard life cycles
Related tasks:
Configuring
Administering
Configuring zones for replica placement

CWOBJ1516E

CANNOT_ACTIVATE_OBJECTGRID_CWOBJ1516=CWOBJ1516E: An exception occurred when attempting to activate the replication process for ObjectGrid ({0}): {1}.

Explanation

While attempting to start a primay replication group member, an exception occurred during the activation processing.

User response

The primary will continue to run. Verify the correct placement of primaries and replicas using the logs. See the Logs and Trace section in the information center for JVM log location, or the ManagementGateway Interface. Restart the server if the primary continues to have errors.
Related tasks:
Configuring
Administering
Enabling logging
Analyzing log and trace data

CWOBJ1518E

CANNOT_COMMIT_REPLICA_CHANGES_CWOBJ1518=CWOBJ1518E: An exception occurred when attempting to commit replica transaction ({0}) for primary transaction ({1}) on Replica ({2}): {3}.

Explanation

There was an exception while committing data on the replica.

User response

If the exception indicates that there is a configuration problem on the replica, correct the problem and restart the replica server. Otherwise, the replica will rollback the failed transaction.
Related concepts:
Shard life cycles
Related tasks:
Configuring
Administering
Configuring zones for replica placement

CWOBJ1519E

CANNOT_ROLLBACK_REPLICA_CHANGES_CWOBJ1519=CWOBJ1519E: An exception occurred when attempting to rollback the LogSequences for replica ({0}): {1}

Explanation

After a transaction failed to commit on a replica, the rollback also failed on the replica.

User response

If the exception indicates that there is a configuration problem on the replica, correct the problem and restart the replica server.
Related concepts:
Shard life cycles
Related tasks:
Configuring
Administering
Configuring zones for replica placement

CWOBJ1524I

LISTENER_REREGISTER_CWOBJ1524=CWOBJ1524I: Replica listener {0} must re-register with the primary. Reason: {1}

Explanation

The replica will deregister and reregister the primary. It will get a new snapshot of the data and then continue processing new transactions. This happens when an error occurs on the replica.

User response

If the replica enters peer mode successfully, no immediate action is neccessary (CWOBJ1526I). To look for the cause, check on the reason for the reregister. If there is an error inflating the key, verify that the server for the replica has the correct classpath and the correct code to deserialize the object. Also, verify that the hashcode() and equals() methods are correct for the key or keys being used if a custom key is used. Otherwise, contact IBM Software Support.
Related concepts:
Shard life cycles

Related tasks:
Configuring
Administering
Configuring zones for replica placement

CWOBJ1525I

CHECKPRELOADSTATE_EXCEPTION_CWOBJ1525=CWOBJ1525I: A ReplicaPreloadController ({0}) for map {1} threw an unexpected exception in method checkPreloadState {2}

Explanation

When promoting from replica to primary an exception occurred when the ReplicaPreloadController was called to determine the state of the replica. The exception is ignored and preload is performed on the map.

User response

Examine the stack trace to determine the cause of the problem. Fix the problem in your implementation or contact IBM Software Support if the problem does not appear to be in your implementation.
Related concepts:
Shard life cycles
Related tasks:
Configuring
Administering
Configuring zones for replica placement

CWOBJ1526I

ENTERING_PEER_MODE_CWOBJ1526=CWOBJ1526I: Replica {0} entering peer mode after {1} seconds, replicating from primary on {2}

Explanation

This is an informational message on how long it took for a replica to enter peer mode where both primary and replica have the same data.

User response

No action is required.
Related concepts:
Shard life cycles
Related tasks:
Configuring
Administering
Configuring zones for replica placement

CWOBJ1527W

FAILED_ENTERING_PEER_MODE_CWOBJ1527=CWOBJ1527W: Replica {0} failed to enter peer mode after {1} seconds

Explanation

The replica failed to enter peer mode. Look for additional messages that point to the specific cause of the failure. Possible reasons may include a timeout or data failing to copy from the primary.

User response

Review the action recommended by the specific message for timeout, bad data copy or other reason.
Related concepts:
Shard life cycles
Related tasks:
Configuring
Administering
Configuring zones for replica placement

CWOBJ1531I

CLOSED_FOR_BUSINESS_CWOBJ1531=CWOBJ1531I: {0} ({1}) stopped on this server.

Explanation

Specified shard stopped running on this server and moved to another server if another is available.

User response

No action is required.
Related tasks:
Configuring
Administering
Controlling the lifecycle of an ObjectGrid with the ObjectGridManager interface

CWOBJ1532I

SHARD_TRANSITION_CWOBJ1532=CWOBJ1532I: {0} (moving from server {1} ({2}), promoting {3} to {4}) in transition.

Explanation

Specified shard is in a transitional state. The message indicates the state of the shard, the state the shard will become after transition completes and where the shard was running before it failed over to the current server. For a moving primary, if there is an existing replica, it will be promoted. If there is no replica, a new replica will be placed and promoted. If there are no replicas to promote and no primary to take over from, a new primary will be started. This is indicated by promoting an inactive shard.

User response

No action is required.
Related tasks:
Configuring
Administering
Enabling logging
Analyzing log and trace data

CWOBJ1533E

REPLICA_IN_PEER_MODE_CWOBJ1533=CWOBJ1533E: {0}:{1} is already in peer mode.

Explanation

Specified replica is in peer mode and attempted to enter peer mode again.

User response

Review the logs. See the Logs and Trace section in the information center for JVM log location for any additional error messages. Verify that the replica does not have additional messages after the CWOBJ1533E message. Placement can also be verified using wsadmin or xsAdmin and the PlacementServiceMBean.
Related tasks:
Configuring
Administering
Enabling logging
Analyzing log and trace data

CWOBJ1535E

REPLICA_FAIL_ON_PEER_MODE_BAD_TRAN_CWOBJ1535=CWOBJ1535E: {0}:{1} Replica map failed to enter peer mode. A transaction threw an error while copying data from the primary. Error received: {2}

Explanation

There was an error on the replica while it copied existing data from the primary.

User response

Review the error message received. Also review any other log messages for related errors from the application. If there is an error inflating the key, verify that the server for the replica has the correct classpath. Also, verify that the hashcode() and equals() methods are correct for the key or keys being used if a custom key is used. Restart the servers or use the triggerPlacement method on the PlacementServiceMBean Mbean to replace the replica. See the Life cycle, recovery, and failure events section in the information center for more information on the triggerPlacement method.
Related concepts:
Shard life cycles
Related tasks:
Configuring

Administering
Configuring zones for replica placement

CWOBJ1536E

REPLICA_FAIL_ON_PEER_MODE_ORDERING_CWOBJ1536=CWOBJ1536E: {0}:{1} Replica map failed to enter peer mode. Received incorrect ordering data from the primary, data copy cannot complete.

Explanation

The data that the replica received from the primary was in the incorrect order.

User response

Use the triggerPlacement method on the PlacementServiceMBean Mbean to place the failed replica again. See the Life cycle, recovery, and failure events section in the information center for more information on the triggerPlacement method.
Related tasks:
Configuring
Administering
Enabling logging
Analyzing log and trace data

CWOBJ1537E

REPLICA_FAIL_TO_REREGISTER_CWOBJ1537=CWOBJ1537E: {0} exceeded the maximum number of times to reregister ({1}) without successful transactions.

Explanation

The replica attempted to reregister several times in a row due to errors, but failed to successfully commit any data before the next reregister. The replica will stop attempting to reregister.

User response

Look why the replica had to reregister by finding CWOBJ1524I messages in the logs. See the Logs and Trace section in the information center for JVM log location. Check for configuration or application errors that would prevent the replica from working correctly, such as classpath issues, errors deserializing the data, and so on. Restart the servers or use the triggerPlacement method on the PlacementServiceMBean Mbean to replace the replica. See the Life cycle, recovery, and failure events section in the information center for more information on the triggerPlacement method.
Related tasks:
Configuring
Administering
Enabling logging
Analyzing log and trace data

CWOBJ1538E

PRIMARY_REJECT_SAME_REPLICA_CWOBJ1538=CWOBJ1538E: {0} received a {1} replica with the same container name as the local container. The replica will not be placed. Container: {2}.

Explanation

The primary shard received a new replica, but the new replica is located on the same container as the primary which is an illegal placement. The replica will be rejected.

User response

Use XSAdmin to review where the primary and replica should have been placed. Otherwise, contact IBM Software Support.
Related tasks:
Configuring
Administering
Enabling logging
Analyzing log and trace data

CWOBJ1539W

CLIENT_REPLICA_SERIALIZATION_ERROR_CWOBJ1539=CWOBJ1539W: Unable to serialize client listener data to send to replica, the client listener may not failover: {0}

Explanation

The client replica is serialized and sent over to server replicas to be saved in the event of failover. If there is an error during serialization, the server replicas will not be able to failover the client replica.

User response

Review the error and correct any classpath issues or custom application errors. Otherwise, contact IBM Software Support.
Related concepts:
Class loader and classpath considerations
Related tasks:
Configuring
Administering

CWOBJ1540E

CREATING_CLIENT_REPLICA_TIMED_OUT_CWOBJ1540=CWOBJ1540E: Creating a client replica map times out after {0} ms.

Explanation

The replica did not enter peer mode before the timeout was met. It was not able to copy the primary's existing data and then start to receive data in realtime.

User response

Review the client log for errors copying data.
Related concepts:
Shard life cycles
Related tasks:
Configuring
Administering
Configuring zones for replica placement

CWOBJ1541E

REPLICA_FAIL_ON_PRIOR_MAP_CWOBJ1541=CWOBJ1541E: {0}:{1} Replica map failed to enter peer mode because a previous map failed to enter peer mode. The entire replica cannot continue to enter peer mode. The prior exception was {2}.

Explanation

There was an error on a previous map entering peer mode. The replica cannot finish entering peer mode with some failed maps. Any other maps will fail as well and the replica will not be successfully placed at this time.

User response

Review the previous errors in the log for the failed map. See the Logs and Trace section in the information center for JVM log location. Correct any configuration or network problems. Restart the servers or use the triggerPlacement method on the PlacementServiceMBean Mbean to replace the replica. See the Life cycle, recovery, and failure events section in the information center for more information on the triggerPlacement method.
Related concepts:
Shard life cycles
Related tasks:
Configuring
Administering
Configuring zones for replica placement

CWOBJ1542I

FOREIGN_PRIMARY_REPLICATING_CWOBJ1542=CWOBJ1542I: Primary {0} started or continued replicating from {3} primary ({1}). Replicating for maps: {2}

Explanation

A primary shard started or restarted replicating to another primary.

User response

No action is required.
Related tasks:
Configuring
Administering
Enabling logging

CWOBJ1543I

REPLICA_REPLICATING_CWOBJ1543=CWOBJ1543I: The {0} {1} started or continued replicating from the primary on {3}. Replicating for maps: {2}

Explanation

A replica shard started or restarted replicating.

User response

No action is required.
Related concepts:
Shard life cycles
Related tasks:
Configuring
Administering
Configuring zones for replica placement

CWOBJ1544I

FOREIGN_PRIMARY_REPLICATING_CWOBJ1544=CWOBJ1544I: Primary {0} stopped replicating from {2} primary ({1}).

Explanation

A primary shard stopped replicating from another primary.

User response

No action is required.
Related tasks:
Configuring
Administering
Enabling logging
Analyzing log and trace data

CWOBJ1545W

REPLICA_NOT_PLACED_CWOBJ1545=CWOBJ1545W: Primary {0} could not place a ({1}) on {2}. The remote container did not respond or returned an error.

Explanation

The primary shard attempted to add a replica on a remote container. Either the remote container was not running or the remote container returned an error creating a replica shard.

User response

Verify that the remote container listed in the message is running. If the container stopped or failed while the primary attempted to place a replica on it, another placement event should occur. If the replica shard is not automatically replaced, placement of the shard can be intiated by using the triggerPlacement method on the PlacementServiceMBean Mbean. If there was an error on the remote container, the JVM logs should container the message key CWOBJ1209E.
Related tasks:
Configuring
Administering
Enabling logging
Analyzing log and trace data

CWOBJ1546W

FOREIGN_PRIMARY_NOT_ADDED_CWOBJ1546=CWOBJ1546W: Primary {0} could not initiate replication to a {2} primary on {1}. The remote container did not respond or returned an error.

Explanation

The primary shard attempted to start replication with a another primary. Either the remote container was not running or the remote container returned an error instead of a reference to the other primary shard.

User response

Verify that the remote container listed in the message is running. If the container stopped or failed while the primary attempted to contact it, another placement event should occur. If there was an error on the remote container, the JVM logs should container the message key CWOBJ1209E.
Related tasks:
Configuring
Administering
Enabling logging
Analyzing log and trace data

CWOBJ1547I

SHARD_DEMOTION_CWOBJ1547=CWOBJ1547I: {0} (demoting {1} to {2}) in transition.

Explanation

Specified shard is in a transitional state. The message indicates the state of the shard and the state the shard will become after transition completes. There will likely be an additional message printed later in the log indicating the new state of the shard.

User response

No action is required.
Related tasks:
Configuring
Administering
Enabling logging
Analyzing log and trace data

CWOBJ1548W

ENTERING_PEER_MODE_CWOBJ1548=CWOBJ1548W: Replica shard {0} did not enter peer mode, and the replication from the primary shard in the {1} container failed.

Explanation

This is an informational message on how long it took for a replica to enter peer mode where both primary and replica have the same data.

User response

No action is required.
Related tasks:
Configuring
Administering
Enabling logging
Analyzing log and trace data

CWOBJ1549I

COPY_FROM_REPLICA_CWOBJ1549=CWOBJ1549I: The transitioning primary shard ({0}) did not finish copying data from the previous primary shard on the {1} primary container. The transitioning primary shard will replicate from the existing {2} shard on the {3} replica container.

Explanation

A new primary shard attempted to replicate data from the previous primary shard, but did not finish. To recover, the new primary shard will copy data from an existing replica shard.

User response

No action is required.
Related tasks:
Configuring
Administering
Enabling logging
Analyzing log and trace data

CWOBJ1550W

ERROR_REPLICATING_FROM_PRIMARY_CWOBJ1550=CWOBJ1550W: The {1} ({0}) shard received exceptions while replicating from the primary shard on the {2} primary container. The {1} shard will continue to poll the primary shard. Exception received: {3}

Explanation

The replica or foreign primary shard received exceptions from the primary shard. Replication will continue until the shard is successful or stopped.

User response

Check the JVM logs of the shard for a later message, which indicates that the replica recovered. Alternatively, use xsadmin to review the map sizes or revisions to verify that the shard recovered and matches the primary shard. If the errors continue, review the exception printed to the JVM log or in the FFDC logs.
Related tasks:
Configuring
Administering
Enabling logging
Analyzing log and trace data

CWOBJ1551I

RECOVERED_REPLICATING_FROM_PRIMARY_CWOBJ1551=CWOBJ1551I: The {1} ({0}) shard successfully recovered and replicated after several exceptions from the primary shard on the {2} primary container.

Explanation

The replica or foreign primary shard was able to start replicating successfully from the primary shard again after a period of several exceptions from the primary shard.

User response

Review the exception previously printed in the JVM or FFDC log. Otherwise, no action is required.
Related tasks:
Configuring
Administering
Enabling logging
Analyzing log and trace data

CWOBJ1552W

REPLICA_NOT_REPLICATING_CWOBJ1552=CWOBJ1552W: The {0} {1} could not start replicating from the primary on {3}. Could not replicate for maps, {2}, because {4}.

Explanation

A replica shard was not able to start replication for the listed maps.

User response

Review the JVM logs for additional shard replication or shard stopping messages. If the shard is not stopping, also review the FFDC logs for additional exceptions.
Related tasks:
Configuring
Administering
Enabling logging
Analyzing log and trace data

CWOBJ1553I

REPLICA_IDLE_TIME_CWOBJ1553=CWOBJ1553I: The maximum replication idle level is set to {0} ({1}).

Explanation

The setting for the maximum replication idle time.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ1554E

REPLICATION_IDLE_INCORRECT_CWOBJ1554=CWOBJ1554E: The replication idle level was set to an invalid value of {0}. Valid levels are {1}.

Explanation

The replication idle level was set to an invalid value. A default value will be used instead.

User response

Correct the replication idle level in the server properties file and restart the server.
Related tasks:
Configuring
Administering
Related reference:
Server properties file

CWOBJ1555E

REPLICA_INITIALIZATION_FAIL_CWOBJ1555=CWOBJ1555E: The {0} ({1}) shard failed to initialize. The shard was added by a primary shard on the {2} primary container. The initialization exception is {3}

Explanation

A replica shard was placed on the container server, but there was an error initializing the shard and the shard cannot be started succesfully.

User response

The failed replica will be stopped and primary shard for the partition will notify the catalog server that the replica could not be placed. The replica will be placed again if shards are rebalanced, such as a container stopping or starting, or when the triggerPlacement command is used on the administrative tools.
Related tasks:
Configuring
Administering
Enabling logging
Analyzing log and trace data

CWOBJ1556I

REPLICA_ORPHANED_CWOBJ1556=CWOBJ1556I: The {0} ({1}) shard is orphaned and does not have a valid primary shard. The last primary shard for this {0} shard is on {2} primary container. This shard will be stopped.

Explanation

A replica shard removal was missed. The removal was likely missed when the primary shard for this shard was terminated before completing the removal work. The replica shard completes its own cleanup when it no longer has a valid primary shard.

User response

No action is required.
Related tasks:
Configuring
Administering
Enabling logging
Analyzing log and trace data

CWOBJ1557W

REPLICA_RECEIVED_OLD_PRIMARY_TRAN_CWOBJ1557=CWOBJ1557W: The {0} ({1}) shard received a transaction from a primary shard on the {2} primary container. The current primary shard is on the {3} primary container. The primary on the {2} primary container is an old primary and should be stopped.

Explanation

The shard received a transaction from a primary shard that is not the current primary shard. A re-register action may take place if an exception occurs on the replica. The old primary shard is delayed stopping or the new primary was unable to contact it to stop it.

User response

Verify the old primary on the listed container is stopped. If the replica shard performs a re-register, verify that the maps for the shard entered peer mode successfully.
Related tasks:
Configuring
Administering
Enabling logging
Analyzing log and trace data

CWOBJ1558E

PRIMARY_FAILED_ACTIVATION_CWOBJ1558=CWOBJ1558E: The {0} ({1}) shard failed to activate. The exception that occurred is {2}.

Explanation

The shard was not able to activate. Until the shard is moved, transactions that run against the shard will fail.

User response

If the shard is not stopped and moved automatically, use triggerPlacement on the administrative tool or restart the container.
Related tasks:
Configuring
Administering
Enabling logging
Analyzing log and trace data

CWOBJ1559I

SCHEDULE_ROUTE_UPDATE_CWOBJ1559=CWOBJ1559I: The routing update for grid:mapSet:partitionId:epoch {0} is scheduled for transfer to the catalog server.

Explanation

As part of processing placement work from the catalog server, routing updates for the partition placed in this container server are being scheduled for transfer to the catalog server.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ1560I

ROUTE_UPDATE_SENT_CWOBJ1560=CWOBJ1560I: The next set of routing updates to transfer has been sent to the catalog server.

Explanation

As part of processing placement work from the catalog server, routing updates for partitions that have been recently placed in this container server have been transferred to the catalog server.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ1561I

ROUTE_TRANSFERS_TO_CLIENT_START_CWOBJ1561=CWOBJ1561I: Routing updates are beginning to directly transfer to eXtreme Scale clients.

Explanation

As client requests come in during this time period, if a particular client has not received the latest routing updates from this process, it retrieves the updates as part of the response to its next request.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ1562I

ROUTE_TRANSFERS_TO_CLIENT_STOP_CWOBJ1562=CWOBJ1562I: Routing updates are no longer being directly transferred to eXtreme Scale clients.

Explanation

Routing updates are no longer included in responses to clients until the next change in routing information.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ1600I

GATEWAY_STARTED_CWOBJ1600=CWOBJ1600I: ManagementGateway service started on port ({0}).

Explanation

ManagementGateway service is ready to process requests.

User response

ManagementGateway service is available.
Related tasks:
Configuring
Administering
Related reference:
Deprecated properties and APIs

CWOBJ1601E

GATEWAY_SERVICE_FAILED_CWOBJ1601=CWOBJ1601E: ManagementGateway service failed to start on port ({0}).

Explanation

ManagementGateway service failed to start.

User response

Ensure specified port is not already in use.
Related tasks:
Configuring
Administering
Related reference:
Deprecated properties and APIs

CWOBJ1602E

GATEWAY_CLIENT_CONNECT_FAILED_CWOBJ1602=CWOBJ1602E: ManagementGateway service failed to connect to server at ({0}):({1}).

Explanation

ManagementGateway service failed to connect to server.

User response

Ensure server is running.
Related tasks:
Configuring
Administering
Related reference:
Deprecated properties and APIs

CWOBJ1603E

MANAGEMENT_SERVICE_RESPONSE_FAILED_CWOBJ1603=CWOBJ1603E: Management service failed to respond to ({0}) remote request: {1}.

Explanation

The request failed due to the exception listed in the error message.

User response

Review the exception returned. If there is a configuration problem, correct it and retry the request. Review the troubleshooting section in the information center.
Related tasks:
Configuring
Administering
Related reference:
Deprecated properties and APIs

CWOBJ1604I

MANAGEMENT_GATEWAY_STOP_FAILED_CWOBJ1604=CWOBJ1604I: ManagementGateway service failed to stop connector due to Throwable {0}. Exiting.

Explanation

ManagementGateway service failed to stop connector.

User response

Verify that the eXtreme Scale configuration is still running. If security is enabled, provide the correct security credentials.
Related tasks:
Configuring
Administering
Related reference:
Deprecated properties and APIs

CWOBJ1605I

MANAGEMENT_GATEWAY_REFRESH_FAILED_CWOBJ1605=CWOBJ1605I: ManagementGateway caught Throwable {0} while refreshing attributes. Exiting.

Explanation

ManagementGateway service failed while refreshing attributes.

User response

Verify that the eXtreme Scale configuration is still running. If security is enabled, provide the correct security credentials.
Related tasks:
Configuring
Administering
Related reference:
Deprecated properties and APIs

CWOBJ1606I

NO_RESPONSE_FROM_SERVER_CWOBJ1606=CWOBJ1606I: {0} - Unable to get response from server {1}. Returning false.

Explanation

Unable to get response from server while attempting to stop.

User response

Verify that the eXtreme Scale configuration is still running. If security is enabled, provide the correct security credentials. Attempt to stop again or manually stop the servers.
Related tasks:
Configuring
Administering
Controlling the lifecycle of an ObjectGrid with the ObjectGridManager interface

CWOBJ1607I

USE_WSADMIN_CWOBJ1607=CWOBJ1607I: {0} - When an ObjectGrid server is colocated with a WebSphere Application Server, use WSADMIN to stop server {1}. Returning false.

Explanation

When in a WebSphere Application Server environment, use WSADMIN to stop server.

User response

When in WebSphere Application Server environment, use WSADMIN to stop server.
Related tasks:
Configuring
Administering
Configuring the catalog service in WebSphere Application Server
Creating and augmenting profiles for WebSphere eXtreme Scale

CWOBJ1608I

SERVER_NOT_RESPONDING_NULL_CWOBJ1608=CWOBJ1608I: {0} - Unable to get response from server {1}. Ensure server is running. Returning null.

Explanation

Unable to get response from server.

User response

Wait several seconds and retry operation.
Related tasks:
Configuring
Administering
Controlling the lifecycle of an ObjectGrid with the ObjectGridManager interface

CWOBJ1609I

NO_ROUTING_TABLE_CWOBJ1609=CWOBJ1609I: {0} - Unable to get routing table. Wait several seconds and retry operation. Returning null.

Explanation

Unable to get routing table.

User response

Wait several seconds and retry operation.
Related tasks:
Configuring
Administering
Controlling the lifecycle of an ObjectGrid with the ObjectGridManager interface

CWOBJ1610W

RESET_NULL_CLUSTER_CWOBJ1610=CWOBJ1610W: Try to reset a null cluster for {0}.

Explanation

Replication group cluster data are not available.

User response

No action is required.
Related tasks:
Configuring
Administering
Enabling logging
Analyzing log and trace data

CWOBJ1616I

JMX_SECURITY_NOT_FOUND_CWOBJ1616=CWOBJ1616I: JMX Security implementation not found.

Explanation

MX4J or Java version 5.0 or above is not available.

User response

If JMX Security is required, add MX4J to the classpath or use a level of Java that supports JMX.
Related concepts:
Class loader and classpath considerations
Related tasks:
Configuring
Administering

CWOBJ1617E

NO_JNDI_NAME_SUPPLIED_BIND_CWOBJ1617=CWOBJ1617E: No JNDI name was supplied while doing a bind. The bind will not complete.

Explanation

A JNDI name is need for binding objects. The eXtreme Scale name service uses the following default naming convention: "objectgrid/second name"

User response

Supply a JNDI name for the bind and try the operation again.
Related tasks:
Configuring
Administering

CWOBJ1619E

NO_JNDI_NAME_SUPPLIED_RESOLVE_CWOBJ1619=CWOBJ1619E: No name was supplied while doing a JNDI resolve.

Explanation

A name is need for resolving objects. The eXtreme Scale name service uses the following default naming convention: "objectgrid/second name"

User response

Supply a name for the resolve and try the operation again.
Related tasks:
Configuring
Administering

CWOBJ1620I

REPLACE_SERVER_CWOBJ1620=CWOBJ1620I: Replacing target for wrongly routed request due to changes in the server. The new target is {0}.

Explanation

Old routing target replaced with new target.

User response

If the intended replication group is out of service, restart the servers in the replication group.
Related tasks:
Configuring
Administering

CWOBJ1621E

UNABLE_TO_RESOVLE_JNDI_CWOBJ1621=CWOBJ1621E: Unable to resolve JNDI name {0}.

Explanation

The eXtreme Scale name service uses the following default naming convention: "objectgrid/second name". The supplied name was not found in the name space.

User response

Correct any spelling errors and try the operation again.
Related tasks:
Configuring
Administering

CWOBJ1630I

DOMINO_MODE_CWOBJ1630=CWOBJ1630I: Replication group cannot serve this request {0}.

Explanation

Routing is refused due to the unavailable service such as the Domino effect

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ1632E

NULL_ID_CWOBJ1632=CWOBJ1632E: Original request does not have a valid ID; no way to forward this request.

Explanation

No way to forward this request because the original request does not have a valid ID.

User response

Contact IBM Software Support.
Related tasks:
Configuring
Administering
Contacting IBM support

CWOBJ1634I

BLIND_FORWARD_CWOBJ1634=CWOBJ1634I: Router cannot find the forwarding target; using blind forwarding.

Explanation

Router cannot find the forwarding target.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ1660I

SERVER_NOT_RIGHT_CWOBJ1660=CWOBJ1660I: Replication group member has changed. This server does not host what is requested anymore. The original request is {0}.

Explanation

Replication group member has changed and does not host the received request.

User response

If the intended replication group is out of service, restart the servers in the replication group.
Related tasks:
Configuring
Administering
Enabling logging
Analyzing log and trace data

CWOBJ1663E

VERIFY_NULL_CLUSTER_CWOBJ1663=CWOBJ1663E: Server router cannot verify server routing for {0}, because cluster data for this replication group are null in the server.

Explanation

No replication group cluster data are available to verify.

User response

Contact IBM Software Support.
Related tasks:
Configuring
Administering

CWOBJ1668W

NOT_STARTED_CWOBJ1668=CWOBJ1668W: A client is making a request to a server that has not completed starting.

Explanation

Server startup takes 60-120 seconds. The request will automatically be retried unless you have configured otherwise.

User response

Adjust your configuration or start your clients 60-120 seconds after you start your servers.
Related tasks:
Configuring
Administering

CWOBJ1700I

STANDLAONE_HAMANAGER_INITIALIZED_CWOBJ1700=CWOBJ1700I: Standalone HAManager initialized with coregroup {0}.

Explanation

Standalone HAManager initialized successfully.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ1701I

STANDLAONE_HAMANAGER_ALREADY_INITIALIZED_CWOBJ1701=CWOBJ1701I: Standalone HAManager is already initialized.

Explanation

Standalone HAManager is already initialized successfully.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ1702E

STANDLAONE_HAMANAGER_NOT_INITIALIZED_CWOBJ1702=CWOBJ1702E: Standalone HAManager is not initialized, so it cannot be started.

Explanation

Standalone HAManager is not initialized.

User response

Review the log for any previous configuration errors. Restart the server. If restarting the server does not resolve the HAManager initialization problem, contact IBM Software Support.
Related tasks:
Configuring
Administering

CWOBJ1710I

STANDLAONE_HAMANAGER_STARTED_CWOBJ1710=CWOBJ1710I: Standalone HAManager started successfully.

Explanation

Standalone HAManager started successfully.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ1711I

STANDLAONE_HAMANAGER_ALREADY_STARTED_CWOBJ1711=CWOBJ1711I: Standalone HAManager already started successfully.

Explanation

Standalone HAManager already started successfully.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ1712E

STANDLAONE_HAMANAGER_NOT_STARTED_CWOBJ1712=CWOBJ1712E: Standalone HAManager is not started.

Explanation

Standalone HAManager is not started.

User response

Review the log for any previous configuration errors. Restart the server. If restarting the server does not successfully start HAManager, contact IBM Software Support.
Related tasks:
Configuring
Administering

CWOBJ1713E

STANDLAONE_HAMANAGER_START_FAIL_CWOBJ1713=CWOBJ1713E: Standalone HAManager failed to start.

Explanation

Standalone HAManager failed to start.

User response

Review the log for port conflict errors. Restart the server and optionally set the -HaManagerPort. See the Planning for network ports section in the information center for the HAManager port.
Related tasks:
Configuring
Administering

CWOBJ1767I

DCS_SLIDEBAR_SET_CWOBJ1767=CWOBJ1767I: The DCS heartbeating interval is {0}.

Explanation

The DCS heartbeating interval is used

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ1768I

DCS_SLIDEBAR_SET_CWOBJ1768=CWOBJ1768I: The DCS heartbeating timeout is {0}.

Explanation

The DCS heartbeating timeout is being used and set to the timeout provided.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ1769I

DCS_SLIDEBAR_SET_CWOBJ1769=CWOBJ1769I: The number of DCS heartbeating threads is {0}.

Explanation

The number of DCS heartbeating threads is set to number provided.

User response

No action is required.
Related tasks:

Configuring
Administering

CWOBJ1770I

HA_NOW_COREGROUP_LEADER_CWOBJ1770=CWOBJ1770I: This process is now the core group leader for the {0} core group.

Explanation

One server in every core group serves as the leader for that core group and periodically notifies the catalog server with the list of servers that are running.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ1771I

HA_NOT_COREGROUP_LEADER_CWOBJ1771=CWOBJ1771I: This process is no longer the core group leader for the {0} core group.

Explanation

One server in every core group serves as the leader for that core group and periodically notifies the catalog server with the list of servers that are running. It is possible that the leader might change when a server is going down.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ1772I

HA_MEMBERS_CHANGED_CWOBJ1772I=CWOBJ1772I: The high availability (HA) manager and Distribution and Consistency Services (DCS) have notified eXtreme Scale that the list of servers that are running in this core group has changed to {0}.

Explanation

eXtreme Scale servers learn when other servers in the catalog service domain start or stop, expectedly or unexpectedly.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ1773I

HA_STANDALONE_DEFINED_SET_IMPORTED_CWOBJ1773I=CWOBJ1773I: This server has received an updated high availabliity (HA) defined set from the catalog server, version {0} with the set now containing the servers: {1}

Explanation

The defined set indicates the list of servers that the catalog server has currently assigned to the core group of which this server is a member.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ1790I

HAMANAGER_CONTROLLER_NEED_STANDALONE_HAM_CWOBJ1790=CWOBJ1790I: Need to initialize and start the standalone high availability (HA) manager.

Explanation

There is no external HAManager manager running from a WebSphere Application Server installation. A stand-alone HAManager is initializing and starting.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ1810I

CLIENT_FORWARDING_CWOBJ1810=CWOBJ1810I: Forwarding is required for response {0}.

Explanation

Forwarding is required for response.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ1811E

FORWARDING_NOT_FOUND_REQUEST_CWOBJ1811=CWOBJ1811E: Forwarding is required, but the original request cannot be found.

Explanation

Forwarding is required, but the original request cannot be found.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ1870I

CLIENT_DOMINO_CWOBJ1870=CWOBJ1870I: Server service is not available for response {0}.

Explanation

Server service is not available due to the number of available replicas or other events.

User response

Bring at least the minimum number of servers up.
Related tasks:
Configuring
Administering

CWOBJ1871E

NULL_DOMINO_CWOBJ1871=CWOBJ1871E: Detected unavailable service, received null response, no way to retry.

Explanation

Null response from the unavailable service.

User response

Contact IBM Software Support.
Related tasks:
Configuring
Administering

CWOBJ1872I

CLIENT_DOMINO_TIMEOUT_CWOBJ1872=CWOBJ1872I: Service is unavailable with response of {0}.

Explanation

Service is not available

User response

Bring at least the minimum number of servers up or check if server startup is successful.
Related tasks:
Configuring
Administering

CWOBJ1890I

DEAD_SERVER_REROUTING_CWOBJ1890=CWOBJ1890I: Re-routing request {0} due to an un-responsive server.

Explanation

The request for intended server failed to complete. Request was re-routed to another server.

User response

No action is required. Handled automatically. If the intended replication group is out of service, restart the servers that host the replication group.
Related tasks:
Configuring
Administering

CWOBJ1891E

NO_SERVER_REROUTING_CWOBJ1891=CWOBJ1891E: All servers are not available in replication group {0}.

Explanation

All servers were either not started or have failed. They are not available

User response

If the intended replication group is out of service, restart the servers that host the replication group.
Related tasks:
Configuring
Administering

CWOBJ1899W

FORWARD_NULL_RGID_CWOBJ1899=CWOBJ1899W: Forwarding is required, but router cannot find right replication group for response {0}

Explanation

Replication group ID is lost.

User response

Restart the servers that host the replication group. If the problem continues, contact IBM Software Support.

Related tasks:
Configuring
Administering

CWOBJ1900I

RPC_SERVICE_INIT_CWOBJ1900=CWOBJ1900I: Client server remote procedure call service is initialized.

Explanation

Client server remote procedure call service is initialized.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ1901I

RPC_SERVICE_START_CWOBJ1901=CWOBJ1901I: Client server remote procedure call service is started.

Explanation

Client server remote procedure call service is started.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ1902I

RPC_HANDLER_THREADS_START_CWOBJ1902=CWOBJ1902I: Client server remote procedure call handler threads are started.

Explanation

Client server remote procedure call handler threads are started.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ1903I

CONFIG_NETWORK_SERVICE_INIT_CWOBJ1903=CWOBJ1903I: Configuration network service is initialized.

Explanation

Configuration network service is initialized.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ1904I

CONFIG_NETWORK_SERVICE_START_CWOBJ1904=CWOBJ1904I: Configuration network service is started.

Explanation

Configuration network service is started.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ1905I

CONFIG_NETWORK_HANDLER_START_CWOBJ1905=CWOBJ1905I: Configuration handler is started.

Explanation

Configuration handler is started.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ1921W

Cannot_Find_host_name=CWOBJ1921W: Cannot find host name {0}.

Explanation

You must provide a valid hostname or the local host will be used.

User response

Verify the host name provided is the correct host name for the server and is set correctly.
Related tasks:
Configuring
Administering

CWOBJ1922E

Cannot_Lookup_IP=CWOBJ1922E: Cannot lookup the IP address for this host {0}.

Explanation

The host name that you configured is not correct.

User response

Check the host name and IP address configured for the server.
Related tasks:
Configuring
Administering

CWOBJ1929W

LocalHostUsed=CWOBJ1929W: LOCALHOST is used in the configuration which may lose server identity in multiple machine environment.

Explanation

In multiple computer systems with remote actions, localhost cannot be used.

User response

Change localhost to an actual host name or IP address.
Related tasks:
Configuring
Administering

CWOBJ1931I

ServerSupport=CWOBJ1931I: The configuration for {0} does not support either an ObjectGrid replication group member or a client-server transaction processor. This server will provide bootstrap support to peer ObjectGrid servers and clients only.

Explanation

The configuration for this server does not support either an ObjectGrid replication group member or a client-server transaction processor.

User response

Check your cluster xml configuration and add replication or ObjectGrid bindings if required.
Related tasks:
Configuring
Administering

CWOBJ1932I

ThreadPoolMinMax=CWOBJ1932I: Client thread pool minimum size is {0} maximum size {1}.

Explanation

The minimum and maxiumum client thread pool size.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ2000E

NO_RGM_CWOBJ2000=CWOBJ2000E: No member in this replication group {0}.

Explanation

No member can be found in this replication group.

User response

Check if servers are started or data are available.
Related tasks:
Configuring
Administering

CWOBJ2002W

NO_AVAILABLE_RT_CWOBJ2002=CWOBJ2002W: No available routing table for this replication group {0}.

Explanation

Clients are not able to route to the provided replication group.

User response

Check if clients have brought in routing table and servers that host the replication group are available.
Related tasks:
Configuring
Administering

CWOBJ2010E

NULL_TARGET_CWOBJ2010=CWOBJ2010E: Target for this request is null.

Explanation

Request did not come with target information.

User response

Contact IBM Software Support.
Related tasks:
Configuring
Administering

CWOBJ2020I

ClientProperty_CWOBJ2020=CWOBJ2020I: Client properties are: {0}.

Explanation

Client properties have been loaded or set and include the specified values.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ2024E

FAILED_TO_LOAD_CLIENT_RETRY_CWOBJ2024E=CWOBJ2024E: Failed to process the value for the client retry property {0}. Value supplied: {1}. The correct value type is an integer up to a long indicating the timeout.

Explanation

While processing the client properties file, the client retry value was incorrect. The client retry will not be set.

User response

Change the value in the client properties file (sometimes named client.propertes) to an integer or a long. Restart the client for the correct value to be used.
Related tasks:
Configuring
Administering

CWOBJ2060I

NEW_RT_CHANGE_CWOBJ2060=CWOBJ2060I: Client received new version of replication group cluster {0}.

Explanation

Client received new version of replication group cluster.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ2100I

STALECONN_CWOBJ2100=CWOBJ2100I: Connection ({0}) is stale, it cannot be reused.

Explanation

Connection is stale.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ2400E

INVALID_MAP_SET_CONFIGURATION_CWOBJ2400=CWOBJ2400E: Invalid Configuration: backingMap {0} is a member of more than one mapSet.

Explanation

A backingMap can belong to only one mapSet.

User response

Edit the cluster XML file so that each backingMap belongs to only one mapSet.
Related tasks:
Configuring
Administering

CWOBJ2401E

BACKING_MAP_WO_MAPSET_CWOBJ2401=CWOBJ2401E: Invalid Configuration: backingMap {0} in distributed ObjectGrid {1} is not in a mapSet.

Explanation

Each backingMap of a distributed ObjectGrid must be placed in a mapSet.

User response

Edit the cluster XML file so that each backingMap in a distributed ObjectGrid belongs to a mapSet.
Related tasks:
Configuring
Administering

CWOBJ2402E

MAPSET_REF_NONEXISTENT_BMAP_CWOBJ2402=CWOBJ2402E: Invalid Configuration: mapSet has a reference to a {0} map. This backingMap does not exist in the ObjectGrid XML file.

Explanation

Each map within a mapSet must reference a backingMap from the ObjectGrid XML file.

User response

Edit the XML file(s) so that each map within the mapSet references a backingMap from the ObjectGrid XML file.
Related tasks:
Configuring
Administering

CWOBJ2403E

INVALID_XML_FILE_CWOBJ2403=CWOBJ2403E: The XML file is invalid. A problem has been detected with {0} at line {1}. The error message is {2}.

Explanation

The XML file does not conform to the schema.

User response

Edit the XML file so that it is conforms to the schema. See the Configuring with XML files section in the information center.
Related tasks:
Configuring
Administering

CWOBJ2404W

INVALID_CONFIG_VALUE_CWOBJ2404=CWOBJ2404W: The value specified for {0} is {1}. This is an invalid value. {0} will not be set.

Explanation

The value for this configuration attribute is not valid.

User response

Set the configuration attribute to a proper value in the XML file. See the Configuring with XML files section in the information center.
Related tasks:
Configuring
Administering

CWOBJ2405E

OG_BINDING_REF_NONEXISTENT_OG_CWOBJ2405=CWOBJ2405E: The objectgridBinding ref {0} in the cluster XML file does not reference a valid ObjectGrid from the ObjectGrid XML file.

Explanation

Each of the objectgridBindings must reference an ObjectGrid from the ObjectGrid XML file.

User response

Edit the XML files so that the objectgridBinding in the cluster XML references a valid ObjectGrid in the ObjectGrid XML.
Related tasks:
Configuring
Administering

CWOBJ2406W

INVALID_XML_FILE_CWOBJ2406=CWOBJ2406W: The XML file is invalid. A problem has been detected with {0} at line {1}. The error message is {2}.

Explanation

The XML file does not conform to the schema.

User response

Edit the XML file so that it is conforms to the schema. See the Configuring with XML files section in the information center.
Related tasks:
Configuring
Administering

CWOBJ2407W

PLUGIN_PROPERTY_INVALID_CWOBJ2407=CWOBJ2407W: The {0} property on the {1} plug-in class could not be set. The exception is {2}.

Explanation

The property for this plug-in could not be set.

User response

See the exception and first failure data capture (FFDC) logs for more information. Correct any configuration problems. See the Logs and Trace section in the information center for JVM log location.
Related tasks:
Configuring
Administering

CWOBJ2408E

INVALID_ARGUMENT_CWOBJ2408=CWOBJ2408E: The argument {0} is invalid.

Explanation

This argument is not a valid command line argument.

User response

The argument is not valid for this command. Review the help text with the command for the valid list of arguments and the correct spelling and syntax of the command line arguments.
Related tasks:
Configuring
Administering

CWOBJ2409E

SERVER_STARTUP_EXCEPTION_CWOBJ2409=CWOBJ2409E: The exception {0} occurred during server startup.

Explanation

An exception prevented the server from starting normally.

User response

Review the exception message. Correct any configuration problems and restart the server.
Related tasks:
Configuring
Administering

CWOBJ2410E

ACTIVATION_FAILURE_CWOBJ2410=CWOBJ2410E: The server failed to activate.

Explanation

A problem occurred which caused server activation to fail.

User response

Look for previous exceptions in the log such as networking, configuration or security exceptions. Correct any problems and restart. See the Logs and Trace section in the information center for JVM log location.
Related tasks:
Configuring
Administering

CWOBJ2411E

INITIALIZATION_FAILURE_CWOBJ2411=CWOBJ2411E: The server failed to initialize.

Explanation

A problem occurred which caused server initialization to fail.

User response

Look for previous exceptions in the log such as networking, configuration or security exceptions. Correct any problems and restart. See the Logs and Trace section in the information center for JVM log location.
Related tasks:

Configuring
Administering

CWOBJ2412E

BOOTSTRAP_FAILURE_CWOBJ2412=CWOBJ2412E: The server failed to bootstrap.

Explanation

A problem occurred which caused server bootstrap to fail.

User response

Look for previous exceptions in the log such as networking, configuration or security exceptions. Correct any problems and restart. See the Logs and Trace section in the information center for JVM log location.
Related tasks:
Configuring
Administering

CWOBJ2413E

SERVER_STOP_UNSUCCESSFUL_CWOBJ2413=CWOBJ2413E: The attempt to stop the server was unsuccessful.

Explanation

A problem occurred during normal server shutdown.

User response

Look for previous exceptions in the log such as networking or time out error messages. See the Logs and Trace section in the information center for JVM log location.
Related tasks:
Configuring
Administering

CWOBJ2414E

FORCEFUL_TERMINATION_CWOBJ2414=CWOBJ2414E: The server will be terminated.

Explanation

The server did not stop normally. The server process will be terminated and treated as a failed server. Any primary and replica shards will be failed over to other servers.

User response

Look for previous exceptions in the log such as networking or time out error messages. See the Logs and Trace section in the information center for JVM log location.
Related tasks:
Configuring
Administering

CWOBJ2415I

SCRIPT_CREATION_CWOBJ2415=CWOBJ2415I: Creating script file {0}.

Explanation

A script file will be created in the OBJECTGRID_HOME directory.

User response

See the OBJECTGRID_HOME directory for the script file.
Related tasks:
Configuring
Administering

CWOBJ2416E

PLUGIN_INSTANTIATION_ERROR_CWOBJ2416=CWOBJ2416E: Plug-in {0} could not be instantiated and is not configured. The exception is {1}.

Explanation

Plug-in instantiation was not completed successfully.

User response

See the accompanying exception. Correct any configuration errors and restart the plug-in.
Related tasks:
Configuring
Administering

CWOBJ2417W

DEPRECATED_CLUSTER_XML_ATTRIBUTE_CWOBJ2417=CWOBJ2417W: The {0} attribute on the objectgridBinding element is deprecated in the cluster XML. Use the {0} attribute on the serverDefinition element.

Explanation

This is no longer a valid attribute

User response

Do not use the deprecated attribute. Use the attribute on the serverDefinition element.
Related tasks:
Configuring
Administering

CWOBJ2418E

SERVER_LAUNCH_FAILED_CWOBJ2418=CWOBJ2418E: The server failed to launch.

Explanation

A problem occurred during server startup.

User response

Look for previous exceptions in the log such as networking, configuration or security exceptions. Correct any problems and restart. See the Logs and Trace section in the information center for JVM log location.
Related tasks:
Configuring
Administering

CWOBJ2419W

MIN_THREADPOOL_SIZE_WARNING_CWOBJ2419=CWOBJ2419W: The minThreadPoolSize cannot be less than 1. The default value of {0} will be used.

Explanation

The minThreadPoolSize was set to a value less than 1.

User response

Set the minThreadPoolSize to a value equal to or greater than 1.
Related tasks:
Configuring
Administering

CWOBJ2420W

MAX_THREADPOOL_SIZE_WARNING_CWOBJ2420=CWOBJ2420W: The minThreadPoolSize is set to {0} and maxThreadPoolSize is set to {1}. The maxThreadPoolSize must be greater than minThreadPoolSize. The default values will be used: minThreadPoolSize = {2}, maxThreadPoolSize = {3}.

Explanation

maxThreadPoolSize must be greater than minThreadPoolSize.

User response

Set maxThreadPoolSize to a value greater than the minThreadPoolSize value.
Related tasks:
Configuring
Administering

CWOBJ2421W

OVERRIDE_WARNING_CWOBJ2421=CWOBJ2421W: The java.util.List supplied to override client-side ObjectGrid settings for cluster {0} contains an element that is not an ObjectGridConfiguration object. This element will be removed from the java.util.List: {1}.

Explanation

Client-side overriding will take place using only the objects in the java.util.List that are of type com.ibm.websphere.objectgrid.config.ObjectGridConfiguration.

User response

Remove objects from the client-side override java.util.List that are not of type com.ibm.websphere.objectgrid.config.ObjectGridConfiguration.
Related tasks:
Configuring
Administering

CWOBJ2422I

CHECKSUM_DIFFERENCE_CWOBJ2422=CWOBJ2422I: Configuration version on client might not be the same as configuration version used by this server. Client side: host = {0}, , port = {1}, , Server side: host = {2}, port = {3}.

Explanation

An ObjectGrid client has connected to this server with a configuration version that is different than this server configuration version. This can occur when an ObjectGrid client bootstraps from one ObjectGrid server, and then contacts another server that was started with a different configuration file or the same configuration file with changes.

User response

Users should have administrators compare the configuration files provided by each server to determine if the differences are incompatiable.
Related tasks:
Configuring
Administering

CWOBJ2423I

CLIENT_OVERRIDE_URL_CWOBJ2423=CWOBJ2423I: Client-side ObjectGrid settings will be overridden for cluster {0} using the URL {1}.

Explanation

Overriding ObjectGrids on the client-side using ObjectGrids found in the URL.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ2424I

CLIENT_OVERRIDE_MAP_CWOBJ2424=CWOBJ2424I: Client-side ObjectGrid settings will be overridden for cluster {0} using an entry supplied in the overrideMap.

Explanation

ObjectGridConfigurations will be used to override client-side settings for the cluster specified.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ2425E

CLIENT_OVERRIDE_MAP_ERROR_CWOBJ2425=CWOBJ2425E: The Map provided to override client-side ObjectGrid settings for cluster {0} contains a value that is not of type java.util.List. Client-side ObjectGrid settings will not be overridden for this cluster.

Explanation

Each value in the overrideMap must be of type java.util.List that contains ObjectGridConfiguration objects.

User response

Make this value of type java.util.List
Related tasks:
Configuring
Administering

CWOBJ2426E

CONTAINER_WITHOUT_ZONE_INVALID_CWOBJ2426=CWOBJ2426E: This container has been started without a zone association. This container must be started within a zone since one or more containers in the domain already have been started within one or more zones.

Explanation

If containers have already been started within zones in this domain, then no subsequent container can be started without an association to a zone in the domain.

User response

Start the container within a zone. See the Zone-preferred routing section in the information center for more information on zones.
Related tasks:
Configuring
Administering

CWOBJ2427E

CONTAINER_WITH_ZONE_INVALID_CWOBJ2427=CWOBJ2427E: This container has been started with a zone association. This container must be started without a zone since one or more containers in the domain already have been started without a zone.

Explanation

If containers have already been started without zones in this domain, then no subsequent container can be started with an association to a zone in the domain.

User response

Start the container without a zone. See the Zone-preferred routing section in the information center for more information on zones.
Related tasks:
Configuring
Administering

CWOBJ2428W

ZONE_CONFIG_DEFAULT_INVALID_CWOBJ2428=CWOBJ2428W: The container {0} has started without an association to a zone, but other containers have already started within zones. {0} will be deactivated.

Explanation

If containers have already been started within zones in this domain, then no subsequent container can be started without an association to a zone in the domain.

User response

Start the container within a zone. See the Zone-preferred routing section in the information center for more information on zones.
Related tasks:
Configuring
Administering

CWOBJ2429W

ZONE_CONFIG_CUSTOM_INVALID_CWOBJ2429=CWOBJ2429W: The container {0} started with an association to a zone, but other containers have already started without zone associations. {0} will be deactivated.

Explanation

If containers have already been started without zones in this domain, then no subsequent container can be started within a zone in the domain.

User response

Start the container without a zone. See the Zone-preferred routing section in the information center for more information on zones.
Related tasks:
Configuring
Administering

CWOBJ2430E

ZONE_RULE_TOO_FEW_ZONES_CWOBJ2430=CWOBJ2430E: The zoneRule {0} contains an insufficient number of zones ({1}) for the number of shardMapping entries ({2}) using zoneRule {0}.

Explanation

If a zoneRule contains more than one zone, it must have at least as many zones as shardMappings that make use of the zoneRule.

User response

Add zones to the zoneRule or move shardMappings to different zoneRules. See the Using zones for replica placement section in the information center for more information on zone rules.
Related tasks:
Configuring
Administering

CWOBJ2431E

MAP_SET_NOT_CONFIGURED_FOR_ZONE_CWOBJ2431=CWOBJ2431E: The container was started in zone {0}, but mapSet {1} for ObjectGrid {2} is not configured to run in zone {0}.

Explanation

The zone name that is used to start the container must be within a zoneRule used by one of the shardMappings for the mapSet.

User response

Ensure that the zone used to start the container is also used by each mapSet within the deploymentPolicy for the container.
Related tasks:
Configuring
Administering

CWOBJ2432E

WRONG_NUMBER_SHARD_MAPPINGS_CWOBJ2432=CWOBJ2432E: The wrong number of {0} shardMappings were found for the {1} mapSet in the {2} ObjectGrid. Expected {3} shardMappings, but found {4}.

Explanation

If the shard type is a replica, the number of shardMappings for the type should match the maximum number of replicas specified on the mapSet. There should be only 1 primary shardMapping.

User response

Adjust the number of shardMappings for the shardType in the deployment policy. See the Using zones for replica placement section in the information center for more information on zone rules.
Related tasks:
Configuring
Administering

CWOBJ2433I

CLIENT_OVERRIDE_URL_CWOBJ2433=CWOBJ2433I: Client-side ObjectGrid settings will be overridden for domain {0} using the URL {1}.

Explanation

Overriding ObjectGrids on the client-side using ObjectGrids found in the URL.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ2500E

SERVER_STARTUP_ERROR_CWOBJ2500=CWOBJ2500E: Failed to start ObjectGrid server {0}.

Explanation

The ObjectGrid server failed to start properly.

User response

Check the log for previous exceptions. See the Logs and Trace section in the information center for JVM log location.
Related tasks:
Configuring
Administering

CWOBJ2501I

LAUNCHING_SERVER_CWOBJ2501=CWOBJ2501I: Launching ObjectGrid server {0}.

Explanation

An ObjectGrid server is starting up.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ2502I

LAUNCHING_SERVER_XML_CWOBJ2502=CWOBJ2502I: Starting ObjectGrid server using ObjectGrid XML file URL {0} and Cluster XML file URL {1}.

Explanation

An ObjectGrid server is starting using a cluster XML file and an ObjectGrid xml file.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ2504I

SERVER_BOOTSTRAP_LIST_CWOBJ2504=CWOBJ2504I: Attempting to bootstrap to a peer ObjectGrid server using the following host(s) and port(s) {0}.

Explanation

This ObjectGrid server will use the list of hosts and ports provided in an attempt to connect to a peer ObjectGrid server.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ2506I

COMMAND_LINE_TRACE_FILE_CWOBJ2506=CWOBJ2506I: Trace is being logged to {0}.

Explanation

The trace file has been set on the command line.

User response

See the specified trace file for ObjectGrid server start-up trace.
Related tasks:
Configuring
Administering

CWOBJ2507I

COMMAND_LINE_TRACE_SPEC_CWOBJ2507=CWOBJ2507I: Trace specification is set to {0}.

Explanation

The trace specification has been set on the command line.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ2508I

LAUNCHING_SERVER_SECURITY_CWOBJ2508=CWOBJ2508I: A security properties file {0} has been specified and will be used to start the server.

Explanation

A security properties file has been provided to start a secure server.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ2509E

SERVER_STARTUP_TIMEOUT_CWOBJ2509=CWOBJ2509E: Timed out after waiting {0} seconds for the server to start.

Explanation

The ObjectGrid server did not start within the timeout interval.

User response

Check the log for previous exceptions. See the Logs and Trace section in the information center for JVM log location.
Related tasks:
Configuring
Administering

CWOBJ2510I

SERVER_STOP_CWOBJ2510=CWOBJ2510I: Stopping ObjectGrid server {0}.

Explanation

Stopping ObjectGrid server.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ2512I

SERVER_STOPPED_CWOBJ2512=CWOBJ2512I: ObjectGrid server {0} stopped.

Explanation

ObjectGrid server stopped.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ2514I

SERVER_START_WAITING_CWOBJ2514=CWOBJ2514I: Waiting for ObjectGrid server activation to complete.

Explanation

The ObjectGrid server has launched. Waiting for the server to complete activation.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ2515E

INVALID_ARGS_CWOBJ2515=CWOBJ2515E: The arguments provided are invalid. The following are valid arguments: {0}{1}

Explanation

The arguments provided to this script are not correct and prevent the operation from completing.

User response

Review the valid arguments, check for spelling errors and submit again.
Related tasks:
Configuring
Administering

CWOBJ2518I

LAUNCHING_CATALOG_SERVICE_CWOBJ2518I=CWOBJ2518I: Launching ObjectGrid catalog service: {0}.

Explanation

An ObjectGrid catalog service is starting up.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ2519I

CWOBJ2519=CWOBJ2519I: The client interceptor has not been registered. Security will not be enabled.

Explanation

The client is running without the ObjectGridInitializer specified in the orb.properties file and/or does not have the eXtreme Scale binaries available to the root classloader.

User response

If eXtreme Scale authentication and authorization are required, then complete a full install of WebSphere eXtreme Scale.
Related tasks:
Configuring
Administering

CWOBJ2520E

SERVER_NAME_NOT_FOUND_CWOBJ2520=CWOBJ2520E: Server reference for {0} not found in the supplied configuration, please verify provided server name.

Explanation

The server name provided to start this server could not be found. The server cannot be started.

User response

Correct the server name provided in the configuration or used to start the server.
Related tasks:
Configuring
Administering

CWOBJ2521I

CATALOG_CLUSTER_BOOTSTRAP_CHANGED_CWOBJ2521I=CWOBJ2521I: The catalog server bootstrap addresses changed from {0} to {1}.

Explanation

The catalog server bootstrap addresses changed. This could be because a new catalog server member joins the cluster, a new catalog server member leaves the cluster, or the catalog server cluster restarts with different bootstrap addresses.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ2522I

STATSSPEC_CHANGED_CWOBJ2522I=CWOBJ2522I: The statistics specification has changed to: {0}.

Explanation

The statistics specification can be updated using the StatsAccessor API and has been changed.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ2601I

ADD_SUFFIX_TO_VIEW_NAME=CWOBJ2601I: Add suffix {0} to stream query views deployed in partition {1}.

Explanation

The stream query is executed to a partitioned map set, so we need to add partition name suffix to view names.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ2602W

VIEW_TRANSFORMER_EXISTS=CWOBJ2602W: The view transformer {0} already exists.

Explanation

The view transfomer has already been added.

User response

Contact IBM Software Support.
Related tasks:
Configuring
Administering

CWOBJ2604I

STREAM_QUERY_JAR_NOT_IN_CLASSPATH=CWOBJ2604I: The stream query jar file is not in the class path.

Explanation

The stream query class is shipped in a separate jar file. The jar file is probably not in the classpath.

User response

If you intend to use stream query functions, add the stream query jar file in the classpath.
Related tasks:
Configuring
Administering

CWOBJ2605E

STREAM_QUERY_LOGGER_ERROR=CWOBJ2605E: The stream query logger setting method introspection or invocation error: {0}

Explanation

The stream query logger setting method cannot be introspected or invoked.

User response

Contact IBM Software Support.
Related tasks:
Configuring
Administering

CWOBJ2606W

VIEW_REMOVE_NON_EXISTING_ENTRY=CWOBJ2606W: Try to remove a non-existing entry for key {0}.

Explanation

The entry does not exist in the stream query view map

User response

Contact IBM Software Support.
Related tasks:
Configuring
Administering

CWOBJ2607E

STREAM_QUERY_SET_ACROSS_MAP_SET=CWOBJ2607E: The stream query set with name {0} contains maps from different map sets.

Explanation

A stream query set can only contain maps from one map set.

User response

Make sure a stream query set contain maps from only one map set.
Related tasks:
Configuring
Administering

CWOBJ2608E

EXCEEDED_RETRY_UNPROJECT_CWOBJ2608=CWOBJ2608E: Exceeded retry attempts to publish the message, exception: {0}

Explanation

While attempting to publish a message, it failed several times and exceeded the number of tries allowed.

User response

Review the exception for any network or configuration related errors.
Related tasks:
Configuring
Administering

CWOBJ2609E

CONTAINER_SCOPE_PER_CONTAINER_STRATEGY_ERROR=CWOBJ2609E: The combination of container scope and per container strategy were specified for map set {0}.

Explanation

The combination of container scope and per container strategy is not currently supported.

User response

Change either the placement scope or placement strategy.
Related tasks:
Configuring
Administering

CWOBJ2610W

CONTAINER_SCOPE_REPLICA_WARNING=CWOBJ2610W: A replica setting greater than zero is specified with container scope for map set {0}.

Explanation

With primaries on every container, replicas do not make sense with container scope.

User response

The system will ignore the replica settings, forcing them to zero. Change your config at the next opportunity.
Related tasks:
Configuring
Administering

CWOBJ2611W

CONTAINER_SCOPE_PARTITION_COUNT_WARNING=CWOBJ2611W: A partition count greater than one was specified on the container scope map set {0}.

Explanation

Having more than one primary shard with a container scope map set is not tyipcal.

User response

Confirm whether multiple partitions are required for your container scope map set.
Related tasks:
Configuring
Administering

CWOBJ3001I

EM_SERVICE_STARTED_CWOBJ3001I=CWOBJ3001I: The ObjectGrid EntityManager service is available to process requests for ObjectGrid: {0} and container or server: {1}

Explanation

The EntityManager service can now accept requests.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ3002I

EM_INIT_ENTITIES_CWOBJ3002I=CWOBJ3002I: Initializing entity metadata for ObjectGrid: {0}

Explanation

Entity metadata is being discovered and cached for use.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ3003I

EM_REGISTERED_CWOBJ3003I=CWOBJ3003I: Entity registered: {0}

Explanation

The specified entity metadata has been successfully bound to the ObjectGrid infrastructure.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ3004E

EM_REGISTER_EXCEPTION_CWOBJ3004E=CWOBJ3004E: An exception occurred while registering an entity: {0}

Explanation

An exception was detected when attempting to register an entity with the EntityManager service.

User response

Review the exception, resolve the error and retry the operation.
Related tasks:
Configuring
Administering

CWOBJ3005I

EM_CREATING_INDEX_CWOBJ3005I=CWOBJ3005I: Creating index {0} for entity BackingMap {1}, attribute {2}.

Explanation

An index was not explicitly defined for an entity BackingMap and was automatically created.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ3006E

EM_UNSUPPORTED_INDEX_TYPE_CWOBJ3006E=CWOBJ3006E: The defined MapIndexPlugin type is not supported for index {0} on BackingMap {1} for attribute {2}.

Explanation

An index was created for an entity BackingMap but is not compatible with the EntityManager service

User response

Change the MapIndexPlugin to use a supported index configuration.
Related tasks:
Configuring
Administering

CWOBJ3007E

EM_LATE_REGISTRATION_CWOBJ3007E=CWOBJ3007E: Unable to register new entity {0} after ObjectGrid initialization has completed.

Explanation

New, non-subset entities must be defined prior to ObjectGrid initialization.

User response

Entities can be registered programmatically when using a local, in-memory ObjectGrid. All entity classes must be registered with the ObjectGrid using the registerEntities method prior to calling the initialize or getSession method.
Related tasks:
Configuring
Administering

CWOBJ3008E

EM_BACKINGMAP_REASSOCIATION_CWOBJ3008E=CWOBJ3008E: BackingMap {0} is associated with entity {1} and cannot be reassociated with entity {2}.

Explanation

A BackingMap can only be associated with a single entity type and cannot be reassigned.

User response

Review the entity metadata definitions and choose a name that is not in use.
Related tasks:
Configuring
Administering

CWOBJ3009E

EM_REPOSITORY_EXCEPTION_CWOBJ3009E=CWOBJ3009E: The exception {0} occurred while communicating with the entity metadata repository.

Explanation

An exception occurred while communicating with the entity metadata repository.

User response

Review the exception, resolve the error and retry the operation.
Related tasks:
Configuring
Administering

CWOBJ3010E

EM_INVALID_MAPSET_CWOBJ3010E=CWOBJ3010E: All entity BackingMaps must be members of a MapSet with the name: "ENTITY_MAPSET".

Explanation

All entity BackingMaps must be defined in a single MapSet named ENTITY_MAPSET when using the EntityManager service in a clustered ObjectGrid.

User response

Identify the entity BackingMaps and add them to the ENTITY_MAPSET MapSet.
Related tasks:
Configuring
Administering

CWOBJ3011E

EM_METADATALISTENER_EXCEPTION_CWOBJ3011E=CWOBJ3011E: Error creating entity metadata for entity {0} ({1}): {2}

Explanation

An exception prevented the entity metadata from being created.

User response

Review the exception, resolve the error and retry the operation.
Related tasks:
Configuring
Administering

CWOBJ3013E

EM_MULTIPLE_MAPSETS_CWOBJ3013E=CWOBJ3013E: The EntityMetadata repository is not available. Timeout threshold reached when trying to register the entity: {0}.

Explanation

The runtime could not register the defined entities with the metadata repository.

User response

Verify that there is at least one primary shard activated for the entity manager. If you are only running with one container, the deployment policy parameter minSyncReplicas must be set to zero. Use xsAdmin to list the current placement of shards or review the logs for entity manager shards. See the Logs and Trace section in the information center for JVM log location.
Related tasks:
Configuring
Administering

CWOBJ3014I

AVAILABILITY_STATE_CHANGED_CWOBJ3014=CWOBJ3014I: The availability state has changed for {0}. The state is now {1}. It was previously {2}.

Explanation

The availability state for a shard has changed.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ3015E

EM_MISSING_MAPSET_CWOBJ3015E=CWOBJ3015E: Invalid entity MapSet configuration. Unable to find MapSet that contains a BackingMap for {0}.

Explanation

Each entity needs a BackingMap of the same name.

User response

Verify that a BackingMap named {0} is defined in your configuration. See the Entity manager in a distributed environment in the information center for more information on writing entity and objectGrid configuration files.
Related tasks:
Configuring
Administering

CWOBJ3016E

EM_SCHEMA_MAPSET_CROSSOVER_CWOBJ3016E=CWOBJ3016E: Invalid entity MapSet configuration. Entity {0} should be present in MapSet {1} but is already exists in MapSet {2}.

Explanation

The entities for a logical schema must be contained in a single mapset.

User response

Make sure that entities do not contain references to other entities that exist in another mapset and restart the server.
Related tasks:
Configuring
Administering

CWOBJ3017E

FAILED_TO_VERIFY_ENTITY_CWOBJ3017E=CWOBJ3017E: An entity {0} has been defined in the entity descriptor XML file, but does not have an associated backing map of the same name defined.

Explanation

Each entity is associated with a single backing map. The backing map must be defined in the ObjectGrid configuration with the same name as the entity.

User response

Verify that there is a backing map configured that matches the entity name and resubmit the operation.
Related tasks:
Configuring
Administering

CWOBJ3018E

FAILED_TO_INIT_ENTITIES_CWOBJ3018E=CWOBJ3018E: Failed to initialize the entities in ObjectGrid {0}.

Explanation

There was a problem loading and initializing one or more of the specified the entity classes.

User response

Make sure all your entity classes are on your classpath and check your entity configuration for errors.
Related tasks:
Configuring
Administering

CWOBJ3019E

FAILED_TO_LOAD_OG_CLASSES_CWOBJ3019E=CWOBJ3019E: The class {0} cannot be found for ObjectGrid {1}.

Explanation

The classes specified could not be loaded or were not present on the classpath.

User response

Update your configuration with the right class name or update your classpath or JEE application module to include the necessary classes.
Related tasks:
Configuring
Administering

CWOBJ3101E

WB_LOADER_INITIALIZATION_FAILED_CWOBJ3101E=CWOBJ3101E: The write-behind loader for map {0} on partition {1} failed to initialize with exception {2}.

Explanation

The write-behind loader initialization failed. The session might not be initialized or the write behind queue map does not exist.

User response

Review the exception for configuration errors or the log for prior errors. Correct any configuration errors and restarts the servers. See the Logs and Trace section in the information center for JVM log location. Otherwise, contact IBM Software Support.
Related tasks:
Configuring
Administering

CWOBJ3102E

WB_LOADER_FAILED_CWOBJ3102E=CWOBJ3102E: Loader failed to do a write-behind update to the database for map {0} on partition {1}. A failed update is logged in the failed update map. The failed update index is {2}, and the failed map key is {3}. The exception causing the failed update was {4}.

Explanation

Loader fails to do a write-behind update to the database. It could be that the database is updated by other applications. The write-behind loader logs the failed update in the failed update map.

User response

Remove the failed update data from the failed update map, examine the exception, and compensate the failed update.
Related tasks:
Configuring
Administering

CWOBJ3103E

WB_LOADER_FAILED_CWOBJ3103E=CWOBJ3103E: The write-behind loader for map {0} on partition {1} failed to complete a transaction. The exception is {2}.

Explanation

The write-behind loader failed to complete a transaction. See exception for more details.

User response

Look at the exception and take appropriate actions to compensate this failure. See the Handling failed write-behind updates in the information center for more information on handling failed updates.
Related tasks:
Configuring
Administering

CWOBJ3104W

WB_LOADER_LOCKTIMEOUT_CWOBJ3104W=CWOBJ3104W: The write-behind loader for map {0} on partition {1} gets a lock timeout exception, {2}, when trying to switch the queue maps.

Explanation

The write-behind loader gets a lock timeout exception when trying to flip the queue map states. See exception for more details.

User response

A lock timeout exception may be normal in some cases. The write-behind loader recovers from the exception and will try to switch the queue maps later. If this problem continues happening, try to increase the lock timeout value, or contact IBM Software Support.
Related tasks:
Configuring
Administering

CWOBJ3105E

WB_LOADER_LOADER_NOT_AVAILABLE_CWOBJ3105E=CWOBJ3105E: The write-behind loader for ObjectGrid {0}, map {1} on partition {2} received an {3} error.

Explanation

The write-behind loader catches a LoaderNotAvailableException. The write-behind loader will try to update the loader again at the next interval.

User response

Examine the exception to find the root problem and correct it. Most likely, the network is down, database is down, or the database ran out of resources.
Related tasks:
Configuring
Administering

CWOBJ3106W

WB_LOADER_LONG_TRAN_CWOBJ3106W=CWOBJ3106W: The write-behind loader for ObjectGrid {0}, map {1} on partition {2} {3} a {4} ms long transaction when removing the data from the queue map and batch updating to the loader. Within this eXtreme Scale transaction, the loader batch update takes {5} ms. The possible causes are: 1) The data store backend cannot keep up. Considering tuning database and using connection pool. 2) The write-behind parameter update count is too big or update time is too long. Consider decreasing the write-behind parameter value.

Explanation

The write-behind loader transaction takes a long time to complete. This normally indicates some tuning is needed. The possible causes are: 1) The data store backend cannot keep up. Considering tuning database and using connection pool. 2) The write-behind parameter update count is too big or update time is too long. Consider decreasing the write-behind parameter value.

User response

Tune database and use connection pool. And consider decreasing the write-behind parameter value.
Related tasks:
Configuring
Administering

CWOBJ3107W

WB_LOADER_SMALL_TRAN_TIMEOUT_CWOBJ3107W=CWOBJ3107W: The write-behind loader for ObjectGrid {0}, map {1} on partition {2} {3} a {4} ms long transaction, which is approaching to the transaction timeout value {5} ms. Within this eXtreme Scale transaction, the loader batch update takes {6} ms. The transaction timeout value is probably too small. Consider increasing the transction timeout value.

Explanation

The transaction timeout value is probably too small. Consider increasing the transction timeout value.

User response

Increase the transaction timeout value.
Related tasks:
Configuring
Administering

CWOBJ3108E

WB_LOADER_REPLICA_UNAVAILABLE_CWOBJ3108E=CWOBJ3108E: The write-behind loader of ObjectGrid {0}, map {1} on partition {2} received a ReplicationVotedToRollbackTransactionException: {3}

Explanation

The required minimum number of sync replicas did not successfully commit the write-behind transaction and voted to rollback the transaction.

User response

If a failover recently occured, a sync replica may be in the process of being placed on another container. Review the container placement status and the route table availability with the xsadmin tool or administrative tools.
Related tasks:
Configuring
Administering

CWOBJ3111E

CLIENT_LOADER_AGENT_FAIL_CWOBJ3111E=CWOBJ3111E: The client loader agent {0} execution fails with exception: {1}.

Explanation

The client loader agent execution fails. The agent will be re-executed for a certain number of times.

User response

No action is required unless the agent fails permanently with a later exception.
Related tasks:
Configuring
Administering

CWOBJ3112I

DEFAULT_PERSISTENCE_UNIT_CWOBJ3112I=CWOBJ3112I: A JPA persistence unit name was not specified. The first persistence unit {0} defined in the persistence.xml will used as the default persistence unit.

Explanation

The JPA persistence unit name was not provided to the method using the persistence.xml. By default, eXtreme Scale will use the first persistence unit defined in the persistence.xml.

User response

No action is required if the first persistence unit is acceptable. Otherwise, specify a persistence unit name when calling methods with the persistence unit name parameter (for example, ClienterLoader#Load).
Related tasks:
Configuring
Administering

CWOBJ3113E

AGENT_FAIL_CWOBJ3113E=CWOBJ3113E: The DataGrid agent {0} execution failed with a exception {1}.

Explanation

The DataGrid agent execution failed with a fatal non-retryable exception.

User response

Examine the exception and agent implementation for possible causes.
Related tasks:
Configuring
Administering

CWOBJ3114E

AGENT_FAIL_RETRYABLE_CWOBJ3114E=CWOBJ3114E: The DataGrid agent {0} execution failed with a retryable exception {1}.

Explanation

The DataGrid agent execution failed with a retryable exception. The agent will be tried again automatically.

User response

No action is required unless there is a configuration exception that needs to be corrected.
Related tasks:
Configuring
Administering

CWOBJ3115E

UNEXPECTED_SHARD_STATE_CWOBJ3115E=CWOBJ3115E: The shard is expected to be in the {0} state, but currently it is in the {1} state. If you have already set the shard to the {0} state, it might take a while for it to move to the target state. If you have not set the shard to the {0} state, revise your application to do so.

Explanation

The shard state is not expected. It might take a while for a shard to move to the target state.

User response

Update the application to set the shard state to the expected state if necessary. Otherwise, no action is required. The state will be automatically checked again.
Related tasks:
Configuring
Administering

CWOBJ3116I

PRELOAD_STARTS_CWOBJ3116I=CWOBJ3116I: Preloading ObjectGrid {0} Map {1} at partition {2} started.

Explanation

The preload started.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ3117I

PRELOAD_FINISHES_CWOBJ3117I=CWOBJ3117I: Preloading ObjectGrid {0} Map {1} at partition {2} finished.

Explanation

The preload finished.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ3118E

PRELOAD_FAILS_CWOBJ3118E=CWOBJ3118E: Failed to preload ObjectGrid {0} Map {1} at partition {2} with the exception {3}.

Explanation

The preload failed with an exception.

User response

Review the exception for any database configuration errors, network exceptions or other errors. Correct them and retry the preload.
Related tasks:
Configuring
Administering

CWOBJ3121E

TIME_BASED_DBUPDATE_AGENT_FAIL_CWOBJ3121E=CWOBJ3121E: The time-based database update agent fails with exception {0}.

Explanation

The time-based database update agent execution fails. The agent will be re-executed for a certain number of times.

User response

No action is required unless there is a configuration exception that needs to be corrected.
Related tasks:
Configuring
Administering

CWOBJ3122E

TIME_BASED_DBUPDATE_FAIL_CWOBJ3122E=CWOBJ3122E: The time-based database update fails with exception {0}.

Explanation

The time-based database update fails. The update will be tried again.

User response

No action is required unless there is a configuration exception that needs to be corrected.
Related tasks:
Configuring
Administering

CWOBJ3131E

JPA_TX_CALLBACK_NOT_FOUND_CWOBJ3131E=CWOBJ3131E: The JPATxCallback transaction callback plug-in cannot be found.

Explanation

When using JPALoader or JPAEntityLoader, the JPATxCallback transaction callback plug-in is expected to be configured on the ObjectGrid.

User response

Configure the JPATxCallback transaction callback plug-in on the ObjectGrid. See the Configuring JPA loaders section in the information center for more information on configuring the JPATxCallback transaction callback plug-in.
Related tasks:
Configuring
Administering

CWOBJ3132E

EMF_NOT_FOUND_CWOBJ3132E=CWOBJ3132E: The JPA EntityManagerFactory with persistence unit name {0} and property map {1} could not be found.

Explanation

The JPA EntityManagerFactory cannot be found.

User response

Review the log for prior configuration errors. Correct any errors and restart the server. See the Logs and Trace section in the information center for JVM log location. Otherwise, contact IBM Software Support
Related tasks:
Configuring
Administering

CWOBJ3133E

OBJECTGRID_CACHE_INITIALIZE_OBJECTGRID_FAILED_CWOBJ3133E=CWOBJ3133E: ObjectGrid cache plug-in initialization with ObjectGrid {1} failed with exception {0}.

Explanation

An exception occurred while attempting to initialize ObjectGrid for an ObjectGrid cache plug-in. Possible causes are problems during creating ObjectGrid server or container, connecting to catalog service, or duplicate ObjectGrid name.

User response

Review the provided exception for configuration or networking errors. Correct any problems and restart the server.
Related tasks:
Configuring
Administering

CWOBJ3134I

OBJECTGRID_CACHE_TYPE_EMBEDDED_CWOBJ3134I=CWOBJ3134I: The ObjectGrid type is {0} and the default maximum number of replicas is {1}.

Explanation

To avoid low performance in JVMs, the configured ObjectGrid type requires that the maximum number of replicas is greater than or equal to the number of JVMs in the system.

User response

The maximum number of replicas must be greater than or equal to the number of JVMs in the system. Adjust the number of replicas in the deployment policy. See the information center for details about configuring deployment policies.
Related tasks:
Configuring
Administering

CWOBJ3135I

OBJECTGRID_CACHE_TYPE_EMBEDDED_PARTITION_CWOBJ3135I=CWOBJ3135I: The ObjectGridType is {0} and the default number of partitions is {1}. The number of partitions must be less than or equal to the number of JVMs in the system.

Explanation

To avoid poor performance, the configured ObjectGrid type requires the number of partitions is less than or equal to the number of JVMs in the system.

User response

The number of partitions must be less than or equal to the number of JVMs in the system. Adjust the number of partitions in the deployment policy. See the information center for details about configuring deployment policies.
Related tasks:
Configuring
Administering

CWOBJ3136I

OBJECTGRID_CACHE_TYPE_EMBEDDED_INTERDOMAIN_CWOBJ3136I=CWOBJ3136I: The ObjectGrid type is {0}. The placement scope is {1} and the scope topology is {2}.

Explanation

This is an informational message describing how the JPA grid is configured.

User response

No user action required.
Related tasks:
Configuring
Administering

CWOBJ3141W

NODEGROUP_NOT_SET_FOR_ZONE_SUPPORT_CWOBJ3141W=CWOBJ3141W: This WebSphere Application Server is not associated with a WebSphere eXtreme Scale zone. In order to start the server in a zone, ensure that the server's node is within a node group whose name begins with the string ReplicationZone.

Explanation

A WebSphere Application Server's node must be within a nodegroup whose name begins with ReplicationZone in order for that server to be placed into a WebSphere eXtreme Scale zone.

User response

In order to start the WebSphere Application Server in a WebSphere eXtreme Scale zone, ensure that the server's node is within a node group whose name begins with the string ReplicationZone. See the Zone-preferred routing section in the information center for more information on zones.
Related tasks:
Configuring
Administering

CWOBJ3142I

NODEGROUP_NOT_SET_FOR_ZONE_SUPPORT_CWOBJ3142I=CWOBJ3142I: This WebSphere Application Server is not associated with a WebSphere eXtreme Scale zone. In order to start the server in a zone, ensure that the server's node is within a node group whose name begins with the string ReplicationZone.

Explanation

A WebSphere Application Server's node must be within a nodegroup whose name begins with ReplicationZone in order for that server to be placed into a WebSphere eXtreme Scale zone.

User response

In order to start the WebSphere Application Server in a WebSphere eXtreme Scale zone, ensure that the server's node is within a node group whose name begins with the string ReplicationZone. See the Zone-preferred routing section in the information center for more information on zones.
Related tasks:
Configuring
Administering

CWOBJ3150E

CLEAR_TIMED_OUT_CWOBJ3150=CWOBJ3150E: The clear operation timed-out after {0} ms.

Explanation

The clear command timed out because the server is not responsive or an exception has occurred on the server.

User response

Ensure all servers are online, examine each server's first failure data capture (FFDC) logs, and resubmit the clear command. See the Logs and Trace section in the information center for JVM log location.
Related tasks:
Configuring
Administering

CWOBJ3175E

FAILED_TO_REGISTER_BEAN_FACTORY_CWOBJ3175E=CWOBJ3175E: Exception {2} occurred when registering Spring bean factory {0} with the ObjectGrid {1}.

Explanation

The Spring bean factory specified could not be registered with the ObjectGrid.

User response

Update your configuration with the right factory name or update your classpath or Java EE application module to include the necessary classes.
Related tasks:
Configuring
Administering

CWOBJ3176E

FAILED_TO_GET_BEAN_FACTORY_CWOBJ3176E=CWOBJ3176E: Exception {1} occurred when loading Spring bean factory with the ObjectGrid {0}.

Explanation

The Spring bean factory specified could not be found.

User response

Update your configuration with the right class name or update your classpath or Java EE application module to include the necessary classes.
Related tasks:
Configuring
Administering

CWOBJ3177E

FAILED_TO_LOCATE_OBJECTGRID_XML_FILE_CWOBJ3177E=CWOBJ3177E: Failed to locate the ObjectGrid XML file: {0}.

Explanation

There was a problem locating the specified ObjectGrid XML file.

User response

Update your configuration with the right file name and ensure the file is in the location specified.
Related tasks:
Configuring
Administering

CWOBJ3178E

BACKING_MAP_NOT_FOUND_IN_OBJECTGRID_XML_CWOBJ3178E=CWOBJ3178E: The map {1} in ObjectGrid {0} referenced in the ObjectGrid XML was not found in the deployment descriptor file.

Explanation

An entry was found for a backing map in the ObjectGrid XML file but not the deployment descriptor XML file.

User response

Add the backing map to the proper mapset in the deployment descriptor XML file or remove the entry from the ObjectGrid XML file.
Related tasks:
Configuring
Administering

CWOBJ3179E

INVALID_BACKING_MAP_IN_DEPLOYMENT_XML_CWOBJ3179E=CWOBJ3179E: The map {0} reference in the mapSet {1} of ObjectGrid {2} deployment descriptor file does not reference a valid backing map from the ObjectGrid XML.

Explanation

The specified map is referenced in the specified mapSet of the ObjectGrid deployment descriptor but is not found in the ObjectGrid XML.

User response

Add the missing map to the specified mapSet in the in the ObjectGrid XML or remove the invalid map in the deployment descriptor XML file.
Related tasks:
Configuring
Administering

CWOBJ3180E

INVALID_OBJECTGRID_IN_DEPLOYMENT_XML_CWOBJ3180E=CWOBJ3180E: The ObjectGrid {0} specified in the deployment descriptor file is not defined in the ObjectGrid XML file.

Explanation

The specified ObjectGrid was found in the deployment descriptor XML file but not in the ObjectGrid XML file.

User response

Add the missing ObjectGrid to the ObjectGrid XML file or remove the entry in the deployment descriptor XML file.
Related tasks:
Configuring
Administering

CWOBJ3181E

INVALID_SERVER_SECURITY_FILE_OPTION_CWOBJ3181E=CWOBJ3181E: The command-line option -serverSecurityFile is invalid for ObjectGrid container servers.

Explanation

The command-line option -serverSecurityFile is not valid for ObjectGrid catalog service.

User response

Remove the command-line option -serverSecurityFile and restart the ObjectGrid catalog service.
Related tasks:
Configuring
Administering

CWOBJ3182E

XERCES_IMPLEMENTATION_NOT_IN_CLASSPATH_CWOBJ3182E=CWOBJ3182E: Apache Xerces2 was not found in the classpath.

Explanation

ObjectGrid uses the Apache Xerces2 XML parser to parse XML configuration files. The Apache Xerces2 libraries must be present in the classpath for ObjectGrid to operate correctly.

User response

Download and add Apache Xerces 2.9 or later to the process classpath and restart the ObjectGrid process. Apache Xerces is available at: http://xerces.apache.org/xerces2-j
Related tasks:
Configuring
Administering

CWOBJ3183W

CONTAINER_PLACEMENT_SCOPE_IN_DEPLOYMENT_XML_IGNORE_REPLICA_CWOBJ3183W=CWOBJ3183W: When the container placement scope of CONTAINER_SCOPE setting is specified, any replica setting must be zero.

Explanation

The mapSet with name {0} contains non zero replica setting(s) in the deployment descriptor file.

User response

Set maxAsyncReplicas="0" maxSyncReplicas="0" minSyncReplicas="0" in the deployment descriptor file.
Related tasks:
Configuring
Administering

CWOBJ3184W

CONTAINER_PLACEMENT_SCOPE_IN_DEPLOYMENT_XML_NON_ONE_PARTITION_COUNT_CWOBJ3184W=CWOBJ3184W: When the container placement scope of CONTAINER_SCOPE setting is specified, the number of partitions setting must be one.

Explanation

The mapSet with name {0} contains {1} for the number of partitions setting in the deployment descriptor file.

User response

Confirm whether multiple partitions are required for your container scope map set.
Related tasks:
Configuring
Administering

CWOBJ3185E

INVALID_PLACEMENT_SCOPE_IN_DEPLOYMENT_XML_CWOBJ3185E=CWOBJ3185E: The placement strategy of per container and the container placement scope of CONTAINER_SCOPE can not be used together.

Explanation

The mapSet with name {0} contains mismatched placementStrategy and placementScope settings in the deployment descriptor file.

User response

Either set the placementStrategy to "FIXED_PARTITIONS" or set the placementScope to "DOMAIN_SCOPE" in the deployment descriptor file.
Related tasks:
Configuring
Administering

CWOBJ3186I

CONTAINER_PLACEMENT_SCOPE_IN_DEPLOYMENT_XML_DEFAULT_COLLISION_ARBITER_CWOBJ3186I=CWOBJ3186I: No custom COLLISION_ARBITER is defined. The default arbitration will be use.

Explanation

The mapSet with name {0} is being defined with the default arbitration.

User response

none
Related tasks:
Configuring
Administering

CWOBJ3187E

ERROR_IN_ARBITER_CWOBJ3187E=CWOBJ3187E: The collision arbiter implementation on {1} generate exception, {0}, which will result in a halt to replication.

Explanation

During replication, the same key collided with updates from another primary. The collision arbiter encountered an error resolving the collision.

User response

Review the provided error in the JVM logs and the FFDC logs. Review your custom collision arbiter code.
Related tasks:
Configuring
Administering

CWOBJ3188E

CONTAINER_PLACEMENT_SCOPE_IN_DOWNLEVEL_CONTAINER_CWOBJ3188E=CWOBJ3188E: A map set with a container placement scope of CONTAINER_SCOPE can not be deployed to a pre 7.1.1 WebSphere eXtreme Scal container.

Explanation

The container {0} on server {1} is at internal version {2} has a configured CONTAINER_SCOPE map set with grid name {3} and map set {4} deployed to it. This map set can not be deployed to the downlevel container.

User response

If you want this map set to run on this container, the container must be upgraded to at least version 7.1.1.
Related tasks:
Configuring
Administering

CWOBJ3189I

CONTAINER_PLACEMENT_SCOPE_HUB_CONTAINER_CWOBJ3189I=CWOBJ3189I: The hub container for the container scope placement scope map set {1} is container {0}.

Explanation

See documentation for explanation of hub topology.

User response

none
Related tasks:
Configuring
Administering

CWOBJ3190E

INVALID_CONTAINER_PLACEMENT_SCOPE_IN_DEPLOYMENT_XML_CWOBJ3190E=CWOBJ3190E: When the container placement scope of CONTAINER_SCOPE setting is specified in the deployment descriptor file, the Loader class in the object grid file can not be specified.

Explanation

The mapSet with name {0} contains mismatched placementScope setting in the deployment descriptor file and Loader class setting in the object grid file.

User response

Either remove the Loader class from the object grid file or set the placementScope to "DOMAIN_SCOPE" in the deployment descriptor file.
Related tasks:
Configuring
Administering

CWOBJ4000I

RESTSERVICE_STARTED_CWOBJ4000I=CWOBJ4000I: The WebSphere eXtreme Scale REST data service has been started.

Explanation

The eXtreme Scale REST data service has successfully been started and can accept HTTP requests from clients.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ4001I

RESTSERVICE_VERSION_CWOBJ4001I=CWOBJ4001I: The WebSphere eXtreme Scale REST data service version is {0}.

Explanation

No action is required.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ4002E

RESTSERVICE_STARTUPFAILURE_CWOBJ4002E=CWOBJ4002E: The WebSphere eXtreme Scale REST data service could not be started.

Explanation

The REST data service could not be started due to an error or exception.

User response

Review previous errors in the log, correct the problem and restart the service.
Related tasks:
Configuring
Administering

CWOBJ4003E

RESTSERVICE_MISSINGCATALOGSERVICE_CWOBJ4003E=CWOBJ4003E: Unable to connect to the catalog service. The catalog service endpoints were not specified.

Explanation

The REST data service properties file is missing the catalogServiceEndpoints value and the WebSphere Application Server catalog.service.cluster property is not set.

User response

If running in a WebSphere Application Server process, set the catalog.service.cluster property or provide a wxsRestServices.properties file in the classpath and specify the catalogServicesEndpoints property.
Related tasks:
Configuring
Administering

CWOBJ4004I

RESTSERVICE_PROPSLOADED_CWOBJ4004I=CWOBJ4004I: The eXtreme Scale REST data service properties files were loaded: {0}

Explanation

The REST data service configuration property files were loaded by the REST data service from the specified location.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ4005E

RESTSERVICE_MISSINGCLIENTOGXML_CWOBJ4005E=CWOBJ4005E: The client ObjectGrid descriptor XML file "{0}" could not be found in the classpath.

Explanation

A client ObjectGrid descriptor XML file was specified in the REST data service properties file but could not be found.

User response

Correct the objectGridClientXML property setting in the REST data service properties file and restart the REST data service.
Related tasks:
Configuring
Administering

CWOBJ4006I

RESTSERVICE_CONNECTIONENDPOINTS_CWOBJ4006I=CWOBJ4006I: Connecting to eXtreme Scale catalog service endpoints: {0}

Explanation

The REST data service is connecting to an eXtreme Scale grid catalog service located at the specified hosts and ports.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ4007E

RESTSERVICE_CONNECTFAILURE_CWOBJ4007E=CWOBJ4007E: The WebSphere eXtreme Scale REST data service was unable to connect to the eXtreme Scale grid: {0}

Explanation

The eXtreme Scale REST data service received an exception from the eXtreme Scale runtime when attempting to connect to one or more ObjectGrids.

User response

Examine the exception and any previous messages, correct the problem and restart the REST data service.
Related tasks:
Configuring
Administering

CWOBJ4008W

RESTSERVICE_ATOM_WRONG_ELEMENT_NAMESPACE_WARNING_CWOBJ4008W=CWOBJ4008W: The XML element "{0}" specified in the AtomPub format XML has a wrong namespace prefix "{1}". The valid namespace prefix should be resolved to "{2}".

Explanation

A wrong namespace prefix is specified for an XML element. The property will be ignored.

User response

Fix the namespace prefix to resolve to the correct namespace.
Related tasks:
Configuring
Administering

CWOBJ4010I

RESTSERVICE_GRIDSAVAILABLE_CWOBJ4010I=CWOBJ4010I: The following ObjectGrids can now be accessed from the REST data service: {0}

Explanation

The eXtreme Scale REST data service successfully connected to the specified ObjectGrid instances. Clients can now access data for those grids.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ4011E

RESTSERVICE_METADATA_INVALID_CWOBJ4011E=CWOBJ4011E: The entity metadata for ObjectGrid "{0}" is configured incorrectly.

Explanation

One or more entities defined in the ObjectGrid is not valid or is incompatible with the eXtreme Scale REST service.

User response

Additional messages are logged which include details of the metadata error. Review the log for previous metadata related error messages. Correct the entity metadata definition as described in the related messages and restart the eXtreme Scale grid and REST data service.
Related tasks:
Configuring
Administering

CWOBJ4012E

RESTSERVICE_METADATA_INVALID_UNIASSOC_CWOBJ4012E=CWOBJ4012E: The association "{0}" defined for entity "{1}" is not mapped to a target association on entity "{2}". All associations must be bi-directional and have the mapped-by attribute defined.

Explanation

The specified association is not valid. All associations must be bi-directional and have the mapped-by attribute defined.

User response

Add the mappedBy annotation attribute to the entity association or add the mapped-by attribute in the entity descriptor XML file for the entity in the eXtreme Scale configuration.
Related tasks:
Configuring
Administering

CWOBJ4013E

RESTSERVICE_METADATA_AUTOGEN_KEY_COLLISION_CWOBJ4013E=CWOBJ4013E: An automatically generated key association name resulted in a duplicate attribute "{0}" for entity "{1}".

Explanation

The eXtreme Scale REST data service generates key names for entities with key relationships. The key name is already defined for the entity.

User response

Rename the attribute field or property defined in the entity to a different name and restart the eXtreme Scale grid and the REST data service.
Related tasks:
Configuring
Administering

CWOBJ4014E

RESTSERVICE_METADATA_NOROOTPATH_CWOBJ4014E=CWOBJ4014E: The partitioned entity "{0}" must be defined as a schema root or must have a key relationship to the schema root.

Explanation

All entities that are included in a partitioned map set must have the key for the schema root as part of its identity. The specified entity is neither a schema root nor has a key relationship to a root entity.

User response

Update the entity metadata to either include a one-to-one or many-to-one key relationship to an entity that is defined as a schema root, an entity that has a key relationship to the schema root entity, or mark the entity as a schema root. Each entity schema can only have one root.
Related tasks:
Configuring
Administering

CWOBJ4015E

RESTSERVICE_METADATA_INVALIDNAME_ATTR_CWOBJ4015E=CWOBJ4015E: The attribute name "{0}" is invalid for entity "{1}". Attributes cannot begin with characters: $_

Explanation

The Microsoft ADO.NET Data Service specification does not allow entity properties to begin with $ or _ characters.

User response

Rename the attribute field or property defined in the entity descriptor XML file or entity metadata class to a name that does not start with a reserved character and restart the eXtreme Scale grid and the REST data service.
Related tasks:
Configuring
Administering

CWOBJ4016E

RESTSERVICE_METADATA_INVALIDNAME_ENTITY_CWOBJ4016E=CWOBJ4016E: The entity name "{0}" is invalid. Entity names cannot begin with characters: $_

Explanation

The Microsoft ADO.NET Data Service specification does not allow entity sets or entity types to begin with $ or _ characters.

User response

Rename the entity metadata class or the name in the entity descriptor XML file to a name that does not start with a reserved character. Restart the eXtreme Scale grid and the REST data service.
Related tasks:
Configuring
Administering

CWOBJ4017E

RESTSERVICE_INCOMPATIBLE_VERSION_CWOBJ4017E=CWOBJ4017E: Version {0} of WebSphere eXtreme Scale runtime is incompatible with version {1} of the REST data service.

Explanation

The REST data service (wxsrestservice.war) must be the same version as the eXtreme Scale runtime.

User response

Redeploy the wxsrestservice.war file from the install_root/ObjectGrid/restservice/lib directory that matches the eXtreme Scale runtime version.
Related tasks:
Configuring
Administering

CWOBJ4018E

RESTSERVICE_METADATA_INVALID_CASCADE_REMOVE_CWOBJ4018E=CWOBJ4018E: The association "{0}" is is invalid for entity "{1}". Many-to-one and many-to-many associations cannot be configured to cascade remove operations.

Explanation

Only the single-side of a bi-directional association, such as a one-to-one or one-to-many association can be configured to cascade remove operations.

User response

Remove the CascadeType.REMOVE annotation from the specified entity association field or property or remove the cascade-type attribute in the entity descriptor XML file for the entity in the eXtreme Scale configuration.
Related tasks:
Configuring
Administering

CWOBJ4019E

RESTSERVICE_METADATA_INVALID_CASCADE_MULTI_CWOBJ4019E=CWOBJ4019E: The association "{0}" for entity "{1}" and the association "{2}" for entity "{3}" are invalid. A cascade remove can only be configured on one end of a bi-directional association.

Explanation

Cascade remove can only be applied to one end of a bi-directional association.

User response

Remove the CascadeType.REMOVE annotation from the specified entity association field or property or remove the cascade-type attribute in the entity descriptor XML file for the entity in the eXtreme Scale configuration.
Related tasks:
Configuring
Administering

CWOBJ4020E

RESTSERVICE_CONFIG_INVALID_OBJECTGRID_NAME_CWOBJ4020E=CWOBJ4020E: The "{0}" property in the REST service properties file contains an incorrect value. At least one ObjectGrid name must be specified.

Explanation

The property in the REST service properties file contains an incorrect value. At least one ObjectGrid name must be specified.

User response

Specify a valid ObjectGrid name and restart the REST service.
Related tasks:
Configuring
Administering

CWOBJ4021E

RESTSERVICE_CONFIG_INVALID_PROPERTY_FILE_CWOBJ4021E=CWOBJ4021E: The REST service properties file "{0}" was not found on the file system or the class path.

Explanation

The REST service properties file was not found on the file system or the class path.

User response

Ensure the REST service properties file exists on the file system or the class path and restart the REST service.
Related tasks:
Configuring
Administering

CWOBJ4022E

RESTSERVICE_CONFIG_INVALID_OBJECT_GRID_NAME_CWOBJ4022E=CWOBJ4022E: The ObjectGrid "{0}" does not exist or is not started and will not be exposed via the REST service.

Explanation

An ObjectGrid with the specified name was specified in the REST service properties file, but is not available in the catalog service.

User response

Specify a valid ObjectGrid name exists in the "objectGridNames" property in the eXtreme Scale REST service properties and that the ObjectGrid with the specified name is started.
Related tasks:
Configuring
Administering

CWOBJ4023E

RESTSERVICE_CONFIG_EXCEP_CLIENT_OBJECT_GRID_XML_CWOBJ4023E=CWOBJ4023E: The exception "{0}" was encountered when loading the client ObjectGrid override XML file "{1}" from the class path.

Explanation

An exception was encountered when loading the client ObjectGrid override XML file from the class path. Check to make sure the file path is correct.

User response

Check to make sure the client ObjectGrid override XML file in the REST service properties is correct and restart the REST service.
Related tasks:
Configuring
Administering

CWOBJ4024E

RESTSERVICE_CLIENT_SECURITY_CONFIG_GEN_PROPS_CWOBJ4024E=CWOBJ4024E: The eXtreme Scale REST service is configured to use ObjectGrid client security but the "credentialGeneratorProps" property is not defined in the "{0}" file.

Explanation

When the eXtreme Scale REST service is configured to use ObjectGrid client security, the "credentialGeneratorProps" property must be defined in the ObjectGrid client property file.

User response

Check the "credentialGeneratorProps" property in the "{0}" file and make sure it is defined.
Related tasks:
Configuring
Administering

CWOBJ4025E

RESTSERVICE_REST_SECURITY_LOGINTYPE_CWOBJ4025E=CWOBJ4025E: The eXtreme Scale REST service is configured to use REST security with an incorrect "{0}" property of "{1}". A "{0}" of "{2}" will be used.

Explanation

When the eXtreme Scale REST service is configured to use REST security the "{0}" property must contain a value of "basic" or "none".

User response

Check the REST service "{0}" property to ensure that it contains a valid value of "basic" or "none".
Related tasks:
Configuring
Administering

CWOBJ4026E

RESTSERVICE_REST_SECURITY_INCORRECT_MAXRESULTS_CWOBJ4026E=CWOBJ4026E: The eXtreme Scale REST service "maxResultsForCollection" config property has an incorrect value of"{0}". The default value of unlimited will be used.

Explanation

The eXtreme Scale REST service "maxResultsForCollection" config property has an incorrect value. The value must be a positive integer that is greater than or equal to zero.

User response

Check the eXtreme REST service "maxResultsPerCollection" config property to ensure that it contains a valid value that is greater than or equal to zero.
Related tasks:
Configuring
Administering

CWOBJ4027W

RESTSERVICE_MBEAN_INCORRECT_OPERATION_PARAM_CWOBJ4027W=CWOBJ4027W: The eXtreme Scale REST service MBean operation "{0}" was invoked with an incorrect paramter of "{1}". The current value will be used.

Explanation

The eXtreme Scale REST service MBean operation "{0}" was invoked with an incorrect paramter of "{1}". The "{0}" operation must be invoked with a parameter of "true" or "false".

User response

Invoke the eXtreme Scale REST service MBean operation "{0}" with a value of "true" or "false".
Related tasks:
Configuring
Administering

CWOBJ4028I

RESTSERVICE_MBEAN_TRACE_DYNAMIC_CWOBJ4028I=CWOBJ4028I: The eXtreme Scale REST service debug tracing was set to "{0}" dynamically.

Explanation

The eXtreme Scale REST service debug tracing can be enabled or disabled through operations on the RestServiceMBean.

User response

If the action is desired, no action is required. If the action is not desired, debug tracing can be enavled or disabled through opertions on the RestServiceMBean.
Related tasks:
Configuring
Administering

CWOBJ4029W

RESTSERVICE_MBEAN_INCORRECT_TRACESPEC_CWOBJ4029W=CWOBJ4029W: The eXtreme Scale REST service MBean operation "{0}" was invoked with an incorrect paramter of "{1}". The current value will be used.

Explanation

The eXtreme Scale REST service MBean operation "{0}" was invoked with an incorrect paramter of "{1}". The "{0}" operation must be invoked with valid debug trace specification string.

User response

The eXtreme Scale REST service MBean operation "{0}" must be invoked with a correct debug trace specification, such as ObjectGridRest*=all=enabled.
Related tasks:
Configuring
Administering

CWOBJ4030W

RESTSERVICE_CONFIG_MALFORMED_LINE_CWOBJ4030W=CWOBJ4030W: The malformed line: "{0}" was encountered when loading the REST service properties file: "{1}".

Explanation

A syntactically incorrect line was encountered when loading the eXtreme Scale REST service properties file and will be ignored.

User response

Correct the line and restart the REST service.
Related tasks:
Configuring
Administering

CWOBJ4031W

RESTSERVICE_CONFIG_INVALID_VALUE_CWOBJ4031W=CWOBJ4031W: Invalid value: "{0}" was encountered on line: "{1}" when loading the rest service properties file: "{2}". Expected value is: "{3}".

Explanation

An incorrect value was encountered on the specified line when loading the rest service properties file and will be ignored.

User response

Correct the line and restart the REST service.
Related tasks:
Configuring
Administering

CWOBJ4032E

RESTSERVICE_CONFIG_EXCEPTION_PARSING_FILE_CWOBJ4032E=CWOBJ4032E: The exception: "{0}" was encountered when loading the REST service properties file: "{1}".

Explanation

An exception was encountered when loading the REST service properties file.

User response

Check to make sure the REST service properties file is formatted correctly and encoded in ASCII format and restart the REST service.
Related tasks:
Configuring
Administering

CWOBJ4033W

RESTSERVICE_CONFIG_INVALID_GRIDNAME_CWOBJ4033W=CWOBJ4033W: Invalid grid name: "{0}" was encountered on line: "{1}" when loading the rest service properties file: "{2}". Please specify name of an existing ObjectGrid.

Explanation

An incorrect value was encountered on the specified line when loading the rest service properties file and will be ignored.

User response

Specify a valid ObjectGrid name and restart the REST service.
Related tasks:
Configuring
Administering

CWOBJ4034W

RESTSERVICE_CONFIG_INVALID_ENTITYNAME_CWOBJ4034W=CWOBJ4034W: Invalid entity name: "{0}" was encountered on line: "{1}" when loading the rest service properties file: "{2}". Please specify name of an existing entity within ObjectGrid: "{3}".

Explanation

An incorrect value was encountered on the specified line when loading the rest service properties file and will be ignored.

User response

Specify a valid entity name of the mentioned ObjectGrid and restart the REST service.
Related tasks:
Configuring
Administering

CWOBJ4500I

DYNACACHE_PROVIDER_INITIALIZED_CWOBJ4500=CWOBJ4500I: WebSphere eXtreme Scale Dynamic Cache provider is successfully initialized.

Explanation

The provider is ready to create Dynamic Cache instances.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ4501E

DYNACACHE_CREATION_FAILURE_CWOBJ4501=CWOBJ4501E: The WebSphere eXtreme Scale Dynamic Cache provider encountered an error while creating the following cache instance: {0}.

Explanation

The provider was unable to complete cache creation for the specified Dynamic Cache instances.

User response

Check to make sure all required JAR files are on the class path and required configuration parameters are set properly.
Related tasks:
Configuring
Administering

CWOBJ4502E

MISSING_REQUIRED_CONFIGURATION_PARAMETER_CWOBJ4502=CWOBJ4502E: Missing the following required configuration parameter: {0}.

Explanation

A required configuration parameter has not been set or has been set with an invalid value. The WebSphere eXtreme Scale Dynamic Cache provider cannot initialize.

User response

Set the required configuration parameter to a supported value and restart the WebSphere Application Server process.
Related tasks:
Configuring
Administering

CWOBJ4503E

DYNACACHE_PROVIDER_FAILED_INIT_CWOBJ4503=CWOBJ4503E: WebSphere eXtreme Scale Dynamic Cache provider failed to initialize successfully.

Explanation

A fatal error occured during initialization. The provider is unable to create Dynamic Cache instances.

User response

Look for more information in first failure data capture (FFDC) and error logs. Restart the WebSphere Application Server process when problems are corrected. See the Logs and Trace section in the information center for JVM log location.
Related tasks:
Configuring
Administering

CWOBJ4504W

DYNACACHE_UNEXPECTED_SPECIAL_VALUE_CWOBJ4504=CWOBJ4504W: Cache Entry is tagged as a Special Value. Value is being ignored.

Explanation

This should not happen under normal circumstances. It indicates unsecure access to Dynamic Cache data stored on standalone ObjectGrid containers.

User response

Check access controls to the machines or networks where your ObjectGrid containers are located.
Related tasks:
Configuring
Administering

CWOBJ4505W

DYNACACHE_CONFIG_MISMATCH_CWOBJ4505=CWOBJ4505W: Dynamic Cache configuration sent from provider does not match currently stored configuration for cache {0}. Stored configuration is {1}. Received configuration is {2}.

Explanation

This is caused by different settings on two or more WebSphere Application Server instances that are sharing a distributed dynacache instance. This may also be encountered and safely ignored when ripple starting after a configuration change.

User response

Make sure that every server using the Dynamic Cache instance has the same configuration.
Related tasks:
Configuring
Administering

CWOBJ4506I

DYNACACHE_EVICTOR_FAILOVER_CWOBJ4506=CWOBJ4506I: Configuration found in map. ObjectGrid shard is becoming primary after a failover. Setting Dynamic Cache Evictor configuration. Configuration: {0}

Explanation

The Dynamic Cache Evictor is being configured after a failover.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ4507E

DYNACACHE_INCORRECT_PARAMETER_FORMAT_CWOBJ4507=CWOBJ4507E: The value {1} set for an optional configuration parameter {0} is invalid.

Explanation

An optional configuration parameter has been configured with an incorrect value and is being ignored.

User response

Remove or set the configuration parameter to an acceptable value and restart the WebSphere Application Server.
Related tasks:
Configuring
Administering

CWOBJ4508I

DYNACACHE_CREATED_CWOBJ4508=CWOBJ4508I: The WebSphere eXtreme Scale provider has created a Dynamic Cache instance with name {0} using topology {1}.

Explanation

A Dynamic Cache instance has been successfully created.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ4509E

DYNACACHE_UNSUPPORTED_REPLICATION_POLICY_CWOBJ4509=CWOBJ4509E: The WebSphere eXtreme Scale Dynamic Cache provider does not support the {0} replication policy for Cache {1} with key {2}.

Explanation

A Dynamic Cache entry has been created with a replication policy that is not supported.

User response

Update the application or cachespec file to set a supported replication policy for the entry.
Related tasks:
Configuring

Administering

CWOBJ4510E

DYNACACHE_REQUIRES_SERVER_CWOBJ4510=CWOBJ4510E: WebSphere eXtreme Scale Server is required to create Dynamic Cache instances with topology {0}. Cache name is {1}.

Explanation

The provider was unable to complete cache creation for the specified Dynamic Cache instances.

User response

Set the cache topology to remote or install WebSphere eXtreme Scale Server.
Related tasks:
Configuring
Administering

CWOBJ4511E

DYNACACHE_GRID_DISCONNECTED_CWOBJ4511=CWOBJ4511E: The WebSphere eXtreme Scale Dynamic Cache provider has become disconnected from {0} WebSphere eXtreme Scale grid and {1} map: {2}

Explanation

The provider has become disconnected from the specified grid and map.

User response

Look for any additional errors in the log. Also review the first failure data capture (FFDC) logs.
Related tasks:
Configuring
Administering

CWOBJ4512I

DYNACACHE_GRID_RECONNECTED_CWOBJ4512=CWOBJ4512I: The WebSphere eXtreme Scale Dynamic Cache provider has reconnected with {0} WebSphere eXtreme Scale grid and {1} map.

Explanation

The provider has been reconnected the specified grid and map.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ4541I

MEMORYSTATS_ENHANCED_SIZING_IN_USE_CWOBJ4541=CWOBJ4541I: Enhanced BackingMap memory sizing is enabled.

Explanation

The USED BYTES statistics for maps will have enhanced accuracy.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ4542I

MEMORYSTATS_DEFAULT_SIZING_IN_USE_CWOBJ4542=CWOBJ4542I: Basic BackingMap memory sizing is enabled.

Explanation

The USED BYTES statistics for maps will be less accurate.

User response

If using JDK 1.5 or later, consult the information center for instructions on enabling the sizing agent.
Related tasks:
Configuring
Administering

CWOBJ4543W

MEMORYSTATS_OBJECT_TOO_COMPLEX_CWOBJ4543=CWOBJ4543W: The size of an object of type {0} is not accurate.

Explanation

The USED BYTES statistics may underestimate the size of the map. Treat the result as a trend instead a precise value.

User response

See the Java object caching concepts and statistics articles in the information center and simplify the Object accordingly.
Related tasks:
Configuring
Administering

CWOBJ4551E

EVICTION_TRIGGER_NOT_SUPPORTED_CWOBJ4551=CWOBJ4551E: The eviction trigger {0} cannot be used with the current Java Virtual Machine configuration {1}.

Explanation

The current Java Virtual Machine configuration is known to be unstable when combined with the eviction trigger.

User response

See the Eviction section in the information center for evictions triggers and use a supported Java Virtual Machine and configuration.
Related concepts:
Evictors
Related tasks:
Configuring
Administering
Writing a custom evictor
Enabling evictors with XML configuration

CWOBJ4552W

EVICTION_TRIGGER_NOT_STABLE_CWOBJ4552=CWOBJ4552W: The eviction trigger {0} might not behave as expected when used with the Java Virtual Machine setting {1}.

Explanation

The current Java Virtual Machine configuration is known to be unstable when combined with the eviction trigger.

User response

See the Eviction section in the information center for evictions triggers and suggested Java Virtual Machine configurations.
Related concepts:
Evictors
Related tasks:
Configuring

Administering
Writing a custom evictor
Enabling evictors with XML configuration

CWOBJ4560W

QUERY_CACHE_MAX_ENTRIES_CWOBJ4560=CWOBJ4560W: The query queue cache of ObjectGrid {0} reached the maximum size of {1}. Eviction of the query queues will occur based on the Least Recently Used rule. This message will only be logged for the first eviction.

Explanation

The query cache will automatically start evicting the least recently used queries.

User response

To use the query cache more efficiently reuse query strings when possible. See the Query performance tuning section in the information center for more information on using the query cache.
Related tasks:
Configuring
Administering

CWOBJ4561W

QUERY_CACHE_MAX_ENTRIES_CWOBJ4561=CWOBJ4561W: The query queue cache of ObjectGrid {0} for partition {1} reached the maximum size of {2}. Eviction of the query queues will occur based on the Least Recently Used rule. This message will only be logged for the first eviction.

Explanation

The query cache will automatically start evicting the least recently used queries.

User response

To use the query cache more efficiently reuse query strings when possible. See the Query performance tuning section in the information center for more information on using the query cache.
Related tasks:
Configuring
Administering

CWOBJ4600E

GET_ATTRIBUTES_EXCEPTION_CWOBJ4600=CWOBJ4600E: Exception {1} occurred on the getAttribute for {0}. Continuing to create attribute list.

Explanation

An exception occurred on getting an attribute. A null value will be set for the failing attribute and the rest of the attribute will be created.

User response

Review the exception and correct any configuration related errors.
Related tasks:
Configuring
Administering

CWOBJ4601E

SET_ATTRIBUTES_EXCEPTION_CWOBJ4601=CWOBJ4601E: Exception {1} occurred on the setAttribute for {0}. Continuing to set other attributes.

Explanation

An exception occurred on setting an attribute. The attribute listed in the message will be skipped and the rest of the attributes set.

User response

Review the exception and correct any configuration related errors.
Related tasks:
Configuring

Administering

CWOBJ4650I

DISK_EVICTOR_DETECTED_CWOBJ4650=CWOBJ4650I: The Evictor is using disk based persistance at the following URI {0}.

Explanation

The Evictor is using disk based storage instead of memory.

User response

No action is required
Related tasks:
Configuring
Administering

CWOBJ4651W

DISK_OFFLOAD_CWOBJ4651=CWOBJ4651W: Persistence directory {0} already exists but does not contain valid data. It will be cleared.

Explanation

The directory to be used for grid data storage already existed but contained corrupted or unrelated data. The directory will be emptied so it can be used for grid data.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ4652W

DISK_OFFLOAD_CWOBJ4652=CWOBJ4652W: Persistence directory {0} cannot be opened because it is in use by another process.

Explanation

The directory to be used for grid data storage is already being accessed by another process, this process cannot open it.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ4700I

DYNAMIC_MAP_CREATED_CWOBJ4700=CWOBJ4700I: The map name {0} matched the regular expression of template map {1}. The {0} map has been created for ObjectGrid {2}.

Explanation

A new dynamic map has been successfully created.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ4701I

TEMPLATE_MAP_CONFIGURED_CWOBJ4701=CWOBJ4701I: Template map {0} is configured in ObjectGrid {1}.

Explanation

A template map configuration has been detected.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ4702E

DYNAMIC_MAP_CREATION_ERROR_CWOBJ4702=CWOBJ4702E: Dynamic creation failed for map {0} due to the following exception {1}.

Explanation

A fatal error occured during dynamic map creation.

User response

Examine the exception for more detail. Correct the problem that caused the exception and retry dynamic map creation.
Related tasks:
Configuring
Administering

CWOBJ4800E

SHARD_ALREADY_RESERVED_ERROR_CWOBJ4800=CWOBJ4800E: Could not reserve shard {0} on container {1} because this shard is already reserved by container {2}.

Explanation

Only one container can reserve a shard at a time.

User response

Release the shard from its owning container before attempting to reserve it with another container.
Related tasks:
Configuring
Administering

CWOBJ4801W

PARTITION_NOT_FOUND_CWOBJ4801=CWOBJ4801W: No shard for partition {0} was released from container {1} because this container has not reserved a shard from this partition.

Explanation

A container can only release shards that are currently placed on it.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ4802E

RESERVE_PARTITION_NON_EXISTENT_CWOBJ4802=CWOBJ4802E: Attempt to reserve shard {0} on container {1} has failed because the partition does not exist.

Explanation

A partition does not exist for the shard that you attempted to reserve.

User response

Attempt to reserve shards for valid partitions. For the FIXED_PARTITIONS placement strategy, valid partitions are only in the range less than the total number of partitions defined in the deployment policy. See the Deployment topology configuration section in the information center for more information partitions and deployment policies.
Related tasks:
Configuring
Administering

CWOBJ4803E

PER_CONTAINER_UNSUPPORTED_CWOBJ4803=CWOBJ4803E: The shard reservation feature is not supported with PER_CONTAINER placement strategy or scope.

Explanation

Shards can only be reserved when using the FIXED_PARTITIONS placement strategy with PER_DOMAIN scope.

User response

Do not attempt to reserve or release shards while using PER_CONTAINER placement strategy or scope.
Related tasks:
Configuring
Administering

CWOBJ4804I

SUCCESSFUL_SHARD_RESERVATION_CWOBJ4804=CWOBJ4804I: Shard {0} was successfully reserved on container {1}.

Explanation

Request to reserve shard was executed successfully.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ4805I

SUCCESSFUL_SHARD_RELEASE_CWOBJ4805=CWOBJ4805I: Shard from partition {0} was successfully released from container {1}.

Explanation

Request to release shard was executed successfully.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ4806I

SHARD_RESERVED_PRIOR_INIT_PLACEMENT_CWOBJ4806=CWOBJ4806I: Shard {0} has been reserved on container {1} prior to initial placement. Shard will be placed onto this container when initial placement occurs.

Explanation

A shard was reserved on the container, but initial placement has not occurred. Initial placement can be gated by several factors. The number of started containers remaining less than the numInitialContainers configured is the most common.

User response

If numInitialContainers is configured, start enough containers to trigger placement. See the Deployment topology configuration section in the information center for more information on numInitialContainers and deployment policies.
Related tasks:
Configuring
Administering

CWOBJ4807E

RESERVE_CONTAINER_NOT_SUPPORTING_MAPSET_CWOBJ4807=CWOBJ4807E: Shard {0} cannot be reserved on container {1} because this container does not support map set {2}.

Explanation

This container does not support the map set specified.

User response

Only reserve shards that a container can support. Optionally, provide an ObjectGrid XML and a deployment policy XML that include the desired ObjectGrid and map set.
Related tasks:
Configuring
Administering

CWOBJ4808I

ROLE_SWAP_SUCCESSFUL_CWOBJ4808=CWOBJ4808I: Request from shard {0} to swap roles with a(n) {1} shard was processed successfully. This shard is now a(n) {1}.

Explanation

This shard requested a role swap with another shard. The request executed successfully. The shards have exchanged roles.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ4809W

ROLE_SWAP_SAME_TYPE_CWOBJ4809=CWOBJ4809W: Request from shard {0} to swap roles with a(n) {1} shard failed to execute because this shard is already a(n) {1}

Explanation

A shard can only swap roles with a shard of another type.

User response

Attempt to swap roles with a shard of another type.
Related tasks:
Configuring
Administering

CWOBJ4810E

ROLE_SWAP_SHARD_TYPE_NOT_PLACED_CWOBJ4810=CWOBJ4810E: Request from shard {0} to swap roles with a(n) {1} shard has failed because no {1} from this partition is currently placed.

Explanation

Only currently placed shards can swap roles.

User response

Attempt to swap roles with a shard that is currently placed.
Related tasks:
Configuring
Administering

CWOBJ4811E

ROLE_SWAP_SHARD_TIMEOUT_CWOBJ4811=CWOBJ4811E: Request from shard {0} to swap roles with a(n) {1} shard has timed out. Shard {0} did not inherit its new role in the allotted time.

Explanation

The shard was able to contact the PlacemeService and request the swap. However, it did not inherit its new role before timing out.

User response

Examine the logs and FFDC for errors that may indicate why the role swap couldn't complete in the allotted time.
Related tasks:
Configuring
Administering

CWOBJ4812E

ROLE_SWAP_SHARD_NOT_FOUND_ON_CONTAINER_CWOBJ4812=CWOBJ4812E: Request from shard {0} to swap roles with the {1} shard on container {2} has failed. No {1} shard was found on the specified container for this partition.

Explanation

Shard of the type specified was not found on the container specified.

User response

Specify a container hosting the shard type desired or use null as the container argument for random selection of shard type specified.
Related tasks:
Configuring
Administering

CWOBJ4813E

ROLE_SWAP_INVALID_CONTAINER_CWOBJ4813=CWOBJ4813E: No container was found to match the name {2}. Request from shard {0} to swap roles with the {1} shard on container {2} has failed.

Explanation

The PlacementService is not aware of a container that matches the name specified.

User response

Specify a valid container that is currently hosting a shard of the desired type.
Related tasks:
Configuring
Administering

CWOBJ4814E

ROLE_SWAP_PER_CONTAINER_SCOPE_NOT_SUPPROTED_CWOBJ4814=CWOBJ4814E: The shard {0} has a placement scope of per container.

Explanation

Shards with per container placement scope only have primary roles.

User response

Perform swap roles on shards with per domain placement scope with replicas defined.
Related tasks:
Configuring
Administering

CWOBJ4900E

MAPSET_INCOMPATIBLE_PARTITION_NUM_CWOBJ4900=CWOBJ4900E: Domain {0} will not send updates to domain {1} for map set {2} from ObjectGrid {3} because of a mismatch in the number of partitions. The map set is configured for {4} partitions in domain {0} and {5} partitions in domain {1}.

Explanation

The map set must be configured with the same number of partitions in each domain in order to achieve a multi-data center topology.

User response

Stop the containers hosting the map set in one of the domains. Restart these containers using a deployment policy containing the map set with the proper number of partitions.
Related tasks:
Configuring
Administering

CWOBJ4901E

MAPSET_INCOMPATIBLE_PLACEMENT_STRAT_CWOBJ4901=CWOBJ4901E: Domain {0} will not send updates to domain {1} for map set {2} from ObjectGrid {3} because of a mismatch in the placement strategy. The map set is configured for {4} placement strategy in domain {0} and {5} placement strategy in domain {1}.

Explanation

The map set must be configured with the same placement strategy in each domain in order to achieve a multi-data center topology.

User response

Stop the containers hosting the map set in one of the domains. Restart these containers using a deployment policy containing the map set with the proper placement strategy.
Related tasks:
Configuring
Administering

CWOBJ4902I

MAPSET_COMPATIBLE_CWOBJ4902=CWOBJ4902I: This domain ({0}) received a compatible map set from domain {1}. Updates for map set {2} from ObjectGrid {3} will be sent to domain {1}.

Explanation

The foreign domain published a map set that is compatible with the local domain. The local domain will now send updates regarding this map set to the foreign domain.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ4903I

FOREIGN_DOMAINS_FOUND_CWOBJ4903=CWOBJ4903I: The following foreign domains have been provided: {0}

Explanation

This domain will attempt to link to the foreign domains for the purpose of supporting a multi-data center topology.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ4904I

FOREIGN_DOMAIN_END_POINTS_CWOBJ4904=CWOBJ4904I: The following end points have been provided for foreign domain {0}: {1}

Explanation

This domain will attempt to link to the foreign domain using the end points provided.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ4905E

MAPSET_WRONG_NUM_MAPS_CWOBJ4905=CWOBJ4905E: {0} from linked domains do not contain the same maps. While domain {1} contains the following maps for this map set: {2}, domain {3} contains: {4}.

Explanation

The map set must be configured with the same maps in each domain in order to achieve a multi-data center topology.

User response

Stop the containers hosting the map set in one of the domains. Restart these containers using a deployment policy and ObjectGrid XML containing the same maps in the map set as found in the linked domain.
Related tasks:
Configuring
Administering

CWOBJ4906E

MAPSET_MISSING_MAP_CWOBJ4906=CWOBJ4906E: Domain {0} will not send updates to domain {1} for map set {2} from ObjectGrid {3}. The maps in the map set are not consistent. The {4} map was found in the map set for domain {5}, but not for domain {6}.

Explanation

The map set must contain the same maps in each domain.

User response

Stop the containers hosting the map set in one of the domains. Restart these containers using a deployment policy and ObjectGrid XML containing the same maps in the map set as found in the linked domain.
Related tasks:
Configuring
Administering

CWOBJ4907E

FOREIGN_ENDPOINTS_NOT_FOUND_CWOBJ4907=CWOBJ4907E: No end points were provided for {0} foreign domain, which was expecting {2} property. No attempt will be made to establish a link between the {1} and {0} domains.

Explanation

A set of end points must accompany each foreign domain. The foreign domain provided will not be linked to this domain.

User response

Define end points for this foreign domain in the server properties file and restart the catalog server.
Related tasks:
Configuring
Administering

CWOBJ4908E

LOCAL_DOMAIN_INCLUDED_IN_FOREIGN_CWOBJ4908=CWOBJ4908E: The local domain name {0} was found in the set of foreign domains in the server properties.

Explanation

The domainName provided in the server properties must not be included in the set of foreign domains in the same server properties.

User response

Remove the domain name from the set of foreign domains and restart the catalog server.
Related tasks:
Configuring
Administering

CWOBJ4909E

WRITE_BEHIND_MAP_FOUND_CWOBJ4909=CWOBJ4909E: Domain {0} will not send updates to domain {1} for map set {2} from ObjectGrid {3} because the {4} map is configued for write-behind support in {5}.

Explanation

Map sets containing maps with write-behind loaders are not supported in a multi-primary environment.

User response

Remove the write-behind loader from all maps in the map set to enable primaires in both domains for this map set. No action is required if you do not wish to support a multi-primary topology with this map set.
Related tasks:
Configuring
Administering

CWOBJ4910E

COPY_TO_BYTES_FOUND_CWOBJ4910=CWOBJ4910E: Domain {0} will not send updates to domain {1} for {2}:{3} because the {4} map is configured as a byte array map in {5}.

Explanation

Map sets containing byte array maps are not supported in a multi-primary environment.

User response

Remove byte array maps from the map set in order to enable primaires in both domains for this map set. No action is required if you do not wish to support a multi-primary topology with this map set.
Related tasks:
Configuring
Administering

CWOBJ4911E

ENTITY_MAP_FOUND_CWOBJ4911=CWOBJ4911E: Domain {0} will not send updates to domain {1} for {2}:{3} because the {4} map is backing an entity in {5}.

Explanation

Map sets containing entities are not supported in a multi-primary environment.

User response

Remove entities from the map set in order to enable primaires in both domains for this map set. No action is required if you do not wish to support a multi-primary topology with this map set.
Related tasks:
Configuring
Administering

CWOBJ4912E

KEYTYPE_BYTES_FOUND_CWOBJ4912=CWOBJ4912E: The {0} local domain will not send updates to the {1} foreign domain for {2}:{3} because the {4} map is configured as a bytes-for-keys map in the {5} domain that contains the bytes for keys.

Explanation

Map sets containing bytes for keys are not supported in an environment with multiple primary shards.

User response

Remove bytes-for-keys maps from the map set to enable primary shards in both domains for this map set. No action is required if you do not want to support a multiple primary shard topology with this map set.
Related tasks:
Configuring
Administering

CWOBJ4913I

FOREIGN_DOMAIN_NOT_AVAILABLE_CWOBJ4913=CWOBJ4913I: During an attempt to {0} the foreign catalog service for foreign domain {1} could not be reached. The procedure completed in the local domain but was not completed in the foreign domain.

Explanation

An action was triggered that normally requires interaction between catalog servers of different domains. That action was not able to coordinate that change with the foreign domain but was able to complete locally.

User response

If the foreign domain is still available, run the action on the foreign domain separately. If the foreign domain is not running, no action is necessary.
Related tasks:
Configuring
Administering

CWOBJ4914W

DOMAIN_PING_FAILURE_CWOBJ4914=CWOBJ4914W: Attempt to ping foreign domain, {0}, failed. The ping will be attempted again in {1} milliseconds.

Explanation

The foreign domain is unreachable.

User response

Restart the unreachable foreign domain in order to replicate data between domains.
Related tasks:
Configuring
Administering

CWOBJ4915I

DOMAIN_PING_SUCCESS_CWOBJ4915=CWOBJ4915I: This domain successfully pinged the foreign domain, {0}.

Explanation

This domain successfully reached the foreign domain.

User response

No user action required.

Related tasks:
Configuring
Administering

CWOBJ4916I

FOREIGN_DOMAIN_RECYCLED_CWOBJ4916=CWOBJ4916I: The local domain detected that domain {0} has been restarted after being unavailable for some time.

Explanation

The foreign domain has been recycled. An exchange of data between domains will be initiated.

User response

No user action required.
Related tasks:
Configuring
Administering

CWOBJ4917I

DOMAIN_PING_SUCCESSFUL_AFTER_UNSUCCESSFUL_CWOBJ4917=CWOBJ4917I: This domain successfully pinged the foreign domain, {0}, after {1} consecutive unsuccessful attempts.

Explanation

This domain successfully reached the foreign domain after a series of unsuccessful attempts.

User response

No user action required.
Related tasks:
Configuring
Administering

CWOBJ5000I

RA_CONNECTED_CWOBJ5000I=CWOBJ5000I: The WebSphere eXtreme Scale resource adapter connected to the {0} grid at the following catalog server endpoints: {1}

Explanation

A Java EE Connector (J2C) managed connection established a client connection to an eXtreme Scale data grid. A client connection is established when the com.ibm.websphere.xs.ra.XSConnectionFactory.getConnection() method is invoked.

User response

No user action is required.
Related tasks:
Configuring
Administering

CWOBJ5001I

RA_DISCONNECTED_CWOBJ5001I=CWOBJ5001I: The WebSphere eXtreme Scale resource adapter disconnected from the {0} grid at the following catalog server endpoints: {1}

Explanation

A resource adapter uses the XSConnectionFactory connection factory to establish a client connection to a remote eXtreme Scale grid. The connection ended because the resource adapter stopped or the resetConnection operation on the management bean started.

User response

The eXtreme Scale client connection is automatically restored when the first com.ibm.websphere.xs.ra.XSConnection connection is created.

Related tasks:
Configuring
Administering

CWOBJ6400I

OSGI_NEW_SERVICE_ADDED_CWOBJ6400=CWOBJ6400I: The {0} OSGi service with the service ranking {1} from the {2} service ID is available.

Explanation

An OSGi service is available to the JVM.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ6401I

OSGI_SERVICE_REMOVED_CWOBJ6401=CWOBJ6401I: The {0} OSGi service with service ranking {1} from the {2} service ID is no longer available.

Explanation

An OSGi service is not available to the JVM.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ6402W

OSGI_SERVICE_NOT_FOUND_CWOBJ6402=CWOBJ6402W: The {0} OSGi service with service ranking {1} from the {2} service ID cannot be found in the eXtreme Scale runtime environment.

Explanation

The OSGi service cannot be found in the eXtreme Scale runtime environment. This error indicates an internal error.

User response

Examine the first failure data capture (FFDC) logs for errors.
Related tasks:
Configuring
Administering

CWOBJ6403I

OSGI_SERVICE_USED_CWOBJ6403=CWOBJ6403I: The {0} OSGi service with service ranking {1} from the {2} service ID is used by the eXtreme Scale runtime.

Explanation

An OSGi service is used by the eXtreme Scale runtime.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ6404I

OSGI_SERVICE_ALREADY_USED_CWOBJ6404=CWOBJ6404I: The {0} OSGi service with service ranking {1} has already been used. The service ID is {2}.

Explanation

An OSGi service with a specific service ranking has already been used by the eXtreme Scale runtime environment. No action will occur.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ6405I

OSGI_BUNDLE_ACTIVATED_CWOBJ6405=CWOBJ6405I: The eXtreme Scale OSGi bundle with the {0} symbolic name is activated.

Explanation

The eXtreme Scale OSGi bundle is activated.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ6406I

OSGI_BUNDLE_STOPPED_CWOBJ6406=CWOBJ6406I: The eXtreme Scale OSGi bundle with the {0} symbolic name is stopped.

Explanation

The eXtreme Scale OSGi bundle is stopped.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ6407W

OSGI_SERVICE_UPGRADE_WARNING_CWOBJ6407=CWOBJ6407W: The OSGi service upgrade did not find a client identifier for the {0} data grid.

Explanation

The identifier for the client initiating the upgrade was incorrectly removed.

User response

Verify that the upgrade completes as expected.
Related tasks:
Configuring
Administering

CWOBJ6408W

SHARD_SCOPE_THREADLOCAL_WARNING_CWOBJ6408=CWOBJ6408W: In the Spring framework, the thread local {0} value for the custom shard scope is not null. It is {1}.

Explanation

In the Spring framework, always clear the thread local {0} value for the custom shard scope in a finally block.

User response

Add the code to clear the thread local value in a finally block.
Related tasks:
Configuring
Administering

CWOBJ6409W

SERVICE_DESTROY_FAILED_CWOBJ6409=CWOBJ6409W: When a new OSGi service is used, the destroy() call on the old service instance {0} throws an exception with the following message: {1}

Explanation

The destroy() call on the old OSGi service fails with an exception. WebSphere eXtreme Scale runtime logs this exception and continues the processing.

User response

For details about the exception stack, check the FFDC directory. Check the destroy() method implementation to see why an exception was thrown. Make sure the error condition is handled in the destroy() method because the eXtreme Scale runtime environment will not throw this exception again.
Related tasks:
Configuring
Administering

CWOBJ6410E

SERVICE_UPDATE_FAILED_CWOBJ6410=CWOBJ6410E: The update for the {0} OSGi service of the {1} ObjectGrid to service ranking {2} failed. The failure is logged and ignored. Error: {3}

Explanation

The OSGi service ranking update failed.

User response

Use OSGi administrative functions to update the OSGi service to a workable service.
Related tasks:
Configuring
Administering

CWOBJ6411W

REPOSITORY_SERVICE_RANKING_USED_CWOBJ6411=CWOBJ6411W: For the {0} ObjectGrid, the OSGi metadata repository is currently using service ranking {1} for service {2}, which is not the highest service ranking {3} for this JVM. The ObjectGrid instance uses service ranking {4} for this service.

Explanation

By default, the highest ranking service available in the JVM is used. However, the other JVMs use a lower ranking service. To be consistent, this JVM attempts to use the lower ranking service too.

User response

Determine why the other JVMs do not have the highest ranking service as this JVM does.
Related tasks:
Configuring
Administering

CWOBJ6412W

MULTI_BLUEPRINT_SERVICE_FOUND_CWOBJ6412=CWOBJ6412W: The following OSGi blueprint container classes are found: {0}.

Explanation

Typically, install only one blueprint container in an OSGi container.

User response

Check the installed bundles of the OSGi container to see why multiple OSGi blueprint container classes are installed. Install one blueprint container to avoid conflicts.
Related tasks:
Configuring
Administering

CWOBJ6413W

NUMBER_SERVICES_NOT_MATCH_AFTER_UPDATE_CWOBJ6413=CWOBJ6413W: After updating the {0} OSGi service from the old service ranking {1} to the new service ranking {2}, the number of service instances is changed from {3} to {4}.

Explanation

One possible explanation is that the new service might be referencing a bean that is using a different scope.

User response

Check the plug-in service bundle to make sure the scope of the bean referenced by the service does not change between different versions.
Related tasks:
Configuring
Administering

CWOBJ6414W

SERVER_PROPERTIES_UPDATED_CWOBJ6414=CWOBJ6414W: The server property file is updated to {0} using OSGi Configuration Admin while the eXtreme Scale server is running. The update does not take effect until the server is restarted.

Explanation

Server property file update is ignored while the server is running. However, if the server is restarted, the new configured server property file will be used.

User response

Determine why the server property file was updated. Make sure that an operation error does not exist.
Related tasks:
Configuring
Administering

CWOBJ6415E

BUNDLE_RESTART_NOT_ALLOWED_CWOBJ6415=CWOBJ6415E: Restarting the eXtreme Scale (XS) bundles is not allowed.

Explanation

The Object Request Broker (ORB) does not allow for dynamic starting and stopping of its consumers which prevents XS from being restarted without also restarting the Java Virtual Machine (JVM).

User response

Exit the current JVM and restart the process with the XS bundle.
Related tasks:
Configuring
Administering

CWOBJ7000I

STATS_MAP_INJECTION_GRID_ENABLED_CWOBJ7000=CWOBJ7000I: ObjectGrid {0} is enabled for storing historic statistics on container "{1}".

Explanation

ObjectGrid is enabled to track and store historic statistics within the grid for monitoring.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ7001I

STATS_MAP_INJECTION_GRID_DISABLED_CWOBJ7001=CWOBJ7001I: ObjectGrid {0} is disabled for storing historic statistics on container "{1}".

Explanation

ObjectGrid is disabled to track and store historic statistics. The ObjectGrid will not be capable of being monitored by the Extreme Scale monitoring user interface.

User response

No action is required.
Related concepts:
Statistics overview
Related tasks:
Configuring
Administering
Monitoring with managed beans (MBeans)
Related reference:
Server properties file

CWOBJ7002I

STATS_MAP_INJECTION_MAPSET_ENABLED_CWOBJ7002=CWOBJ7002I: ObjectGrid:MapSet {0}:{1} is enabled for storing historic statistics on container "{2}".

Explanation

ObjectGrid's mapset is enabled to track and store historic statistics within the grid for monitoring.

User response

No action is required.
Related concepts:
Statistics overview
Related tasks:
Configuring
Administering
Monitoring with managed beans (MBeans)
Related reference:
Server properties file

CWOBJ7003I

STATS_MAP_INJECTION_MAPSET_DISABLED_CWOBJ7003=CWOBJ7003I: ObjectGrid:MapSet {0}:{1} is disabled for storing historic statistics on container "{2}".

Explanation

ObjectGrid's mapset is disabled to track and store historic statistics. The maps within the mapset will not be capable of being monitored by the Extreme Scale monitoring user interface.

User response

No action is required.
Related concepts:
Statistics overview
Related tasks:
Configuring
Administering
Monitoring with managed beans (MBeans)
Related reference:
Server properties file

CWOBJ7006I

DynamicPort=CWOBJ7006I: ObjectGrid server agent generated dynamic port {0}.

Explanation

ObjectGrid server agent generated dynamic port for HAManager.

User response

No action is required.
Related concepts:
Statistics overview
Related tasks:
Configuring
Administering
Monitoring with managed beans (MBeans)
Related reference:
Server properties file

CWOBJ7100E

STATS_COLLECTOR_CWOBJ7100=CWOBJ7100E: Could not locate internal ObjectGrid information map for the following shard:{0}. Ensure that the grid and mapset is appropriately enabled for historic statistics monitoring.

Explanation

Statistic monitoring infrastructure injects a very small information map into each ObjectGrid enabled for historic stats monitoring in order to provide and track information about active shards. If the information map can not be located, historic statistics monitoring will fail for that shard.

User response

Ensure that the grid and mapset is appropriately enabled for historic statistic monitoring. Review the statsSpec setting in server properties file. See the Managed MBean usage overview section in the information center for more information on using statistics.
Related concepts:
Statistics overview
Related tasks:
Configuring
Administering
Monitoring with managed beans (MBeans)
Related reference:
Server properties file

CWOBJ7101E

STATS_COLLECTOR_CWOBJ7101=CWOBJ7101E: Statistic monitoring infrastructure could not find any maps associated with ObjectGrid {0}. No monitoring will be performed on an empty ObjectGrid

Explanation

Statistic monitoring infrastructure monitors ObjectGrids configured with maps which store client data. If the monitoring infrastructure can not locate maps within the ObjectGrid, no monitoring can be performed.

User response

Ensure that the ObjectGrid deployed has map definitions.
Related concepts:
Statistics overview
Related tasks:
Configuring
Administering
Monitoring with managed beans (MBeans)
Related reference:
Server properties file

CWOBJ7102E

STATS_COLLECTOR_CWOBJ7102=CWOBJ7102E: Statistic monitoring infrastructure could not retrieve statistics for path {0}. Ensure statistics tracking is enabled for this process.

Explanation

Statistic monitoring infrastructure monitors ObjectGrids, their maps, and jvm level statistics, however requires that statistics tracking be enabled for the process. Ensure statistics tracking is enabled such that they can be monitored and historically tracked.

User response

Ensure that the statistics tracking is enabled for this process in the server properties file. Review the statsSpec setting in server properties file. See the Managed MBean usage overview section in the information center for more information on using statistics.
Related concepts:
Statistics overview
Related tasks:
Configuring
Administering
Monitoring with managed beans (MBeans)
Related reference:
Server properties file

CWOBJ7103I

STATS_COLLECTOR_ROUTING_ADDITION_PROCESSED_CWOBJ7103=CWOBJ7103I: Statistic charting can now display charts using statistics from partition {0}.

Explanation

The statistics charting function has received the proper connections to pull statistics for the given ObjectGrid partition.

User response

No action is required.
Related concepts:
Statistics overview
Related tasks:
Configuring
Administering
Monitoring with managed beans (MBeans)
Related reference:
Server properties file

CWOBJ7104I

STATS_COLLECTOR_ROUTING_DELETION_PROCESSED_CWOBJ7104=CWOBJ7104I: Statistic charting has been informed to remove its reference to partition {0}.

Explanation

The statistics charting function has received indication that stop pulling statistics for the given ObjectGrid partition.

User response

No action is required.
Related concepts:
Statistics overview
Related tasks:
Configuring
Administering
Monitoring with managed beans (MBeans)
Related reference:
Server properties file

CWOBJ7200I

DeadServer=CWOBJ7200I: Detected the failure of server ({0}) in core group ({1}).

Explanation

Server failure is detected and reported to catalog service.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ7201I

NewServer=CWOBJ7201I: Detected the addition of new server ({0}) in core group ({1}).

Explanation

New server is detected and reported to catalog service.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ7203I

NewLeader=CWOBJ7203I: Leader changed. New leader ({0}) is elected in core group ({1}) and reported to the catalog service.

Explanation

New leader is elected and reported to the catalog service.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ7205I

CWOBJ7205=CWOBJ7205I: Server, {0}, has sent a membership change notice that will be rejected as this member has already been removed from the core group.

Explanation

The server was removed from the grid's core group because it was previously unreachable, the new action sent to this server will be ignored.

User response

Restart this server as it is no longer a member of the grid.
Related tasks:
Configuring
Administering

CWOBJ7211I

CWOBJ7211=CWOBJ7211I: As a result of a heartbeat (view heartbeat type) from leader {0} for core group {1} with member list {2}, the server {3} is being removed from the core group view.

Explanation

The server was removed from the core group because it was deemed unreachable.

User response

Investigate whether the becomes network accessible to the server sending the heartbeat, and if not, restart the server and if possible address accessibility.
Related tasks:

Configuring
Administering

CWOBJ7212I

HA_STANDALONE_DEFINED_SET_EXPORTED_CWOBJ7212I=CWOBJ7212I: The catalog server has sent an updated defined set with version {0} and host:port list {1} to the following list of servers: {2}.

Explanation

The defined set indicates the list of servers that the catalog server has currently assigned to a given core group.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ7213W

HA_SPLIT_PARTITION_DETECTED_CWOBJ7213W=CWOBJ7213W: The core group {0} received a heart beat notfication from the server {1} with revision {2} and a current view listing {3} and previous listing {4} - such a combination indicates a partitioned core group.

Explanation

Resource contentation, such as networking issues or Java garbage collection, can lead to eXtreme Scale processes being unable to communicate with each other. As such, the core group could be divided into multiple core groups, each with a leader.

User response

Correction of the resource contention or problem is strongly suggested.
Related tasks:
Configuring
Administering

CWOBJ7214I

HA_DEFINED_SET_VIEW_DIFFERENCE_IGNORED_CWOBJ7214I=CWOBJ7214I: While processing a container heart beat for the core group {0}, a difference between the defined set and view was detected. However, since the previous and current views are the same, {1}, this difference can be ignored.

Explanation

Especially during concurrent server starts, there are potential windows where defined set and view discrepancies can be ignored. But we note the condition for potential debug purposes.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ7300W

CONTAINER_FAILED_BOOTSTRAP_CWOBJ7300=CWOBJ7300W: This server failed to bootstrap to a catalog server at the following address(es): {0}. Will retry in {1} ms.

Explanation

This server must be able to reach an active catalog server in order to initialize itself.

User response

Ensure that a catalog server is active and reachable prior to launching this container server.
Related tasks:

Configuring
Administering

CWOBJ7301E

CONTAINER_TIMEOUT_BOOTSTRAP_CWOBJ7301=CWOBJ7301E: This server failed to start because it exceeded the maximum allowable number of retry attempts for bootstraping to a catalog server.

Explanation

This server must be able to reach an active catalog server within 24 attempts in order to initialize itself.

User response

Ensure that a catalog server is active and reachable prior to launching this container server.
Related tasks:
Configuring
Administering

CWOBJ7400E

CATALOGSERVICE_ENDPOINTS_INCONSISTENT_ORDER_CWOBJ7400=CWOBJ7400E: The decision to honor the order of catalogServiceEndPoints must be consistent across the catalog servers in the domain. This server ({0}) will be stopped. Exception detected: {1}

Explanation

Each catalog server in the domain must have the same value for the ordered argument/property.

User response

Restart your catalog servers. Ensure that every catalog server in the domain has set the ordered argument/property consistently.
Related tasks:
Configuring
Administering

CWOBJ7401E

CATALOGSERVICE_ENDPOINTS_STRING_INCONSISTENT_CWOBJ7401=CWOBJ7401E: A discrepancy in the catalogServiceEndPoints value was detected. The catalogServiceEndPoints value must be the same on each catalog server. This server ({0}) will be stopped. Exception detected: {1}

Explanation

Each catalog server in the domain must use the same catalogServiceEndPoints. Order is important when the ordered argument/property is true.

User response

Ensure that every catalog server in the domain is using the same catalogServiceEndPoints. Restart this catalog server with catalogServiceEndPoints that are consistent with those used by other catalog servers in the domain.
Related tasks:
Configuring
Administering

CWOBJ7402I

CATALOGSERVICE_ENDPOINTS_ORDERED_CWOBJ7402=CWOBJ7402I: This catalog server is honoring the order of catalogServiceEndPoints.

Explanation

The ordered argument/property was set to true for this server.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ7403E

UNABLE_TO_START_EXTREME_SCALE_TRANSPORT_CWOBJ7403=CWOBJ7403E: The eXtreme Scale transport did not start.

Explanation

An unexpected exception occurred while starting the eXtreme Scale transport.

User response

Examine the JVM log files and FFDC for errors that might indicate why the transport did not start.
Related tasks:
Configuring
Administering

CWOBJ7404I

OFFHEAP_ENABLED_CWOBJ7404=CWOBJ7404I: Off-heap memory storage is enabled for the {0} server.

Explanation

The offHeapEnabled property was set to "true" for this server.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ7405E

FAILED_TO_GET_EVICTION_LIST_CWOBJ7405=CWOBJ7405E: Failed to get eviction list from off-heap address.

Explanation

An internal error occurred failing to get eviction list from off-heap address.

User response

Retrieve the problem determination information for debugging eXtremeMemory errors.
Related tasks:
Configuring
Administering

CWOBJ7406W

XM_NO_CONTAINER_CWOBJ7406=CWOBJ7406W: No container named {0} hosted on this server.

Explanation

Trying to replicate to a container that no longer exists on this server.

User response

Verify that the replica failed over to another container correctly. If failover was not successful, retrieve the problem determination information for debugging eXtremeMemory errors.
Related tasks:
Configuring
Administering

CWOBJ7407W

XM_NO_SHARD_CWOBJ7407=CWOBJ7407W: No shard named {0} hosted on this server.

Explanation

Trying to replicate to a shard that no longer exists on this server.

User response

Verify that the replica failed over to another shard correctly. If failover was not successful, retrieve the problem determination information for debugging eXtremeMemory errors.
Related tasks:
Configuring
Administering

CWOBJ7408E

ERROR_STARTING_XIO_TRANSPORT_CWOBJ7408=CWOBJ7408E: Caught exception starting eXtremeIO transport service.

Explanation

An unexpected error occurred starting the eXtremeIO transport service.

User response

Verify there are no port conflicts with the port chosen. Otherwise, retrieve the must gathers for debugging eXtremeMemory errors.
Related tasks:
Configuring
Administering

CWOBJ7409E

ERROR_STARTING_LOADING_XM_NATIVE_LIBRARIES_CWOBJ7409=CWOBJ7409E: Caught exception starting eXtremeMemory due to missing native libraries.

Explanation

Native libraries are required to be loaded when using eXtremeMemory.

User response

Verify you are using a supported platform for eXtremeMemory.
Related tasks:
Configuring
Administering

CWOBJ7500W

ROUTE_TABLE_PARTITION_PURGE_CWOBJ7500=CWOBJ7500W: Partition {0} will be removed from the route table because that partition entry is stale.

Explanation

This partition was found in the route table. However, the placement service is no longer tracking it.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ7501I

ROUTE_TABLE_UPDATES_CWOBJ7501=CWOBJ7501I: The following partitions listed by the form grid:mapSet:partitionId:gridEpoch:partitionEpoch just had their routing entries update: {0}.

Explanation

As a result of placement work recently sent to a set of containers, the list of partitions listed are the ones whose routing entries were received from the various containers and subsequently processed and made available to clients.

User response

No actoin is required.
Related tasks:
Configuring
Administering

CWOBJ7600E

CANNOT_SERIALIZE_VALUE_CWOBJ7600=CWOBJ7600E: Cannot serialize cache entry value {0}. Serialization failed.

Explanation

An error occured during the serialization of a cache entry value.

User response

Ensure the value class is serializable.
Related tasks:
Configuring
Administering

CWOBJ7601E

CANNOT_SERIALIZE_KEY_CWOBJ7601=CWOBJ7601E: Cannot serialize cache entry key {0}. Serialization failed.

Explanation

An error occured during the serialization of a cache entry key.

User response

Ensure the key class is serializable.
Related tasks:
Configuring
Administering

CWOBJ7602E

CATALOG_SERVICE_DOMAIN_BEAN_INITIALIZATION_FAIL_CWOBJ7602=CWOBJ7602E: Object Grid Catalog Service Domain Bean failed to initialize. Exception occured {0}

Explanation

The attempt to initialize the Object Grid Catalog Service Domain Bean failed. The configuration may be incorrect or the WebSphere eXtreme Scale environment may be unreacable.

User response

Review the exception, resolve the error and retry the operation.
Related tasks:
Configuring
Administering

CWOBJ7603E

CLIENT_BEAN_INITIALIZATION_FAIL_CWOBJ7603=CWOBJ7603E: Object Grid Client Bean failed to initialize. Exception occured {0}

Explanation

The attempt to initialize the Object Grid Client Bean failed. The Object Grid name may be incorrect.

User response

Review the exception, resolve the error and retry the operation.
Related tasks:

Configuring
Administering

CWOBJ7700I

PeerManagerStart=CWOBJ7700I: Peer Manager service started successfully in server ({0}) with core group ({1}).

Explanation

Peer Manager service started successfully.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ7800I

Start_HAController=CWOBJ7800I: Start ObjectGrid HA Controller with core group ({0}), host ({1}), and port ({2}).

Explanation

Start ObjectGrid High Availability Controller.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ8000I

Register_CWOBJ8000=CWOBJ8000I: Registration is successful with zone ({0}) and coregroup of ({1}).

Explanation

This process succesfully registered with the specified zone and the catalog service successfully allocated a coregroup for this process.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ8009E

Failed_Register_CWOBJ8009=CWOBJ8009E: Registration failed for zone ({0})

Explanation

This process failed to register with catalog service.

User response

Examine other messages and exceptions in this log and the first failure data capture (FFDC), resolve the problem and restart the process.
Related tasks:
Configuring
Administering

CWOBJ8101I

StandbyCatalogServerCreated_CWOBJ8101=CWOBJ8101I: Notify that standby catalog service is created with domain= {0} and with IOR= {1}

Explanation

Standby Catalog Service is created and notified.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ8102I

MasterCatalogServerCreated_CWOBJ8102=CWOBJ8102I: Notify that master catalog service is created with domain= {0} and with IOR= {1}

Explanation

Master Catalog Service is created and notified.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ8106I

MasterCatalogServerActivated_CWOBJ8106=CWOBJ8106I: The master catalog service cluster activated with cluster {0}

Explanation

Master catalog service is activated.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ8108I

ResentStandbyCatalogServer_CWOBJ8108=CWOBJ8108I: Re-send standby catalog service on the request of master catalog service with domain= {0} and IOR= {1}

Explanation

Tell master catalog service who are replica catalog services

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ8109I

UpdateCatalogServerCluster_CWOBJ8109=CWOBJ8109I: Updated catalog service cluster {0} from server {1} with entry {2}

Explanation

Tell master catalog service who are replica catalog services

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ8401I

WaitForReplica_CWOBJ8401=CWOBJ8401I: Waiting for a server replica to be started. Start another server(s) immediately.

Explanation

Waiting for a server replica to be started. Start another server(s) immediately.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ8601I

PeerServers_CWOBJ8601=CWOBJ8601I: PeerManager found peers of size {0}

Explanation

Processing batch jobs for this coregroup.

User response

No action is required.
Related tasks:
Configuring
Administering

CWOBJ9000I

ENGLISH_ONLY_INFO_MESSAGE_CWOBJ9000=CWOBJ9000I: This message is an English-only Informational message: {0}.

Explanation

This informational message is not translated.

User response

See message for details.
Related tasks:
Configuring
Administering

CWOBJ9001W

ENGLISH_ONLY_WARN_MESSAGE_CWOBJ9001=CWOBJ9001W: This message is an English-only Warning message: {0}.

Explanation

This warning message is not translated.

User response

See message for details.
Related tasks:
Configuring
Administering

CWOBJ9002E

ENGLISH_ONLY_ERROR_MESSAGE_CWOBJ9002=CWOBJ9002E: This message is an English only Error message: {0}.

Explanation

This error message is not translated.

User response

See message for details.
Related tasks:
Configuring
Administering

CWPRJ: WebSphere eXtreme Scale messages for the Projector component for entity and tuple projection

When you encounter a message in a log or other parts of the product interface, look up the message by its message ID to find out more information.

- CWPRJ0001E
  INVALID_ARGUMENT_NULL_CWPRJ0001E=CWPRJ0001E: Invalid value detected. The value for field {0} must be a valid non-null value.
- CWPRJ1001E
  INVALID_SUBSET_ATTRIBUTE_MISSING_CWPRJ1001E=CWPRJ1001E: Invalid subset attribute. Attribute does not exist on superset entity. Entity={0}, Entity Class={1}, Attribute={2}
- CWPRJ1002E
  INVALID_SUBSET_ATTRIBUTE_EXTRA_CWPRJ1002E=CWPRJ1002E: Invalid subset attribute. Extra attribute defined: Entity={0}, Entity Class={1}, Attribute={2}
- CWPRJ1003E
  INVALID_SUBSET_ATTRIBUTE_TYPEMISMATCH_CWPRJ1003E=CWPRJ1003E: Invalid subset attribute. Attribute or association type mismatch: Entity={0}, Entity Class={1}, Attribute={2}
- CWPRJ1004E
  INVALID_ASSOCIATION_KEY_CARDINALITY_CWPRJ1004E=CWPRJ1004E: Invalid association. Associations that are also keys can only have uni-directional OneToOne or ManyToOne relationships.
- CWPRJ1005E
  ERROR_RESOLVING_ASSOCIATION_CWPRJ1005E=CWPRJ1005E: Error resolving entity association. Entity={0}, association={1}.
- CWPRJ1006E
  ERROR_LOADING_ENTITY_CLASS_CWPRJ1006E=CWPRJ1006E: The class {0} cannot be found when loading entity {1}.
- CWPRJ1007E
  INVALID_ENTITY_DEFINITION_CWPRJ1007E=CWPRJ1007E: Invalid entity definition for class: {0}. {1}
- CWPRJ1008E
  INVALID_ENTITY_DEFINITION_TOPLEVEL_CWPRJ1008E=CWPRJ1008E: {0} class must be defined as a top level class.
- CWPRJ1009E
  INVALID_ENTITY_DEFINITION_FINAL_CWPRJ1009E=CWPRJ1009E: {0} class must not be final
- CWPRJ1010E
  INVALID_ENTITY_DEFINITION_CONSTRUCTOR_CWPRJ1010E=CWPRJ1010E: {0} class must define a default, public or protected, no-argument constructor.
- CWPRJ1011E
  INVALID_ENTITY_DEFINITION_PUBLIC_CWPRJ1011E=CWPRJ1011E: {0} class must be public.
- CWPRJ1012E
  INVALID_ENTITY_DEFINITION_COMPARABLE_CWPRJ1012E=CWPRJ1012E: IdClass {0} must define equals() and hashCode() methods.
- CWPRJ1013E
  INVALID_ENTITY_DEFINITION_NOMETADATA_CWPRJ1013E=CWPRJ1013E: No metadata information was defined for the entity {0}.
- CWPRJ1014E
  INVALID_SUPERSET_ENTITY_NOMETADATA_CWPRJ1014E=CWPRJ1014E: No superset metadata information was defined for entity {0} with superset {1}.
- CWPRJ1015E
  ENTITY_DEFINITION_EXCEPTION_CWPRJ1015E=CWPRJ1015E: An exception occurred while creating the entity configuration for an annotated class or XML metadata for entity: {0}.
- CWPRJ1016E
  INVALID_IDCLASS_DEFINITION_MISSINGATTRIBUTE_CWPRJ1016E=CWPRJ1016E: Key class {0} must define attribute: {1}.
- CWPRJ1017E
  INVALID_IDCLASS_DEFINITION_INVALIDATTRIBUTETYPE_CWPRJ1017E=CWPRJ1017E: Key class {0} attribute {1} type is incorrect. Declared type is: {2}. Required type is: {3}.
- CWPRJ1020E
  INVALID_ORDERBY_SPECIFIED_CWPRJ1020E=CWPRJ1020E: Error in OrderBy configuration. Field {3} in entity {1} does not exist. Check attribute {2} of entity {0}.
- CWPRJ1021E
  INVALID_VERSION_TYPE_SPECIFIED_CWPRJ1021E=CWPRJ1021E: Error in version type. Specified version type of {0} is not supported.
- CWPRJ1022W
  INVALID_ASSOCIATION_REFERENCE_CWPRJ1022W=CWPRJ1022W: Association reference from entity: {0} attribute {1}, to entity: {2} for attribute key(s)

{3}, association key {4} could not be found.

- CWPRJ1023E
  INVALID_ENTITY_ACCESSTYPE_CWPRJ1023E=CWPRJ1023E: Invalid entity access-type specified: {0}. The entity class does not have an @Id annotation, and the entity descriptor file does not define the access type for the entity.
- CWPRJ1024E
  INVALID_XML_FILE_CWPRJ1024E=CWPRJ1024E: The XML file is invalid. A problem has been detected with {0} at line {1}. The error message is {2}.
- CWPRJ1025E
  MULTIPLE_ENTITY_ACCESSTYPE_CWPRJ1025E=CWPRJ1025E: Unable to determine entity access type. Both fields and properties are annotated.
- CWPRJ1026E
  MISSING_ATTRIBUTES_CWPRJ1026E=CWPRJ1026E: No attributes or associations defined.
- CWPRJ1027E
  MISSING_ENTITYCLASS_XML_CWPRJ1027E=CWPRJ1027E: Entity class: {0} does not exist in entity descriptor file: {1}.
- CWPRJ1029E
  MULTIPLE_SCHEMAROOTS_CWPRJ1029E=CWPRJ1029E: Multiple schema root references detected for Entity class: {0}, First Root Class: {1}, Second Root Class: {2}.
- CWPRJ1030E
  INVALID_COMPOSITE_INDEX_DEFINITION_CWPRJ1030E=CWPRJ1030E: Invalid composite index definition for entity: {0}. Either composite index name: {1} or attributeNames: {2} is empty or is not unique for the entity: {0}.
- CWPRJ1031E
  COLLECTION_ATTRIBUTE_NOT_SUPPORTED_CWPRJ1031E=CWPRJ1031E: The multi-valued association: {0} is not supported in composite HashIndex: {1} for entity: {2}.
- CWPRJ1032E
  EM_CLASSLESS_NOID_CWPRJ1032E=CWPRJ1032E: The Entity configuration {0} does not contain a valid id.
- CWPRJ1033E
  EM_CLASSLESS_MIXED_CONFIGURATION_CWPRJ1033E=CWPRJ1033E: The Entity configuration {0} contains both a class reference and a classless identifier marked with an @ symbol.
- CWPRJ1100E
  INVALID_ATTRIBUTE_CWPRJ1100E=CWPRJ1100E: Invalid attribute or association: {0}.
- CWPRJ1101E
  MISSING_FIELD_CWPRJ1101E=CWPRJ1101E: Field is undefined for class: {0}.
- CWPRJ1102E
  MISSING_PROPERTY_CWPRJ1102E=CWPRJ1102E: Property is undefined for class: {0}.
- CWPRJ1103E
  MISSING_ASSOCIATION_TARGET_ENTITY_CWPRJ1103E=CWPRJ1103E: The target entity is undefined.
- CWPRJ1104E
  DUPLICATE_FIELDPROPERTY_CWPRJ1104E=CWPRJ1104E: Attribute is defined more than once.
- CWPRJ1105E
  MULTIPLE_INVERSE_ASSOCIATIONS_CWPRJ1105E=CWPRJ1105E: The target association has more than one inverse relationship to this entity and is missing the MappedBy relationship definition. Source entity: {0}, Inverse entity: {1}, Duplicate attributes: {2}
- CWPRJ1108E
  INVALID_INVERSE_ASSOCIATION_TYPE_CWPRJ1108E=CWPRJ1108E: The inverse target association references an invalid entity type. Inverse, target entity: {0}, association name: {1}
- CWPRJ1109E
  INVALID_ENTITY_DEFINITION_MISSING_TARGET_CWPRJ1109E=CWPRJ1109E: The target entity type of {0} is not defined.
- CWPRJ1110E
  INVALID_INVERSE_KEY_ASSOCIATION_CWPRJ1110E=CWPRJ1110E: An association that is also a key must not have an inverse association. Inverse, target entity: {0}, association name: {1}
- CWPRJ1111E
  INVERSE_ASSOCIATION_TYPE_MISMATCH_CWPRJ1111E=CWPRJ1111E: The inverse target association must match the source target type. Inverse, target entity: {0}, association name: {1}
- CWPRJ1112E
  TARGET_ENTITY_NOT_DEFINED_CWPRJ1112E=CWPRJ1112E: Target entity not defined for field or property: {0}.
- CWPRJ1113E
  ENTITY_ATTRIBUTE_MISSING_CARDINALITY_CWPRJ1113E=CWPRJ1113E: Attribute {0} is an entity type but a relationship to the entity is not defined.
- CWPRJ1114E
  ENTITY_ATTRIBUTE_NOT_SERIALIZABLE_CWPRJ1114E=CWPRJ1114E: Attribute {0} of type {1} is not serializable.
- CWPRJ1115E
  INVALID_MANY_ASSOCIATION_TYPE_CWPRJ1115E=CWPRJ1115E: Invalid OneToMany or ManyToMany association type of: {0}.
- CWPRJ1200I
  PROJECTOR_INSTRUMENTATION_ENABLED_CWPRJ1200I=CWPRJ1200I: Projector entity class instrumentation is enabled. The instrumentation mode is {0}.
- CWPRJ1201E
  FIELD_ACCESS_ENTITY_NOT_INSTRUMENTED_CWPRJ1201E=CWPRJ1201E: Field-access entity not instrumented. Entity class={0}.
- CWPRJ1202W
  PROXY_UNAVAILABLE_CWPRJ1202W=CWPRJ1202W: Entity proxy support is unavailable.
- CWPRJ1300E
  MULTIPLE_METHODS_PER_TYPE_CWPRJ1300E=CWPRJ1300E: Multiple methods listen to the same event {0} in class {1}.
- CWPRJ1301E
  ENTITY_CALLBACK_NO_PARAM_CWPRJ1301E=CWPRJ1301E: Entity lifecycle callback method, {0}, defined in the entity class {1} must have no parameters.
- CWPRJ1302E
  LISTENER_CALLBACK_ONE_PARAM_CWPRJ1302E=CWPRJ1302E: Entity lifecycle callback method, {0}, defined in the entity listener class {1} must have only one parameter.
- CWPRJ1303E
  LISTENER_UNASSIGNABLE_TYPE_CWPRJ1303E=CWPRJ1303E: The parameter of the entity callback method, {0}, is not assignable to entity {1}.

- CWPRJ1304E
  NO_SUCH_CALLBACK_METHOD_CWPRJ1304E=CWPRJ1304E: The method name {0} with parameter {1} does not exist in the class {2}.
- CWPRJ1305E
  UNEXPECTED_CALLBACK_EXCEPTION_CWPRJ1305E=CWPRJ1305E: Unexpected exception encounted when invoking the lifecycle callback method {0}: {1}
- CWPRJ5000I
  INSTRUMENTATION_ENABLED_CWPRJ5000I=CWPRJ5000I: Java instrumentation mechanism is enabled. The instrumentation mode is {0}.
- CWPRJ9000I
  ENGLISH_ONLY_INFO_MESSAGE_CWPRJ9000=CWPRJ9000I: This message is an English-only Informational message: {0}.
- CWPRJ9001W
  ENGLISH_ONLY_WARN_MESSAGE_CWPRJ9001=CWPRJ9001W: This message is an English-only Warning message: {0}.
- CWPRJ9002E
  ENGLISH_ONLY_ERROR_MESSAGE_CWPRJ9002=CWPRJ9002E: This message is an English only Error message: {0}.

Related reference:
Deprecated properties and APIs

CWPRJ0001E

INVALID_ARGUMENT_NULL_CWPRJ0001E=CWPRJ0001E: Invalid value detected. The value for field {0} must be a valid non-null value.

Explanation

The specified field or argument must not be null.

User response

Change the method or object call to set the appropriate argument or verify that the field value is properly set.
Related reference:
Deprecated properties and APIs

CWPRJ1001E

INVALID_SUBSET_ATTRIBUTE_MISSING_CWPRJ1001E=CWPRJ1001E: Invalid subset attribute. Attribute does not exist on superset entity. Entity={0}, Entity Class={1}, Attribute={2}

Explanation

The specified entity attribute or association must be defined on the subset entity.

User response

Review the metadata for the entity attribute and verify that the name and type match the superset entity.
Related reference:
Deprecated properties and APIs

CWPRJ1002E

INVALID_SUBSET_ATTRIBUTE_EXTRA_CWPRJ1002E=CWPRJ1002E: Invalid subset attribute. Extra attribute defined: Entity={0}, Entity Class={1}, Attribute={2}

Explanation

The specified entity attribute or association does not exist on the superset entity.

User response

Review the metadata for the entity attribute and verify that the name and type match the superset entity and all key attributes are defined.
Related reference:
Deprecated properties and APIs

CWPRJ1003E

INVALID_SUBSET_ATTRIBUTE_TYPEMISMATCH_CWPRJ1003E=CWPRJ1003E: Invalid subset attribute. Attribute or association type mismatch: Entity={0}, Entity Class={1}, Attribute={2}

Explanation

The specified entity attribute or association exists on the supserset entity, but is of a different type or cardinality.

User response

Review the metadata for the entity attribute and verify that the name and type match the superset entity.
Related reference:
Deprecated properties and APIs

CWPRJ1004E

INVALID_ASSOCIATION_KEY_CARDINALITY_CWPRJ1004E=CWPRJ1004E: Invalid association. Associations that are also keys can only have uni-directional OneToOne or ManyToOne relationships.

Explanation

Entities cannot use multi-valued associations as keys. All keys must be attributes or uni-directional OneToOne or ManyToOne relationships.

User response

Review the metadata for the entity key associations and verify that the entity does not use OneToMany or ManyToMany associations.
Related reference:
Deprecated properties and APIs

CWPRJ1005E

ERROR_RESOLVING_ASSOCIATION_CWPRJ1005E=CWPRJ1005E: Error resolving entity association. Entity={0}, association={1}.

Explanation

The specified entity association is not valid. The metadata processor was unable to create the entity because of this error.

User response

Check the nested exception message for details on how to resolve the error and resubmit the operation.
Related reference:
Deprecated properties and APIs

CWPRJ1006E

ERROR_LOADING_ENTITY_CLASS_CWPRJ1006E=CWPRJ1006E: The class {0} cannot be found when loading entity {1}.

Explanation

The specified entity class or related class could not be loaded from the classpath.

User response

Verify that the class definition exists on the classpath.
Related reference:
Deprecated properties and APIs

CWPRJ1007E

INVALID_ENTITY_DEFINITION_CWPRJ1007E=CWPRJ1007E: Invalid entity definition for class: {0}. {1}

Explanation

The specified entity definition is invalid.

User response

Review the exception text, resolve the error and resubmit the operation.
Related reference:
Deprecated properties and APIs

CWPRJ1008E

INVALID_ENTITY_DEFINITION_TOPLEVEL_CWPRJ1008E=CWPRJ1008E: {0} class must be defined as a top level class.

Explanation

All entities must be top-level classes. They cannot be interfaces.

User response

Verify that the entity class is a top-level class and resubmit the operation.
Related reference:
Deprecated properties and APIs

CWPRJ1009E

INVALID_ENTITY_DEFINITION_FINAL_CWPRJ1009E=CWPRJ1009E: {0} class must not be final

Explanation

Entity classes must not have the final modifier specified.

User response

Verify that the entity class does not include the final class modifier and resubmit the operation.
Related reference:
Deprecated properties and APIs

CWPRJ1010E

INVALID_ENTITY_DEFINITION_CONSTRUCTOR_CWPRJ1010E=CWPRJ1010E: {0} class must define a default, public or protected, no-argument constructor.

Explanation

All entities must have a default, public or protected, no-argument constructor.

User response

Verify that the enity class has default, public or protected, no-argument constructor and resubmit the operation.
Related reference:
Deprecated properties and APIs

CWPRJ1011E

INVALID_ENTITY_DEFINITION_PUBLIC_CWPRJ1011E=CWPRJ1011E: {0} class must be public.

Explanation

All entities must have a public modifier.

User response

Verify that the entity class has a public class modifier and resubmit the operation.
Related reference:
Deprecated properties and APIs

CWPRJ1012E

INVALID_ENTITY_DEFINITION_COMPARABLE_CWPRJ1012E=CWPRJ1012E: IdClass {0} must define equals() and hashCode() methods.

Explanation

Classes that are used as composite entity key classes must implement the equals() and hashCode() methods.

User response

Verify that the IdClass class provides public equals() and hashCode() methods and resubmit the operation.
Related reference:
Deprecated properties and APIs

CWPRJ1013E

INVALID_ENTITY_DEFINITION_NOMETADATA_CWPRJ1013E=CWPRJ1013E: No metadata information was defined for the entity {0}.

Explanation

An entity was referenced in XML, but no metadata exists for that entity.

User response

Verify that the entity class includes the @Entity annotation and at least one primary key is defined using the @Id annotation, or verify that the <id> XML attribute is defined in the entity descriptor XML file and resubmit the operation.
Related reference:
Deprecated properties and APIs

CWPRJ1014E

INVALID_SUPERSET_ENTITY_NOMETADATA_CWPRJ1014E=CWPRJ1014E: No superset metadata information was defined for entity {0} with superset {1}.

Explanation

An entity was defined with a superset entity, but no entity metadata was defined for the superset entity.

User response

Verify that the superset entity class includes the @Entity annotation and at least one primary key is defined using the @Id annotation, or verify that the <id> XML attribute is defined in the entity descriptor XML file for the superset entity and resubmit the operation.
Related reference:
Deprecated properties and APIs

CWPRJ1015E

ENTITY_DEFINITION_EXCEPTION_CWPRJ1015E=CWPRJ1015E: An exception occurred while creating the entity configuration for an annotated class or XML metadata for entity: {0}.

Explanation

There is an error in the metadata information that describes the entity characteristics.

User response

Check the nested exception message for details on how to resolve the error and resubmit the operation.
Related reference:
Deprecated properties and APIs

CWPRJ1016E

INVALID_IDCLASS_DEFINITION_MISSINGATTRIBUTE_CWPRJ1016E=CWPRJ1016E: Key class {0} must define attribute: {1}.

Explanation

Key classes must define all of the attributes from the referencing entity class.

User response

Verify that the class or entity metadata definition is valid and resubmit the operation.
Related reference:
Deprecated properties and APIs

CWPRJ1017E

INVALID_IDCLASS_DEFINITION_INVALIDATTRIBUTETYPE_CWPRJ1017E=CWPRJ1017E: Key class {0} attribute {1} type is incorrect. Declared type is: {2}. Required type is: {3}.

Explanation

Key classes must define all of the attributes with the same type from the referencing entity class.

User response

Verify that the class or entity metadata definition is valid and resubmit the operation.
Related reference:
Deprecated properties and APIs

CWPRJ1020E

INVALID_ORDERBY_SPECIFIED_CWPRJ1020E=CWPRJ1020E: Error in OrderBy configuration. Field {3} in entity {1} does not exist. Check attribute {2} of entity {0}.

Explanation

The specified field does not exist in the target entity.

User response

Verify the field exists.
Related reference:
Deprecated properties and APIs

CWPRJ1021E

INVALID_VERSION_TYPE_SPECIFIED_CWPRJ1021E=CWPRJ1021E: Error in version type. Specified version type of {0} is not supported.

Explanation

Specified version type is not supported.

User response

Review the eXtreme Scale documentation for the data types that can be used for version attributes. The documentation can be found at the following URL: http://publib.boulder.ibm.com/infocenter/wxdinfo/v7r0/index.jsp
Related reference:
Deprecated properties and APIs

CWPRJ1022W

INVALID_ASSOCIATION_REFERENCE_CWPRJ1022W=CWPRJ1022W: Association reference from entity: {0} attribute {1}, to entity: {2} for attribute key(s) {3}, association key {4} could not be found.

Explanation

The source entity contains a reference to an entity that has been removed. The source entity's association to the missing target entity is set to null.

User response

Remove the association from the source entity by setting the reference to null, remove the association from the source entity's collection, or persist the removed entity using the EntityManager.persist method.
Related reference:
Deprecated properties and APIs

CWPRJ1023E

INVALID_ENTITY_ACCESSTYPE_CWPRJ1023E=CWPRJ1023E: Invalid entity access-type specified: {0}. The entity class does not have an @Id annotation, and the entity descriptor file does not define the access type for the entity.

Explanation

Entities and entity identification classes must have an access-type of FIELD or PROPERTY. If the entity class has an @Id annotation, the projector will detect the access type by where (field or method) the @Id is annotated. If annotations are not used, the access type must be defined in the entity descriptor file.

User response

Review the entity metadata and resubmit the operation.
Related reference:
Deprecated properties and APIs

CWPRJ1024E

INVALID_XML_FILE_CWPRJ1024E=CWPRJ1024E: The XML file is invalid. A problem has been detected with {0} at line {1}. The error message is {2}.

Explanation

The XML file does not conform to the schema.

User response

Edit the XML file so that it is conforms to the schema. The schema is defined in the emd.xsd file included in any of the ObjectGrid Java archive files.
Related reference:
Deprecated properties and APIs

CWPRJ1025E

MULTIPLE_ENTITY_ACCESSTYPE_CWPRJ1025E=CWPRJ1025E: Unable to determine entity access type. Both fields and properties are annotated.

Explanation

Annotated entities can only have annotations specified on the field or property methods, but not both.

User response

Review the entity annotations and verify that the entity metadata annotations are only specified on the fields or get method of each attribute.
Related reference:
Deprecated properties and APIs

CWPRJ1026E

MISSING_ATTRIBUTES_CWPRJ1026E=CWPRJ1026E: No attributes or associations defined.

Explanation

All entities must have at least 1 attribute or association defined.

User response

Verify that the class or entity metadata definition is valid and resubmit the operation.
Related reference:
Deprecated properties and APIs

CWPRJ1027E

MISSING_ENTITYCLASS_XML_CWPRJ1027E=CWPRJ1027E: Entity class: {0} does not exist in entity descriptor file: {1}.

Explanation

The specified entity class could not be found.

User response

Verify that the class or entity metadata definition is valid and resubmit the operation.
Related reference:
Deprecated properties and APIs

CWPRJ1029E

MULTIPLE_SCHEMAROOTS_CWPRJ1029E=CWPRJ1029E: Multiple schema root references detected for Entity class: {0}, First Root Class: {1}, Second Root Class: {2}.

Explanation

Entities may only reference one schema root entity explicitly or implictly via its composite key.

User response

Verify that only one schema root entity is reachable from the key associations of the entity class and resubmit the operation.
Related reference:
Deprecated properties and APIs

CWPRJ1030E

INVALID_COMPOSITE_INDEX_DEFINITION_CWPRJ1030E=CWPRJ1030E: Invalid composite index definition for entity: {0}. Either composite index name: {1} or attributeNames: {2} is empty or is not unique for the entity: {0}.

Explanation

The specified composite index definition is invalid. The composite index name and attributeNames cannot be empty and must be unique.

User response

Verify that the composite index definition is valid, resolve the error and resubmit the operation.
Related reference:
Deprecated properties and APIs

CWPRJ1031E

COLLECTION_ATTRIBUTE_NOT_SUPPORTED_CWPRJ1031E=CWPRJ1031E: The multi-valued association: {0} is not supported in composite HashIndex: {1} for entity: {2}.

Explanation

A HashIndex configured as a composite index cannot include multi-valued associations such as collections.

User response

Verify that the attribute names of the composite HashIndex definition only include attributes and single-valued associations; then resubmit the operation.
Related reference:
Deprecated properties and APIs

CWPRJ1032E

EM_CLASSLESS_NOID_CWPRJ1032E=CWPRJ1032E: The Entity configuration {0} does not contain a valid id.

Explanation

The specified entity did not contain a valid id.

User response

Verify that there is at least one field marked with an id annotation or an association with id set to true.
Related reference:
Deprecated properties and APIs

CWPRJ1033E

EM_CLASSLESS_MIXED_CONFIGURATION_CWPRJ1033E=CWPRJ1033E: The Entity configuration {0} contains both a class reference and a classless identifier marked with an @ symbol.

Explanation

The Entity configuration contains both entities with class references and those with classless identifiers.

User response

Verify that the entity configuration is either completely classless or all classes are present and referenced.
Related reference:
Deprecated properties and APIs

CWPRJ1100E

INVALID_ATTRIBUTE_CWPRJ1100E=CWPRJ1100E: Invalid attribute or association: {0}.

Explanation

The specified attribute is invalid.

User response

Review the exception text, resolve the error and resubmit the operation.
Related reference:
Deprecated properties and APIs

CWPRJ1101E

MISSING_FIELD_CWPRJ1101E=CWPRJ1101E: Field is undefined for class: {0}.

Explanation

The specified field does not exist on the specified class.

User response

Verify that the entity class specified in the entity metadata definition includes the specified field or that the field name is spelled correctly and resubmit the operation.
Related reference:
Deprecated properties and APIs

CWPRJ1102E

MISSING_PROPERTY_CWPRJ1102E=CWPRJ1102E: Property is undefined for class: {0}.

Explanation

The specified property does not exist on the specified class.

User response

Verify that the entity class specified in the entity metadata definition includes the specified property get and set methods or that the property name is spelled correctly and resubmit the operation. The property methods must follow the naming conventions defined in the JavaBeans Introspector API: http://java.sun.com/j2se/1.4.2/docs/api/java/beans/Introspector.html.
Related reference:
Deprecated properties and APIs

CWPRJ1103E

MISSING_ASSOCIATION_TARGET_ENTITY_CWPRJ1103E=CWPRJ1103E: The target entity is undefined.

Explanation

The target entity type could not be determined.

User response

The target entity type of an association must be explicitly set on the association metadata if it cannot be determined by the field or property type.
Related reference:
Deprecated properties and APIs

CWPRJ1104E

DUPLICATE_FIELDPROPERTY_CWPRJ1104E=CWPRJ1104E: Attribute is defined more than once.

Explanation

An attribute or association must have a unique name and cannot be defined more than one time for a single entity.

User response

Review the XML definition for the entity and verify that there is a different name for each element: basic, one-to-one, many-to-one, one-to-many or many-to-many.
Related reference:
Deprecated properties and APIs

CWPRJ1105E

MULTIPLE_INVERSE_ASSOCIATIONS_CWPRJ1105E=CWPRJ1105E: The target association has more than one inverse relationship to this entity and is missing the MappedBy relationship definition. Source entity: {0}, Inverse entity: {1}, Duplicate attributes: {2}

Explanation

A bi-directional relationship between two entities must have one field or property defined with the mappedBy annotation parameter or XML attribute.

User response

Verify that the class or entity metadata definition is valid and resubmit the operation.
Related reference:
Deprecated properties and APIs

CWPRJ1108E

INVALID_INVERSE_ASSOCIATION_TYPE_CWPRJ1108E=CWPRJ1108E: The inverse target association references an invalid entity type. Inverse, target entity: {0}, association name: {1}

Explanation

The inverse side of a bi-directional relationship between two entities has an invalid target entity type defined.

User response

Verify that the class type for both ends of the bi-directional relationship match the respective entity class and resubmit the operation.
Related reference:
Deprecated properties and APIs

CWPRJ1109E

INVALID_ENTITY_DEFINITION_MISSING_TARGET_CWPRJ1109E=CWPRJ1109E: The target entity type of {0} is not defined.

Explanation

An entity has an association to another entity, but the target entity does not exist in the metadata repository.

User response

Verify that the entity association name and type matches a defined entity and resubmit the operation.
Related reference:
Deprecated properties and APIs

CWPRJ1110E

INVALID_INVERSE_KEY_ASSOCIATION_CWPRJ1110E=CWPRJ1110E: An association that is also a key must not have an inverse association. Inverse, target entity: {0}, association name: {1}

Explanation

An entity key may have a one-to-one or many-to-one association to another entity, but the target entity must not have an bi-directional relationship to the same entity.

User response

Remove the association from the target entity and resubmit the operation.
Related reference:
Deprecated properties and APIs

CWPRJ1111E

INVERSE_ASSOCIATION_TYPE_MISMATCH_CWPRJ1111E=CWPRJ1111E: The inverse target association must match the source target type. Inverse, target entity: {0}, association name: {1}

Explanation

The inverse side of a bi-directional relationship between two entities has a target type that does not match the source type.

User response

Verify that the class type for both ends of the bi-directional relationship match the respective entity class and resubmit the operation.
Related reference:
Deprecated properties and APIs

CWPRJ1112E

TARGET_ENTITY_NOT_DEFINED_CWPRJ1112E=CWPRJ1112E: Target entity not defined for field or property: {0}.

Explanation

The target Entity for the association is not defined.

User response

Verify that the target entity is defined.
Related reference:
Deprecated properties and APIs

CWPRJ1113E

ENTITY_ATTRIBUTE_MISSING_CARDINALITY_CWPRJ1113E=CWPRJ1113E: Attribute {0} is an entity type but a relationship to the entity is not defined.

Explanation

The attribute type must declare the relationship cardinality.

User response

Define the cardinality of the entity relationship using the OneToOne or ManyToOne annotation or in the entity descriptor file.
Related reference:
Deprecated properties and APIs

CWPRJ1114E

ENTITY_ATTRIBUTE_NOT_SERIALIZABLE_CWPRJ1114E=CWPRJ1114E: Attribute {0} of type {1} is not serializable.

Explanation

All attribute types must be serializable.

User response

Verify that the attribute implements the java.io.Serializable or java.io.Externalizable interfaces.
Related reference:
Deprecated properties and APIs

CWPRJ1115E

INVALID_MANY_ASSOCIATION_TYPE_CWPRJ1115E=CWPRJ1115E: Invalid OneToMany or ManyToMany association type of: {0}.

Explanation

All multi-valued association types must be of type: java.util.Collection, java.util.List or java.util.Set.

User response

Change the attribute type to one of the supported data types.
Related reference:
Deprecated properties and APIs

CWPRJ1200I

PROJECTOR_INSTRUMENTATION_ENABLED_CWPRJ1200I=CWPRJ1200I: Projector entity class instrumentation is enabled. The instrumentation mode is {0}.

Explanation

Projector entity class instrumentation is enabled. Java classes in the configured transformation domain may be transformed to support field-access entities.

User response

None.
Related reference:
Deprecated properties and APIs

CWPRJ1201E

FIELD_ACCESS_ENTITY_NOT_INSTRUMENTED_CWPRJ1201E=CWPRJ1201E: Field-access entity not instrumented. Entity class={0}.

Explanation

This error occurs when the Projector Java instrumentation agent is enabled but the field-access entity class is not included in instrumentation domain, or the field-access entity class is included in instrumentation domain, but is not in the field-access entity domain. The field-access entity domain is an optional configuration. When specified, all field-access entity classes must included in the field-access entity domain.

User response

Verify that the field-access entity class is included in instrumentation domain if no field-access entity domain is defined. If field-access entity domain is defined, ensure the entity class is included.
Related reference:
Deprecated properties and APIs

CWPRJ1202W

PROXY_UNAVAILABLE_CWPRJ1202W=CWPRJ1202W: Entity proxy support is unavailable.

Explanation

Proxies cannot be created for managed entities because the CGLIB library is not installed or is incompatible. The absence of proxy support may degrade performance.

User response

Review the FFDC logs for details on why the CGLIB was unable to load and verify that a supported CGLIB library version is installed in the classpath.

Related reference:
Deprecated properties and APIs

CWPRJ1300E

MULTIPLE_METHODS_PER_TYPE_CWPRJ1300E=CWPRJ1300E: Multiple methods listen to the same event {0} in class {1}.

Explanation

Only one method can listen to one particular lifecycle event in one entity or entity listener class definition.

User response

Remove extra methods.
Related reference:
Deprecated properties and APIs

CWPRJ1301E

ENTITY_CALLBACK_NO_PARAM_CWPRJ1301E=CWPRJ1301E: Entity lifecycle callback method, {0}, defined in the entity class {1} must have no parameters.

Explanation

The entity lifecycle callback method defined in an entity class should not have parameters.

User response

Redefine the entity lifecycle callback method to remove the parameters.
Related reference:
Deprecated properties and APIs

CWPRJ1302E

LISTENER_CALLBACK_ONE_PARAM_CWPRJ1302E=CWPRJ1302E: Entity lifecycle callback method, {0}, defined in the entity listener class {1} must have only one parameter.

Explanation

The lifecycle callback method defined in an entity listener class should have just one parameter.

User response

Redefine the entity lifecycle callback method in the listener class to have just one parameter.
Related reference:
Deprecated properties and APIs

CWPRJ1303E

LISTENER_UNASSIGNABLE_TYPE_CWPRJ1303E=CWPRJ1303E: The parameter of the entity callback method, {0}, is not assignable to entity {1}.

Explanation

The parameter of a lifecycle callback method defined in an entity listener should be the same type as the entity type or a super type of the entity type.

User response

Redefine the entity lifecycle callback method to make sure the parameter type is the entity type or its super type.
Related reference:
Deprecated properties and APIs

CWPRJ1304E

NO_SUCH_CALLBACK_METHOD_CWPRJ1304E=CWPRJ1304E: The method name {0} with parameter {1} does not exist in the class {2}.

Explanation

The configured callback method name does not exist in the specified entity or entity listener class.

User response

Verify that the entity lifecycle callback or listener method name specified in the entity metadata matches a valid method name in the specified class. If the method is a callback method defined the entity class, verify that the method does not define any arguments. If the class is a listener method, verify that the method includes one argument of type Object or of a specific entity class type.
Related reference:
Deprecated properties and APIs

CWPRJ1305E

UNEXPECTED_CALLBACK_EXCEPTION_CWPRJ1305E=CWPRJ1305E: Unexpected exception encounted when invoking the lifecycle callback method {0}: {1}

Explanation

A runtime exception is caught during the invocation of the lifecycle callback method.

User response

Inspect the exception and the lifecycle callback method to determine the cause of the exception.
Related reference:
Deprecated properties and APIs

CWPRJ5000I

INSTRUMENTATION_ENABLED_CWPRJ5000I=CWPRJ5000I: Java instrumentation mechanism is enabled. The instrumentation mode is {0}.

Explanation

Java instrumentation mechanism is enabled. Java classes in the configured transformation domain may be transformed to support field-access entities.

User response

None.
Related reference:
Deprecated properties and APIs

CWPRJ9000I

ENGLISH_ONLY_INFO_MESSAGE_CWPRJ9000=CWPRJ9000I: This message is an English-only Informational message: {0}.

Explanation

This informational message is not translated.

User response

See message for details.
Related reference:
Deprecated properties and APIs

CWPRJ9001W

ENGLISH_ONLY_WARN_MESSAGE_CWPRJ9001=CWPRJ9001W: This message is an English-only Warning message: {0}.

Explanation

This warning message is not translated.

User response

See message for details.
Related reference:
Deprecated properties and APIs

CWPRJ9002E

ENGLISH_ONLY_ERROR_MESSAGE_CWPRJ9002=CWPRJ9002E: This message is an English only Error message: {0}.

Explanation

This error message is not translated.

User response

See message for details.
Related reference:
Deprecated properties and APIs

CWWSM: WebSphere eXtreme Scale messages for the WebSphere eXtreme Scale HTTP Session manager

When you encounter a message in a log or other parts of the product interface, look up the message by its message ID to find out more information.

- CWWSM0001E
  session.parseXML=CWWSM0001E: An exception occured when parsing the web.xml : {0}
- CWWSM0002E
  session.lineXML=CWWSM0002E: An exception occured at line {0} when parsing the web.xml.
- CWWSM0003E
  session.columnXML=CWWSM0003E: An exception occured at column {0} when parsing the web.xml.
- CWWSM0004E
  session.DGException=CWWSM0004E: Caught a datagrid exception {0} when performing a Datagrid operation.
- CWWSM0005E
  session.throwable=CWWSM0005E: Caught a runtime exception {0}.
- CWWSM0006E
  session.affinityManager=CWWSM0006E: Caught exception when instantiating the affinity manager: {0}.
- CWWSM0007I
  session.objectgrid=CWWSM0007I: Using the ObjectGrid based Session Manager.
- CWWSM0008I
  session.webapp=CWWSM0008I: Web application {0} contains declaration for HttpSessionFilter.
- CWWSM0009I
  session.filter.override=CWWSM0009I: Session Override did not contain any catalog end points. Checking servlet context.
- CWWSM0010I
  session.filter.conn.failed=CWWSM0010I: Connection attempt to ObjectGrid failed: Exception is {0}.
- CWWSM0011W
  session.filter.conn.attribute=CWWSM0011W: An attribute is not able to be serialized. Attribute value is {0}.
- CWWSM0020E
  ERR_MISSING_KEY=CWWSM0020E: Message key {0} was not found in any searched resource bundles.
- CWWSM0021I
  INFO_ARCHIVE_NAME=CWWSM0021I: Reading archive: {0}
- CWWSM0022E
  ERROR_NULL_ARCHIVE_SPECIFIED=CWWSM0022E: Null archive name specifed. Exiting.
- CWWSM0023I
  INFO_PROPERTIES_FILE_NAME=CWWSM0023I: Reading properties file: {0}
- CWWSM0024E
  ERROR_CANNOT_FIND_PROPERTIES_FILE=CWWSM0024E: Unable to locate properties file. Exiting.
- CWWSM0025E
  ERROR_CANNOT_LOAD_PROPERTIES_FILE=CWWSM0025E: Unable to load properties file. Exiting.
- CWWSM0026E
  ERROR_EAR_OR_WAR_FILE_ONLY=CWWSM0026E: Input file must be a .ear or .war file only. Exiting.
- CWWSM0027I
  INFO_PROCESSING_WAR_FILE=CWWSM0027I: Processing .war file: {0}
- CWWSM0028I
  INFO_CONTEXT_PARAMS=CWWSM0028I: Context parameters are:
- CWWSM0029I
  INFO_CONTEXT_NAME=CWWSM0029I: Context name
- CWWSM0030I
  INFO_SPLICING_SUCCESSFUL=CWWSM0030I: Application splicing completed successfully.
- CWWSM0031E
  ERROR_UNABLE_TO_OPEN_FILE=CWWSM0031E: Unable to open file: {0}. Exiting.
- CWWSM0032E
  ERROR_UNABLE_TO_LOAD_FILE=CWWSM0032E: Unable to load file: {0}. Exiting.
- CWWSM0033E
  ERROR_EXCEPTION_CAUGHT=CWWSM0033E: Exception caught when trying to access the archive: {0}

- CWWSM0034E
  ERROR_USAGE=CWWSM0034E: Usage: {0} <location of ear file> <location of properties file>
- CWWSM0034I
  INFO_SERVLET_FOUND=CWWSM0034I: Found servlet: {0}
- CWWSM0035E
  ERROR_USAGE=CWWSM0035E: Usage: {0} <location of ear file> <location of properties file>

Related tasks:
Configuring HTTP session managers

CWWSM0001E

session.parseXML=CWWSM0001E: An exception occured when parsing the web.xml : {0}

Explanation

A parse exception occured while analyzing the web.xml.

User response

Check the integrity of the web.xml as well as its contents for conformance with the web.xml schema.
Related tasks:
Configuring HTTP session managers

CWWSM0002E

session.lineXML=CWWSM0002E: An exception occured at line {0} when parsing the web.xml.

Explanation

A parse exception occured while analyzing the web.xml.

User response

Check the integrity of the web.xml as well as its contents for conformance with the web.xml schema.
Related tasks:
Configuring HTTP session managers

CWWSM0003E

session.columnXML=CWWSM0003E: An exception occured at column {0} when parsing the web.xml.

Explanation

A parse exception occured while analyzing the web.xml.

User response

Check the integrity of the web.xml as well as its contents for conformance with the web.xml schema.
Related tasks:
Configuring HTTP session managers

CWWSM0004E

session.DGException=CWWSM0004E: Caught a datagrid exception {0} when performing a Datagrid operation.

Explanation

A datagrid exception was caught when attempting to perform a datagrid operation.

User response

The exception itself may contain additional details of the problem causing the exception.
Related tasks:
Configuring HTTP session managers

CWWSM0005E

session.throwable=CWWSM0005E: Caught a runtime exception {0}.

Explanation

none.

User response

none.
Related tasks:
Configuring HTTP session managers

CWWSM0006E

session.affinityManager=CWWSM0006E: Caught exception when instantiating the affinity manager: {0}.

Explanation

none.

User response

none.
Related tasks:
Configuring HTTP session managers

CWWSM0007I

session.objectgrid=CWWSM0007I: Using the ObjectGrid based Session Manager.

Explanation

none.

User response

none.
Related tasks:
Configuring HTTP session managers

CWWSM0008I

session.webapp=CWWSM0008I: Web application {0} contains declaration for HttpSessionFilter.

Explanation

The user application was found to have been instrumented with a declaration for a HttpSessionFilter.

User response

none.
Related tasks:
Configuring HTTP session managers

CWWSM0009I

session.filter.override=CWWSM0009I: Session Override did not contain any catalog end points. Checking servlet context.

Explanation

none.

User response

none.
Related tasks:
Configuring HTTP session managers

CWWSM0010I

session.filter.conn.failed=CWWSM0010I: Connection attempt to ObjectGrid failed: Exception is {0}.

Explanation

none.

User response

none.
Related tasks:
Configuring HTTP session managers

CWWSM0011W

session.filter.conn.attribute=CWWSM0011W: An attribute is not able to be serialized. Attribute value is {0}.

Explanation

Attribute data will not be available during a fail-over.

User response

none.
Related tasks:
Configuring HTTP session managers

CWWSM0020E

ERR_MISSING_KEY=CWWSM0020E: Message key {0} was not found in any searched resource bundles.

Explanation

A key was passed into the Messages class to resolve to a string, but a properties file for the locale could not be found and the appropriate message could not be retrieved.

User response

Determine why the message catalog could not be found.
Related tasks:
Configuring HTTP session managers

CWWSM0021I

INFO_ARCHIVE_NAME=CWWSM0021I: Reading archive: {0}

Explanation

Reading specified application archive file for splicing.

User response

No action is required.
Related tasks:
Configuring HTTP session managers

CWWSM0022E

ERROR_NULL_ARCHIVE_SPECIFIED=CWWSM0022E: Null archive name specifed. Exiting.

Explanation

Null archive name specifed. Command exited with failure and application splicing was not performed.

User response

Specify a valid application archive name and try this command.
Related tasks:
Configuring HTTP session managers

CWWSM0023I

INFO_PROPERTIES_FILE_NAME=CWWSM0023I: Reading properties file: {0}

Explanation

Reading specified properties file for splicing.

User response

No action is required.
Related tasks:
Configuring HTTP session managers

CWWSM0024E

ERROR_CANNOT_FIND_PROPERTIES_FILE=CWWSM0024E: Unable to locate properties file. Exiting.

Explanation

Unable to locate the specified properties file. Command exited with failure and application splicing was not performed.

User response

Specify absolute path to the properties file and run the command. Also, ensure that the file specified in path exists.
Related tasks:
Configuring HTTP session managers

CWWSM0025E

ERROR_CANNOT_LOAD_PROPERTIES_FILE=CWWSM0025E: Unable to load properties file. Exiting.

Explanation

Unable to load the specified properties file. Command exited with failure and application splicing was not performed.

User response

Review sample splicer.properties provided and ensure that the property file specified in command line argument has valid content.
Related tasks:
Configuring HTTP session managers

CWWSM0026E

ERROR_EAR_OR_WAR_FILE_ONLY=CWWSM0026E: Input file must be a .ear or .war file only. Exiting.

Explanation

Input file specified must be a .ear or .war file only. Command exited with failure and application splicing was not performed.

User response

Specify absolute path to the .ear file or .war file and run the command. Also, ensure that the file specified in path exists.
Related tasks:

Configuring HTTP session managers

CWWSM0027I

INFO_PROCESSING_WAR_FILE=CWWSM0027I: Processing .war file: {0}

Explanation

Application splicing started for the specified .war file.

User response

No action is required.
Related tasks:
Configuring HTTP session managers

CWWSM0028I

INFO_CONTEXT_PARAMS=CWWSM0028I: Context parameters are:

Explanation

Context parameters for the application splicing.

User response

No action is required.
Related tasks:
Configuring HTTP session managers

CWWSM0029I

INFO_CONTEXT_NAME=CWWSM0029I: Context name

Explanation

Context name for the application that is being spliced.

User response

No action is required.
Related tasks:
Configuring HTTP session managers

CWWSM0030I

INFO_SPLICING_SUCCESSFUL=CWWSM0030I: Application splicing completed successfully.

Explanation

Application splicing completed successfully.

User response

No action is required.
Related tasks:
Configuring HTTP session managers

CWWSM0031E

ERROR_UNABLE_TO_OPEN_FILE=CWWSM0031E: Unable to open file: {0}. Exiting.

Explanation

Unable to open the specified archive file for application splicing. Command exited with failure and application splicing was not performed.

User response

Specify a valid application archive name and try this command. Also, ensure that the archive file is not locked by a different process and you have sufficient permissions to modify.
Related tasks:
Configuring HTTP session managers

CWWSM0032E

ERROR_UNABLE_TO_LOAD_FILE=CWWSM0032E: Unable to load file: {0}. Exiting.

Explanation

Unable to load the application configuration for application splicing.

User response

Specify a valid application archive name and try this command. Also, ensure that application configuration files have valid content.
Related tasks:
Configuring HTTP session managers

CWWSM0033E

ERROR_EXCEPTION_CAUGHT=CWWSM0033E: Exception caught when trying to access the archive: {0}

Explanation

Exception caught when trying to access the specified archive file for application splicing.

User response

Review exception message and take appropriate action.
Related tasks:
Configuring HTTP session managers

CWWSM0034E

ERROR_USAGE=CWWSM0034E: Usage: {0} <location of ear file> <location of properties file>

Explanation

location of ear file and location of properties file are required arguments.

User response

Run this command with proper arguments.
Related tasks:
Configuring HTTP session managers

CWWSM0034I

INFO_SERVLET_FOUND=CWWSM0034I: Found servlet: {0}

Explanation

Servlet com.ibm.ws.httpsession.HttpServletWrapper was found in application.

User response

No action is required.
Related tasks:
Configuring HTTP session managers

CWWSM0035E

ERROR_USAGE=CWWSM0035E: Usage: {0} <location of ear file> <location of properties file>

Explanation

Required parameters for application command are location of .ear or .war file and location of properties file.

User response

Specify the location of .ear or .war file and properties file and run the command.
Related tasks:
Configuring HTTP session managers

CWXQY: WebSphere eXtreme Scale messages for the query engine component

When you encounter a message in a log or other parts of the product interface, look up the message by its message ID to find out more information.

- CWXQY1200E
  FAILTOPARSE=CWXQY1200E: Syntax error. Encountered {0} at line {1}, column {2}.
- CWXQY1201E
  INVAGGINWHERE=CWXQY1201E: Aggregate functions are not allowed in where clause.
- CWXQY1202E
  FAILINITCONFIG=CWXQY1202E: Failed to initialize ObjectMap configuration {0}.
- CWXQY1203E
  UNDEFINEDALIAS=CWXQY1203E: {0} can not be resolved.
- CWXQY1204E
  UNDEFINEDNAME=CWXQY1204E: {0} can not be resolved.
- CWXQY1205E
  UNDEFINEDCONSTR=CWXQY1205E: The constructor {0} is not defined.
- CWXQY1206E
  AMBIGUOUSCONSTR=CWXQY1206E: Ambiguous constructors.
- CWXQY1207E
  BADNAVIGATIONCMP=CWXQY1207E: Can not apply navigation operation operator to {0}
- CWXQY1208E
  BADNAVIGATION=CWXQY1208E: Can not apply navigation operation to {0}
- CWXQY1209E
  BADRETURN=CWXQY1209E: Invalid select clause {0}
- CWXQY1210E
  INTERNAL=CWXQY1210E: Internal Error. {0}
- CWXQY1211E
  RESOLVEERRORS=CWXQY1211E: One or more resolution errors. {0}
- CWXQY1212E
  SUBQRYNUMPROP=CWXQY1212E: Subquery can only return a single property.
- CWXQY1213E
  INVALIDGRPBY=CWXQY1213E: The property type {0} used in the group by clause does not support grouping.
- CWXQY1214E
  INVALIDORDBY=CWXQY1214E: The property type {0} used in order by clause does not support ordering.
- CWXQY1215E
  ARGMUSTBASIC=CWXQY1215E: Operator is not supported on given property type.
- CWXQY1217E
  CANTCOMPARI=CWXQY1217E: Comparison operator not supported on given types.
- CWXQY1220E
  EQNOTSUPPORTED=CWXQY1220E: Equal operator is not supported on given property types.
- CWXQY1221E
  EQONLONG=CWXQY1221E: Equal operator is not supported on long types.
- CWXQY1222E
  CMPENTITYCOLL=CWXQY1222E: Comparison not supported on entity collections.
- CWXQY1223E
  INVALIDCMP=CWXQY1223E: Comparison operator used incorrectly.
- CWXQY1225E
  LIKENONCHAR=CWXQY1225E: Operator LIKE is not supported on given property types.
- CWXQY1227E
  MAXMINBADARG=CWXQY1227E: Function min or max has invalid argument.
- CWXQY1229E
  ARGMUSTBOOLEA=CWXQY1229E: The operand has to be of boolean type.
- CWXQY1230E
  ALLOWONLYONEO=CWXQY1230E: Only one operand is allowed.
- CWXQY1231E
  INVALIDARGTYP=CWXQY1231E: Invalid argument type.

- CWXQY1232E
  ONETOTHREEARG=CWXQY1232E: Function requires at least one argument and no more than three arguments.
- CWXQY1233E
  TWOTOTHREEARG=CWXQY1233E: Function requires at least two arguments and no more than three arguments.
- CWXQY1234E
  TWOARG=CWXQY1234E: Function requires two arguments.
- CWXQY1235E
  ONEARG=CWXQY1235E: Function requires one argument.
- CWXQY1236E
  SUMARGNUMERIC=CWXQY1236E: Function sum argument must be a type of numeric.
- CWXQY1238E
  AVGARGNUMERIC=CWXQY1238E: Function avg argument must be numeric.
- CWXQY1240E
  ASSINGDIFFTYP=CWXQY1240E: Assignment on different types not supported.
- CWXQY1241E
  ASSIGNCONVFAI=CWXQY1241E: Conversion of a numeric type failed on assignment.
- CWXQY1242E
  ARITHNONNUMER=CWXQY1242E: Arithmetic operation not supported on nonnumeric types.
- CWXQY1243E
  UNKNOWNSCALAR=CWXQY1243E: Function {0} is not a supported function.
- CWXQY1244E
  TYPECHECKERRORS=CWXQY1244E: One or more properties used incorrectly. {0}
- CWXQY1246E
  ALIASDUP=CWXQY1246E: Identifier {0} is already defined.
- CWXQY1247E
  OPERATORNOTSUPPORTED=CWXQY1247E: {0} operation is not supported.
- CWXQY1248E
  INVALIDENTITYCOMP=CWXQY1248E: Comparison between entity beans of different types not allowed.
- CWXQY1249E
  REQUIRECOLLECTION=CWXQY1249E: Empty predicate can only be applied to a valued relationship.
- CWXQY1250E
  INVALIDMEMBEROF=CWXQY1250E: Member predicate can not be applied to given property types.
- CWXQY1251E
  UNDEFINEDCONSTT=CWXQY1251E: Internal error. Invalid constant {0}.
- CWXQY1252E
  MISSINGORDERBYTERM=CWXQY1252E: ORDER BY term does not appear in SELECT.
- CWXQY1253E
  MISSINGIDEXOBJ=CWXQY1253E: No index available for ObjectMap {0}.
- CWXQY1254E
  MISSINGALIASECAT=CWXQY1254E: Internal error MISSINGALIASECAT.
- CWXQY1255E
  WRONGTERMFORGP=CWXQY1255E: The field {0} appears in a SELECT or HAVING clause without an aggregate function but is not specified in the GROUP BY clause.
- CWXQY1256E
  NONESTEDAGGFUNC=CWXQY1256E: Nested aggregate functions are not allowed.
- CWXQY1257E
  AGGHASMOREDISTIN=CWXQY1257E: DISTINCT is specified more than once in aggregate functions.
- CWXQY1258E
  INVALIDNEXTSTATE=CWXQY1258E: Internal error. Invalid state on call to next.
- CWXQY1259E
  ARITHMETICOPFAIL=CWXQY1259E: An exception occurred while evaluating the arithmetic expression {0}.
- CWXQY1260E
  ARITHMETICOVERFLOW=CWXQY1260E: Underflow or overflow occurred while evaluating the arithmetic expression {0}.
- CWXQY1261E
  ARITYMETICDIVBYZERO=CWXQY1261E: An Arithmetic exception occurred due to division by zero.
- CWXQY1262E
  NOTFOUNDINMAP=CWXQY1262E: ObjectMap {0} not found.
- CWXQY1263E
  NOTFOUNDINDEX=CWXQY1263E: An [{0}] occurred because the ObjectMap [{1}] does not have index [{2}].
- CWXQY1264E
  NOOBJECTININDEX=CWXQY1264E: An [{0}] occurred because the index [{1}] does not contain the object [{2}].
- CWXQY1265E
  EVALINTERNALERROR=CWXQY1265E: An internal error found in [{0}].
- CWXQY1266E
  INTROSPMETHOD=CWXQY1266E: [{0}] occurred while introspecting method [{1}] of class [{2}].
- CWXQY1267E
  FIELDGETOBJECTFAILED=CWXQY1267E: [{0}] occurred because the specified object [{1}] is not an instance of the class or interface declaring the underlying field [{2}]
- CWXQY1268E
  INVOKMETHODFAIL=CWXQY1268E: [{0}] occurred while invoking method [{1}] on object [{2}].
- CWXQY1269E
  FIELDACCESSFAILED=CWXQY1269E: [{0}] occurred because the field [{1}] is inaccessible.
- CWXQY1270E
  DATEWRONGJDBCESCAPE=CWXQY1270E: Date given [{0}] is not in the JDBC date escape format[yyyy-mm-dd].

- CWXQY1271E
  TIMEWRONGJDBCESCAPE=CWXQY1271E: Time given [{0}] is not in the JDBC time escape format[hh:mm:ss].
- CWXQY1272E
  TIMESTAMPWRONGJDBCESCAPE=CWXQY1272E: Timestamp given [{0}] is not in the JDBC timestamp escape format[yyyy-mm-dd hh:mm:ss.fffffffff].
- CWXQY1282E
  SUBSTRWRONGRANGE=CWXQY1282E: The second or third argument of the SUBSTR function is out of range.
- CWXQY1283E
  NOTNEEDEDPARAMETER=CWXQY1283E: Parameter {0} is not used in the query.
- CWXQY1285E
  OVERFLOWAVG=CWXQY1285E: Counter overflow occurred computing AVG.
- CWXQY1286E
  OVERFLOWCOUNT=CWXQY1286E: Counter overflow occurred computing COUNT.
- CWXQY1287E
  TOOMANYPROJTIONITEMS=CWXQY1287E: Exceeds the maximum number of elements [{0}] allowed in Tuple object fail to add the element [{1}].
- CWXQY1288E
  FEWPARAMETER=CWXQY1288E: The query uses {1} parameters but only {0} were passed.
- CWXQY1289E
  NOPARAMETER=CWXQY1289E: No parameters were passed to a query that required parameters.
- CWXQY1290E
  NOTDEFINEDPARAMETER=CWXQY1290E: Parameter {0} is not defined.
- CWXQY1291E
  INVALIDPARAMETERTYPE=CWXQY1291E: Parameter {0} passed in is a type of {1} which is not the expected type {2}.
- CWXQY1292E
  SCALARSUBQNODATE=CWXQY1292E: Scalar subquery returned no data.
- CWXQY1293E
  SCALARSUBQMORECOL=CWXQY1293E: Scalar subquery returns more than one column.
- CWXQY1294E
  SCALARSUBQMOREROW=CWXQY1294E: Scalar subquery returned more than one row.
- CWXQY1296E
  INVALIDINDEXTYPE=CWXQY1296E: Internal Error. Undefined type [{0}] for index field.
- CWXQY1297E
  ONECHARACTERONLY=CWXQY1297E: Character can only be compared to Character or String of length 1. {0} has more than one character.
- CWXQY1298E
  CONFLICTNAME=CWXQY1298E: Duplicate name {0} in select expressions.
- CWXQY1299E
  INVALIDTOKEN=CWXQY1299E: Invalid token is found in query. {0} in {1}.
- CWXQY1300E
  PARSEERROR=CWXQY1300E: Query parser encountered an error. {0}.
- CWXQY1301E
  INVALIDAS=CWXQY1301E: {0} of an aggregate function is not valid in the context where it it used.
- CWXQY1302E
  INVALIDMEMBEROFMISPK=CWXQY1302E: Member operation failed either because a primary key is not contained in the object or the primary key is not identified in the object's metadata.
- CWXQY1304E
  INVALIDEQMISPK=CWXQY1304E: Equal operation failed either because a primary key is not contained in the object or the primary key is not identified in the object's metadata.
- CWXQY1305E
  INDEXRETRYLIMITEXCEEDED=CWXQY1305E: A [{0}] exception occurred because the index [{1}] exceeded the retry limit for finding the object [{2}].
- CWXQY1306E
  RANGEINDEXRETRYLIMITEXCEEDED=CWXQY1306E: A [{0}] exception occurred because the index [{1}] exceeded the retry limit for running the range query [{2}].
- CWXQY1307E
  NOACTIVETRAN=CWXQY1307E: A [{0}] occurred because there is no active transaction.

Related tasks:
Retrieving entities and objects (Query API)

CWXQY1200E

FAILTOPARSE=CWXQY1200E: Syntax error. Encountered {0} at line {1}, column {2}.

Explanation

Syntax error. The query string may contain syntax error or a numeric literal that is out of range.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1201E

INVAGGINWHERE=CWXQY1201E: Aggregate functions are not allowed in where clause.

Explanation

The query does not satisfy the syntax rule.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1202E

FAILINITCONFIG=CWXQY1202E: Failed to initialize ObjectMap configuration {0}.

Explanation

The ObjectMap is not configured properly.

User response

Correct the ObjectMap configuration.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1203E

UNDEFINEDALIAS=CWXQY1203E: {0} can not be resolved.

Explanation

The identification variable is not defined.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1204E

UNDEFINEDNAME=CWXQY1204E: {0} can not be resolved.

Explanation

The property is not defined.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1205E

UNDEFINEDCONSTR=CWXQY1205E: The constructor {0} is not defined.

Explanation

Undefined constructor {0}.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1206E

AMBIGUOUSCONSTR=CWXQY1206E: Ambiguous constructors.

Explanation

Ambiguous constructors.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1207E

BADNAVIGATIONCMP=CWXQY1207E: Can not apply navigation operation operator to {0}

Explanation

A navigation operation can not be applied to a multi valued relationship or property.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1208E

BADNAVIGATION=CWXQY1208E: Can not apply navigation operation to {0}

Explanation

A navigation operation can not be applied to a multi valued relationship or property.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1209E

BADRETURN=CWXQY1209E: Invalid select clause {0}

Explanation

SELECT clause cannot project a multi valued relationship.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1210E

INTERNAL=CWXQY1210E: Internal Error. {0}

Explanation

Internal Error.

User response

Report this problem to IBM service.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1211E

RESOLVEERRORS=CWXQY1211E: One or more resolution errors. {0}

Explanation

One or more resolution errors.

User response

Read details followed by this message.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1212E

SUBQRYNUMPROP=CWXQY1212E: Subquery can only return a single property.

Explanation

A subquery contains multiple elements in the select clause.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1213E

INVALIDGRPBY=CWXQY1213E: The property type {0} used in the group by clause does not support grouping.

Explanation

The property type used in the group by clause does not support grouping.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1214E

INVALIDORDBY=CWXQY1214E: The property type {0} used in order by clause does not support ordering.

Explanation

The property type used in order by clause does not support ordering.

User response

Correct the query statement.
Related tasks:

Retrieving entities and objects (Query API)

CWXQY1215E

ARGMUSTBASIC=CWXQY1215E: Operator is not supported on given property type.

Explanation

Operator is not supported on given property type.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1217E

CANTCOMPARI=CWXQY1217E: Comparison operator not supported on given types.

Explanation

Comparison operator not supported on given types.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1220E

EQNOTSUPPORTED=CWXQY1220E: Equal operator is not supported on given property types.

Explanation

Equal operator is not supported on given property types.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1221E

EQONLONG=CWXQY1221E: Equal operator is not supported on long types.

Explanation

Equal operator is not supported on long types.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1222E

CMPENTITYCOLL=CWXQY1222E: Comparison not supported on entity collections.

Explanation

Comparison not supported on entity collections.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1223E

INVALIDCMP=CWXQY1223E: Comparison operator used incorrectly.

Explanation

Comparison operator used incorrectly.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1225E

LIKENONCHAR=CWXQY1225E: Operator LIKE is not supported on given property types.

Explanation

Operator LIKE is not supported on non character types.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1227E

MAXMINBADARG=CWXQY1227E: Function min or max has invalid argument.

Explanation

Function min or max has invalid argument.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1229E

ARGMUSTBOOLEA=CWXQY1229E: The operand has to be of boolean type.

Explanation

The operand has to be of boolean type.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1230E

ALLOWONLYONEO=CWXQY1230E: Only one operand is allowed.

Explanation

Only one operand is allowed.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1231E

INVALIDARGTYP=CWXQY1231E: Invalid argument type.

Explanation

Invalid argument type.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1232E

ONETOTHREEARG=CWXQY1232E: Function requires at least one argument and no more than three arguments.

Explanation

Function requires at least one argument and no more than three arguments.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1233E

TWOTOTHREEARG=CWXQY1233E: Function requires at least two arguments and no more than three arguments.

Explanation

Function requires at least two arguments and no more than three arguments.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1234E

TWOARG=CWXQY1234E: Function requires two arguments.

Explanation

Function requires two arguments.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1235E

ONEARG=CWXQY1235E: Function requires one argument.

Explanation

Function requires one argument.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1236E

SUMARGNUMERIC=CWXQY1236E: Function sum argument must be a type of numeric.

Explanation

Function sum argument must be a type of numeric.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1238E

AVGARGNUMERIC=CWXQY1238E: Function avg argument must be numeric.

Explanation

Function avg argument must be numeric.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1240E

ASSINGDIFFTYP=CWXQY1240E: Assignment on different types not supported.

Explanation

Assignment on different types not supported.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1241E

ASSIGNCONVFAI=CWXQY1241E: Conversion of a numeric type failed on assignment.

Explanation

Conversion of a numeric type failed on assignment.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1242E

ARITHNONNUMER=CWXQY1242E: Arithmetic operation not supported on nonnumeric types.

Explanation

Arithmetic operation not supported on nonnumeric types.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1243E

UNKNOWNSCALAR=CWXQY1243E: Function {0} is not a supported function.

Explanation

Function is not supported in EJB query language.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1244E

TYPECHECKERRORS=CWXQY1244E: One or more properties used incorrectly. {0}

Explanation

One or more properties used incorrectly.

User response

Read details followed by this message.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1246E

ALIASDUP=CWXQY1246E: Identifier {0} is already defined.

Explanation

Identifier is already defined.

User response

Correct the query statement.
Related tasks:

Retrieving entities and objects (Query API)

CWXQY1247E

OPERATORNOTSUPPORTED=CWXQY1247E: {0} operation is not supported.

Explanation

operation is not supported.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1248E

INVALIDENTITYCOMP=CWXQY1248E: Comparison between entity beans of different types not allowed.

Explanation

Comparison between entity beans of different types not allowed.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1249E

REQUIRECOLLECTION=CWXQY1249E: Empty predicate can only be applied to a valued relationship.

Explanation

Empty predicate can only be applied to a valued relationship.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1250E

INVALIDMEMBEROF=CWXQY1250E: Member predicate can not be applied to given property types.

Explanation

The Member predicate can only compare an entity bean to a collection of beans of a compatible type.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1251E

UNDEFINEDCONSTT=CWXQY1251E: Internal error. Invalid constant {0}.

Explanation

Internal error. Invalid constant {0}.

User response

Report this problem to IBM service.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1252E

MISSINGORDERBYTERM=CWXQY1252E: ORDER BY term does not appear in SELECT.

Explanation

If the ORDER BY clause is used, the element being ordered by must appear in the SELECT clause.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1253E

MISSINGIDEXOBJ=CWXQY1253E: No index available for ObjectMap {0}.

Explanation

At least one index should be defined for each ObjectMap.

User response

Correct the metadata.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1254E

MISSINGALIASECAT=CWXQY1254E: Internal error MISSINGALIASECAT.

Explanation

Internal error.

User response

Report this problem to IBM service.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1255E

WRONGTERMFORGP=CWXQY1255E: The field {0} appears in a SELECT or HAVING clause without an aggregate function but is not specified in the GROUP BY clause.

Explanation

When performing a grouping operation, elements of the SELECT clause must either be aggregation functions or be grouping elements. The field indicated by token {0} is used in the SELECT clause and does not appear in an aggregation function but is not a grouping element.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1256E

NONESTEDAGGFUNC=CWXQY1256E: Nested aggregate functions are not allowed.

Explanation

An aggregate function such as SUM cannot contain another aggregate function in the argument expression.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1257E

AGGHASMOREDISTIN=CWXQY1257E: DISTINCT is specified more than once in aggregate functions.

Explanation

You cannot use DISTINCT more than once in aggregate functions in a query.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1258E

INVALIDNEXTSTATE=CWXQY1258E: Internal error. Invalid state on call to next.

Explanation

Internal error.

User response

Report this problem to IBM.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1259E

ARITHMETICOPFAIL=CWXQY1259E: An exception occurred while evaluating the arithmetic expression {0}.

Explanation

An exception occurred while evaluating the given expression.

User response

Correct the arithmetic expression.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1260E

ARITHMETICOVERFLOW=CWXQY1260E: Underflow or overflow occurred while evaluating the arithmetic expression {0}.

Explanation

Underflow or overflow occurred while evaluating the arithmetic expression {0}.

User response

Correct the arithmetic expression.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1261E

ARITYMETICDIVBYZERO=CWXQY1261E: An Arithmetic exception occurred due to division by zero.

Explanation

An Arithmetic exception occurred due to division by zero.

User response

Correct the arithmetic expression.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1262E

NOTFOUNDINMAP=CWXQY1262E: ObjectMap {0} not found.

Explanation

ObjectMap {0} is either not defined or it is defined but it is not found in this server.

User response

Correct the query statement and make sure all ObjectMap referenced in the query statement are defined and can be found in the same server.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1263E

NOTFOUNDINDEX=CWXQY1263E: An [{0}] occurred because the ObjectMap [{1}] does not have index [{2}].

Explanation

The ObjectGrid Entity Manager catalog is out of sync with the actual ObjectMap configuration.

User response

Check the ObjectMap configuration.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1264E

NOOBJECTININDEX=CWXQY1264E: An [{0}] occurred because the index [{1}] does not contain the object [{2}].

Explanation

An object {2} returned from the ObjectMap using index {1} is not the expected type.

User response

Check the class definition and the ObjectMap and index configuration.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1265E

EVALINTERNALERROR=CWXQY1265E: An internal error found in [{0}].

Explanation

An internal error found in [{0}].

User response

Report this problem to IBM service.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1266E

INTROSPMETHOD=CWXQY1266E: [{0}] occurred while introspecting method [{1}] of class [{2}].

Explanation

Methods should be defined as public, must not be void and have no arguments.

User response

Correct the class definition. Methods should be defined as public and must not be void and have no arguments.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1267E

FIELDGETOBJECTFAILED=CWXQY1267E: [{0}] occurred because the specified object [{1}] is not an instance of the class or interface declaring the underlying field [{2}]

Explanation

An error detected while attempting to retrieve a given field.

User response

correct the given class definition or the field name
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1268E

INVOKMETHODFAIL=CWXQY1268E: [{0}] occurred while invoking method [{1}] on object [{2}].

Explanation

The invoked method threw an exception.

User response

correct the invoked method in the application model.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1269E

FIELDACCESSFAILED=CWXQY1269E: [{0}] occurred because the field [{1}] is inaccessible.

Explanation

Cannot access the field {1}

User response

Correct the application model.

Related tasks:
Retrieving entities and objects (Query API)

CWXQY1270E

DATEWRONGJDBCESCAPE=CWXQY1270E: Date given [{0}] is not in the JDBC date escape format[yyyy-mm-dd].

Explanation

The correct literal constant for date is yyyy-mm-dd.

User response

Correct the literal constant {0}
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1271E

TIMEWRONGJDBCESCAPE=CWXQY1271E: Time given [{0}] is not in the JDBC time escape format[hh:mm:ss].

Explanation

The correct literal constant for time is hh:mm:ss.

User response

Correct the literal constant {0}
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1272E

TIMESTAMPWRONGJDBCESCAPE=CWXQY1272E: Timestamp given [{0}] is not in the JDBC timestamp escape format[yyyy-mm-dd hh:mm:ss.fffffffff].

Explanation

The correct literal constant for timestamp is yyyy-mm-dd hh:mm:ss.fffffffff.

User response

Correct the literal constant {0}
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1282E

SUBSTRWRONGRANGE=CWXQY1282E: The second or third argument of the SUBSTR function is out of range.

Explanation

The sum of the second and third arguments is greater than the length of the first argument.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1283E

NOTNEEDEDPARAMETER=CWXQY1283E: Parameter {0} is not used in the query.

Explanation

The parameter indicated by token {0} is not used in the query.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1285E

OVERFLOWAVG=CWXQY1285E: Counter overflow occurred computing AVG.

Explanation

Overflow occurred while computing AVG aggregate function.

User response

Correct the query statement to avoid the overflow.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1286E

OVERFLOWCOUNT=CWXQY1286E: Counter overflow occurred computing COUNT.

Explanation

Overflow occurred while computing COUNT aggregate function.

User response

Correct the query statement to avoid the overflow.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1287E

TOOMANYPROJTIONITEMS=CWXQY1287E: Exceeds the maximum number of elements [{0}] allowed in Tuple object fail to add the element [{1}].

Explanation

A maximum of {0} identification variables can be used in the query plan which is generated by the query statement.

User response

Reduce the complexity of the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1288E

FEWPARAMETER=CWXQY1288E: The query uses {1} parameters but only {0} were passed.

Explanation

The number of parameters passed in the query engine is less than expected.

User response

Correct the number of parameter passed in.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1289E

NOPARAMETER=CWXQY1289E: No parameters were passed to a query that required parameters.

Explanation

No parameters were passed to a query that required parameters.

User response

Correct the parameter passed in.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1290E

NOTDEFINEDPARAMETER=CWXQY1290E: Parameter {0} is not defined.

Explanation

The parameter indicated by token {0} is not defined.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1291E

INVALIDPARAMETERTYPE=CWXQY1291E: Parameter {0} passed in is a type of {1} which is not the expected type {2}.

Explanation

Parameter {0} passed in is a type of {1} which is not the expected type {2}.

User response

Correct the parameter passed in.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1292E

SCALARSUBQNODATE=CWXQY1292E: Scalar subquery returned no data.

Explanation

A subquery used with a basic predicate must return a single value. No values were returned at runtime when the subquery was evaluated.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1293E

SCALARSUBQMORECOL=CWXQY1293E: Scalar subquery returns more than one column.

Explanation

The SELECT clause of a subquery is invalid because it specifies multiple columns.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1294E

SCALARSUBQMOREROW=CWXQY1294E: Scalar subquery returned more than one row.

Explanation

A subquery used with a basic predicate must only return a single value. Multiple values were returned at runtime when the subquery was evaluated.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1296E

INVALIDINDEXTYPE=CWXQY1296E: Internal Error. Undefined type [{0}] for index field.

Explanation

Internal error.

User response

Report this problem to IBM service.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1297E

ONECHARACTERONLY=CWXQY1297E: Character can only be compared to Character or String of length 1. {0} has more than one character.

Explanation

Character can only be compared to Character or String of length 1.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1298E

CONFLICTNAME=CWXQY1298E: Duplicate name {0} in select expressions.

Explanation

There is name conflict in selection list, use alias to make sure the output column names are unique.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1299E

INVALIDTOKEN=CWXQY1299E: Invalid token is found in query. {0} in {1}.

Explanation

The query statement contains an invalid token.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1300E

PARSEERROR=CWXQY1300E: Query parser encountered an error. {0}.

Explanation

Internal error.

User response

Report this problem to IBM service.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1301E

INVALIDAS=CWXQY1301E: {0} of an aggregate function is not valid in the context where it it used.

Explanation

The select alias of an aggregate function is not valid in group by clause.

User response

Correct the query statement.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1302E

INVALIDMEMBEROFMISPK=CWXQY1302E: Member operation failed either because a primary key is not contained in the object or the primary key is not identified in the object's metadata.

Explanation

The Member operation requires the object to have a primary key and also identify the primary key.

User response

Correct the object definition.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1304E

INVALIDEQMISPK=CWXQY1304E: Equal operation failed either because a primary key is not contained in the object or the primary key is not identified in the object's metadata.

Explanation

The Equal operation requires the object to have a primary key and also identify the primary key.

User response

Correct the object definition.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1305E

INDEXRETRYLIMITEXCEEDED=CWXQY1305E: A [{0}] exception occurred because the index [{1}] exceeded the retry limit for finding the object [{2}].

Explanation

Too many update operations are running against the ObjectMap while the index [{1}] is trying to find the object [{2}].

User response

Reduce the number of update operations while using index to find objects.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1306E

RANGEINDEXRETRYLIMITEXCEEDED=CWXQY1306E: A [{0}] exception occurred because the index [{1}] exceeded the retry limit for running the range query [{2}].

Explanation

Too many update operations are running against the ObjectMap while the index [{1}] is trying to run the range query [{2}].

User response

Reduce the number of update operations while using index to run the range query.
Related tasks:
Retrieving entities and objects (Query API)

CWXQY1307E

NOACTIVETRAN=CWXQY1307E: A [{0}] occurred because there is no active transaction.

Explanation

A transaction was not started before running a query.

User response

Begin a transaction before calling the query.
Related tasks:
Retrieving entities and objects (Query API)

CWXSA: WebSphere eXtreme Scale messages for the extension point component

When you encounter a message in a log or other parts of the product interface, look up the message by its message ID to find out more information.

- CWXSA0501E
  Error.ExecutePointProvider.name=CWXSA0501E: Execute point provider not added to extension point. Execute point provider name {0} already exists.
- CWXSA0502E
  Error.ExecutePointProvider.name.missing=CWXSA0502E: Execute point provider {0} was not found in the {1} extension point.

CWXSA0501E

Error.ExecutePointProvider.name=CWXSA0501E: Execute point provider not added to extension point. Execute point provider name {0} already exists.

Explanation

An execute point name attempted to register with an extension point but was already registered.

User response

No action is required. If developing a new execute point, rename your execute point to have unique name.

CWXSA0502E

Error.ExecutePointProvider.name.missing=CWXSA0502E: Execute point provider {0} was not found in the {1} extension point.

Explanation

An attempt to was made to retrieve an execute point from an extension point and it was not available.

User response

No action is required. If developing a new execute point, check that the execute point is registered.

CWXSB: : WebSphere eXtreme Scale messages for the XsByteBuffer component

When you encounter a message in a log or other parts of the product interface, look up the message by its message ID to find out more information.

- CWXSB0861E
  NOT_VALID_CUSTOM_PROPERTY=CWXSB0861E: The custom property {0} has an value of {1}. This value is not valid.
- CWXSB0862W
  UNRECOGNIZED_CUSTOM_PROPERTY=CWXSB0862W: The custom property {0} specified for the XsByteBuffer Component is not valid.
- CWXSB0864E
  POOL_MISMATCH=CWXSB0864E: The XsByteBuffer Pool Sizes and Pool Depths specification do not have the same number of entries, Sizes: {0} Depths: {1}
- CWXSB0865E
  ALREADY_RELEASED=CWXSB0865E: Attempted to access XsByteBuffer that was already released. ID={0} ByteBuffer={1}

Related reference:
Trace options

CWXSB0861E

NOT_VALID_CUSTOM_PROPERTY=CWXSB0861E: The custom property {0} has an value of {1}. This value is not valid.

Explanation

A custom property was specified with a value that is outside the range of valid values.

User response

Correct the configuration error.
Related reference:
Trace options

CWXSB0862W

UNRECOGNIZED_CUSTOM_PROPERTY=CWXSB0862W: The custom property {0} specified for the XsByteBuffer Component is not valid.

Explanation

A custom property was specified incorrectly.

User response

Correct the configuration error.
Related reference:
Trace options

CWXSB0864E

POOL_MISMATCH=CWXSB0864E: The XsByteBuffer Pool Sizes and Pool Depths specification do not have the same number of entries, Sizes: {0} Depths: {1}

Explanation

The Pool Sizes or Pool Depths were specified incorrectly.

User response

Correct the configuration error.
Related reference:
Trace options

CWXSB0865E

ALREADY_RELEASED=CWXSB0865E: Attempted to access XsByteBuffer that was already released. ID={0} ByteBuffer={1}

Explanation

An internal error occurred attempting to access XsByteBuffer that was already released.

User response

Gather XsByteBuffer=all=enabled trace for IBM support to determine the root cause.
Related reference:
Trace options

CWXSI: WebSphere eXtreme Scale messages for the xscmd component

When you encounter a message in a log or other parts of the product interface, look up the message by its message ID to find out more information.

- CWXSI0001E
  UNRECOGNIZED_OPTION_EXCEPTION_CWXSI0001=CWXSI0001E: Unrecognized option: {0}.
- CWXSI0002E
  ALREADY_SELECTED_EXCEPTION_CWXSI0002=CWXSI0002E: The {0} option was specified, but an option from this group was already selected: {1}.
- CWXSI0003E
  MISSING_ARGUMENT_EXCEPTION_CWXSI0003=CWXSI0003E: Missing argument for option: {0}.
- CWXSI0004E
  MISSING_REQUIRED_OPTION_EXCEPTION_CWXSI0004=CWXSI0004E: Missing required options: {0}.
- CWXSI0005E
  PLACEMENT_XML_NULL_ERROR_CWXSI0005=CWXSI0005E: The placement XML file was null. The task cannot continue.
- CWXSI0006E
  NO_SERVER_ADDRESSES_DETECTED_ERROR_CWXSI0006=CWXSI0006E: No server addresses were detected. Stopping the task.
- CWXSI0007E
  PLACEMENT_MISSING_CONTAINER_ERROR_CWXSI0007=CWXSI0007E: The placement service requires that container servers are running, but only a catalog server was detected. Stopping the task.
- CWXSI0008E
  JMX_QUERY_TIMEOUT_ERROR_CWXSI0008=CWXSI0008E: Command results cannot be displayed because the Java Management Extensions (JMX) query threads timed out.
- CWXSI0009E
  UNRECOGNIZED_COMMAND_EXCEPTION_CWXSI0009=CWXSI0009E: Cannot run the command {0} because the command is not defined in the xscmd tool.
- CWXSI0011E
  GENERIC_CLI_EXCEPTION_CWXSI0011=CWXSI0011E: A general exception occurred while processing the {0} command. Exception: {1}
- CWXSI0012E
  PARSE_EXCEPTION_CWXSI0012=CWXSI0012E: An exception occurred while parsing the {0} command. Exception: {1}
- CWXSI0013E
  COREGROUP_XML_NULL_ERROR_CWXSI0013=CWXSI0013E: The task cannot run because the core group XML file was null.
- CWXSI0014E
  PLACEMENT_STATUS_RETRIEVE_ERROR_CWXSI0014=CWXSI0014E: The PlacementServiceMBean MBean did not return an ObjectGrid placement status.
- CWXSI0015E
  SHARD_MVMT_SUMMARY_RETRIEVE_ERROR_CWXSI0015=CWXSI0015E: A shard movement summary cannot display because the JMXContainer URL was not found.
- CWXSI0016E
  JMX_CONTAINER_NOT_FOUND_ERROR_CWXSI0016=CWXSI0016E: A shard movement summary cannot display because the JMXContainer URL was not found.
- CWXSI0017E
  SWAPPING_SHARD_W_PRIMARY_ERROR_CWXSI0017=CWXSI0017E: Swapping shard {0} with the primary shard failed with the following exception: {1}
- CWXSI0018E
  JMX_CONN_CLOSE_ERROR_CWXSI0018=CWXSI0018E: The Java Management Extensions (JMX) connection cannot close.

- CWXSI0019E
  SUSPEND_BAL_ERROR_CWXSI0019=CWXSI0019E: The placement service attempt to suspend shard balancing did not complete successfully.
- CWXSI0020E
  RESUME_BAL_ERROR_CWXSI0020=CWXSI0020E: The placement service attempt to resume shard balancing did not complete successfully.
- CWXSI0021E
  CONTAINER_SVC_MBEAN_ERROR_CWXSI0021=CWXSI0021E: Cannot connect to the container server because container service MBeans are not available.
- CWXSI0022E
  CLI_GRID_MS_NOT_FOUND_ERRROR_CWXSI0022=CWXSI0022E: The specified data grid name {0} and map set name {1} were not found.
- CWXSI0023E
  CLI_MS_NOT_FOUND_ERRROR_CWXSI0023=CWXSI0023E: The specified map set name {0} was not found.
- CWXSI0024E
  CLI_GRID_NOT_FOUND_ERRROR_CWXSI0024=CWXSI0024E: The specified data grid name {0} was not found.
- CWXSI0025E
  CLI_SHOW_QUORUM_SERVER_RETRIEVE_ERRROR_CWXSI0025=CWXSI0025E: Server names cannot be retrieved due to a connection issue.
- CWXSI0026W
  CLI_TEARDOWN_SERVER_NON_MATCH_CWXSI0026=CWXSI0026W: The arguments to the specified parameters did not match any known servers.
- CWXSI0027W
  CLI_ROUTETABLE_CONN_FAILED_CWXSI0027=CWXSI0027W: Could not connect to the catalog server at {0}:{1}.
- CWXSI0028I
  CLI_ROUTETABLE_RETRY_CONN_CWXSI0028=CWXSI0028I: Setting the catalog server host: {0} and port: {1}.
- CWXSI0029E
  CLI_MEDIATION_SERVICE_CONN_ERROR_CWXSI0029=CWXSI0029E: Mediation service MBean not available.
- CWXSI0030E
  CLI_DOMAIN_LINK_NA_ERROR_CWXSI0030=CWXSI0030E: This catalog service domain does not support multi-data center linking. The catalog service domain must be at Version 7.1 or later.
- CWXSI0031W
  CLI_SET_SPEC_ERROR_CWXSI0031=CWXSI0031W: Cannot set trace or statistic specification due to the following exception: {0}
- CWXSI0032W
  CLI_GET_TRACE_SPEC_ERROR_CWXSI0032=CWXSI0032W: Cannot retrieve the {2} for server {0} due to the following exception: {1}
- CWXSI0033W
  CLI_GET_TRACE_SPEC_UNKNOWN_ERROR_CWXSI0033=CWXSI0033W: Cannot retrieve the trace specification for server {0} because the server is running software before Version 7.1.
- CWXSI0034E
  PLACEMENT_XML_GRID_NULL_ERROR_CWXSI0034=CWXSI0034E: The task cannot run because the placement XML file for the data grid {0} was null.
- CWXSI0035W
  CLI_SERVER_MBEAN_MISMATCH_WARNING_CWXSI0035=CWXSI0035W: The queried Server MBean server name {0} did not match the server that was specified as an option: {1}
- CWXSI0036W
  CLI_SERVER_MBEAN_NULL_WARNING_CWXSI0036=CWXSI0036W: The Server MBean returned a null name.
- CWXSI0037I
  CLI_CLEARGRID_CONN_RETRY_CWXSI0037=CWXSI0037I: Retrying the connection to the catalog server host with -jh {0} and port with -lp {1}.
- CWXSI0038I
  CLI_CLEARGRID_VERBOSE_MSG_CWXSI0038=CWXSI0038I: For more information about the command you are running, use the verbose option, -v.
- CWXSI0039E
  UNRECOGNIZED_COMMAND_GROUP_EXCEPTION_CWXSI0039=CWXSI0039E: The command group {0} is not defined in the xscmd tool.
- CWXSI0040I
  COMMAND_SUCCESSFUL_MSG_CWXSI0040=CWXSI0040I: The {0} command completed successfully.
- CWXSI0041E
  UNRECOGNIZED_SF_ARGUMENT_ERROR_CWXSI0041=CWXSI0041E: Unexpected argument for -sf option.
- CWXSI0042E
  CATALOG_SERVER_CONN_ERROR_CWXSI0042=CWXSI0042E: Cannot connect to the catalog server at: {0}
- CWXSI0043E
  OSGI_UPDATE_ERROR_CWXSI0043=CWXSI0043E: OSGi update operation failed for the following reason: {0}
- CWXSI0044E
  SERVER_MBEAN_CONN_ERROR_CWXSI0044=CWXSI0044E: Exception occurred while connecting to JMX server connection at JMX URL: {0}
- CWXSI0045E
  OSGI_SR_SERVICE_MISSING_ERROR_CWXSI0045=CWXSI0045E: The service ranking list part ''{0}'' does not contain a service value.
- CWXSI0046E
  OSGI_SR_SERVICE_RANK_MISSING_ERROR_CWXSI0046=CWXSI0046E: The service ranking list part ''{0}'' does not contain service ranking value.
- CWXSI0047E
  OSGI_SR_SERVICE_REPEATED_ERROR_CWXSI0047=CWXSI0047E: The service ''{0}'' is repeated in the service ranking list: ''{1}''
- CWXSI0048E
  OSGI_SR_NOT_INTEGER_ERROR_CWXSI0048=CWXSI0048E: The service ranking ''{0}'' of OSGi service ''{1}'' is not an integer.
- CWXSI0049E
  PLACEMENT_XML_NULL_OGMS_ERROR_CWXSI0049=CWXSI0049E: The placement XML for ObjectGrid ''{0}'' and map set name ''{1}'' was null. The task cannot continue.
- CWXSI0050W
  OSGI_RETRIEVE_SERIVCE_RANKING_MSG_CWXSI0050=CWXSI0050W: An exception occurred when getting current service rankings from OSGi service name {0}, and service URL: {1}, with exception: {2}. Stack trace: {3}
- CWXSI0051I
  OSGI_CONTINUE_MSG_CWXSI0051=CWXSI0051I: Ignoring the exception from this container server, and continuing processing with the next container server.
- CWXSI0052W
  OSGI_RETRIEVE_ALL_SERIVCE_RANKING_MSG_CWXSI0052=CWXSI0052W: Exception occurred when getting service rankings from OSGi services at

service URL: {0}, with exception: {1}. Stack trace: {2}
- CWXSI0053W
  OSGI_CANNOT_FIND_SERVER_MSG_CWXSI0053=CWXSI0053W: Unable to find server referred to by service URL: {0}
- CWXSI0054W
  OSGI_SERVER_RETRIEVE_SERIVCE_RANKING_MSG_CWXSI0054=CWXSI0054W: Exception occurred when getting service rankings from server {0}, and service URL: {1}, with exception: {2}. Stack trace: {3}
- CWXSI0055W
  OSGI_SERVER_RETRIEVE_SERIVCE_RANKING_ERROR_LIST_CWXSI0055=CWXSI0055W: The following servers failed to retrieve OSGi service rankings, with the following exception messages:
- CWXSI0056E
  OSGI_CURRENT_ERROR_MSG_CWXSI0056=CWXSI0056E: The following errors have been discovered during processing.
- CWXSI0057E
  OSGI_SERVER_RETRIEVE_MBEAN_ERROR_CWXSI0057=CWXSI0057E: Error occurred when getting the Java Management Extensions (JMX) OSGi MBean from server {0} with service URL {1}, with exception {2}. Stack trace: {3}
- CWXSI0058I
  OSGI_SERVER_RETRIEVE_MBEAN_CONTINUE_CWXSI0058=CWXSI0058I: Continue processing with next available MBean.
- CWXSI0059E
  JMX_CONNECTION_ERROR_CWXSI0059=CWXSI0059E: An error might have occurred when attempting to connect to the JMX connector server with service URL: {0}. Exception: {1}.
- CWXSI0060W
  OSGI_CONTINUE_MSG_CWXSI0060=CWXSI0060W: Could not obtain server MBean for this container server. Continue with processing of results for the next server.
- CWXSI0061I
  CATALOG_SVR_CONN_MSG_CWXSI0061=CWXSI0061I: Connecting to catalog service at {0}:{1}
- CWXSI0062E
  CATALOG_SVR_UNAVAILABLE_CWXSI0062=CWXSI0062E: Catalog service is unavailable at the {0} endpoint.
- CWXSI0063W
  CLI_CLEARGRID_MAP_NONMATCH_CWXSI0063=CWXSI0063W: No maps were found.
- CWXSI0064W
  CLI_SSL_REQ_ON_CONT_WITHOUT_PORT_CWXSI0064=CWXSI0064W: Ran a command that contains SSL options on an environment that includes a server that was not started with the JMXServicePort property specified.
- CWXSI0065E
  REPEATED_OPTION_EXCEPTION_CWXSI0065=CWXSI0065E: Repeated option {0} was detected
- CWXSI0066E
  UNMATCHED_ARGUMENT_EXCEPTION_CWXSI0066=CWXSI0066E: Unmatched argument {0} was detected
- CWXSI0067E
  BAL_STATUS_RETRIEVE_ERROR_CWXSI0067=CWXSI0067E: The PlacementServiceMBean MBean did not return balance status results.
- CWXSI0068I
  EXECUTING_CMD_INFO_CWXSI0068=CWXSI0068I: Executing command: {0}
- CWXSI0069W
  MAPSET_FOR_GRID_MAP_NOT_FOUND_CWXSI0069=CWXSI0069W: The map set name for the specified data grid name {0} and map name {1} was not found. Execution continues.
- CWXSI0070I
  CLI_ROUTETABLE_LPORT_MISSING_CWXSI0070=CWXSI0070I: The -lp option was not specified as a command-line option. Using {0} as the bootstrap port value.
- CWXSI0071I
  COMMAND_LINK_FINISHED_MSG_CWXSI0071I=CWXSI0071I: The link operation was submitted. Use {0} command to verify the results of the link operation.
- CWXSI0072E
  SWAP_SHARD_NOT_FOUND_CWXSI0072=CWXSI0072E: A shard that matches the specified objectGrid name, map set name or partition number was not found. Verify that the arguments are correct and the {0} objectGrid is available. An objectGrid name of {0}, a map set name of {1} and a partition number of {2} were provided.
- CWXSI0073W
  CLI_GET_SPEC_ERROR_CWXSI0073=CWXSI0073W: Exception occurred when getting specification for service URL: {0}, with exception: {1}. Stack trace: {2}
- CWXSI0074W
  CLI_SET_SPEC_ERROR_CWXSI0074=CWXSI0074W: Exception occurred when setting specification for service URL: {0}, with exception: {1}. Stack trace: {2}
- CWXSI0075E
  BAL_SHARD_TYPE_ERROR_CWXSI0075=CWXSI0075E: PlacementServiceMBean did not return results from attempt to balance shard types
- CWXSI0076E
  CLI_SPEC_INVALID_ARG_ERROR_CWXSI0076=CWXSI0076E: The specification string "{0}" is not valid
- CWXSI0077E
  CLI_REMOVE_PROFILE_ERROR_CWXSI0077=CWXSI0077E: Error removing profile {0}.
- CWXSI0078E
  BAL_SHARD_TYPE_XML_ERROR_CWXSI0078=CWXSI0078E: PlacementServiceMBean returned invalid XML
- CWXSI0079E
  ERROR_ARG_FILE_EXISTS_CWXSI0079=CWXSI0079E: The argument for parameter {0}: {1} does not exist.
- CWXSI0080E
  RELEASE_SHARD_ERROR_CWXSI0080=CWXSI0080E: Releasing shard {0} failed with the following exception: {1}
- CWXSI0081E
  RESERVE_SHARD_ERROR_CWXSI0081=CWXSI0081E: Reserving shard {0} failed with the following exception: {1}
- CWXSI0082E
  CLI_QUORUM_ERROR_CWXSI0082=CWXSI0082E: Command {0} failed. Quorum is enabled and catalog server is waiting for quorum. Check your

environment to determine whether it is undergoing a brown out or black out. If you detect a brown out, try the command again at a later time. If you detect a black out, consider overriding quorum.

- CWXSI0083E
  CLI_MAP_NOT_FOUND_ERRROR_CWXSI0083=CWXSI0083E: The specified map name {0} was not found.
- CWXSI0084E
  CLI_CG_NOT_FOUND_ERRROR_CWXSI0084=CWXSI0084E: The specified core group name {0} was not found.
- CWXSI0085W
  CLI_METHOD_NOT_AVAILABLE_CWXSI0085=CWXSI0085W: The specified command {0} was not available on server {1}. The specified command requires WebSphere eXtreme Scale version {2} or later.
- CWXSI0086W
  CLI_NO_RESULTS_RETURNED_CWXSI0086=CWXSI0086W: The specified command {0} did not return any results for {1}.

Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0001E

UNRECOGNIZED_OPTION_EXCEPTION_CWXSI0001=CWXSI0001E: Unrecognized option: {0}.

Explanation

Each command has a specific set of available options.

User response

See the help for the attempted command to see the available options.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0002E

ALREADY_SELECTED_EXCEPTION_CWXSI0002=CWXSI0002E: The {0} option was specified, but an option from this group was already selected: {1}.

Explanation

You can run one option at a time from a mutually-exclusive group of options.

User response

Examine the options that you entered on the command line and remove the extra option.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0003E

MISSING_ARGUMENT_EXCEPTION_CWXSI0003=CWXSI0003E: Missing argument for option: {0}.

Explanation

The specified option requires an argument.

User response

See the help text for the attempted command to determine the missing argument for the specified option.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0004E

MISSING_REQUIRED_OPTION_EXCEPTION_CWXSI0004=CWXSI0004E: Missing required options: {0}.

Explanation

One or more required options are missing from the command line.

User response

See the help text to determine what options are missing from the specified command.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0005E

PLACEMENT_XML_NULL_ERROR_CWXSI0005=CWXSI0005E: The placement XML file was null. The task cannot continue.

Explanation

The runtime environment cannot find the placement XML file for the specified object grid or map set name.

User response

Examine your objectgrid.xml file to verify that an ObjectGrid and map set are specified.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0006E

NO_SERVER_ADDRESSES_DETECTED_ERROR_CWXSI0006=CWXSI0006E: No server addresses were detected. Stopping the task.

Explanation

The runtime environment cannot retrieve addresses to servers that are registered with the placement service.

User response

Examine your xscmd command-line options to ensure that you specified the correct Java Management Extensions (JMX) host name and JMX port number.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0007E

PLACEMENT_MISSING_CONTAINER_ERROR_CWXSI0007=CWXSI0007E: The placement service requires that container servers are running, but only a catalog server was detected. Stopping the task.

Explanation

When you run the xscmd command, a container server must be started.

User response

Verify that your environment includes a running container server. For more information about starting container servers, see the information center.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0008E

JMX_QUERY_TIMEOUT_ERROR_CWXSI0008=CWXSI0008E: Command results cannot be displayed because the Java Management Extensions (JMX) query threads timed out.

Explanation

The xscmd command timed out while waiting to establish a JMX connection.

User response

Verify that any xscmd command-line options include the appropriate JMX host name and JMX port number, and that the MBean server is running.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0009E

UNRECOGNIZED_COMMAND_EXCEPTION_CWXSI0009=CWXSI0009E: Cannot run the command {0} because the command is not defined in the xscmd tool.

Explanation

The command being invoked is not registered with the xscmd tool.

User response

Verify that any xscmd command name being specified is spelled correctly. Use the -cg and -l options to see a list of all valid command groups and commands.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0011E

GENERIC_CLI_EXCEPTION_CWXSI0011=CWXSI0011E: A general exception occurred while processing the {0} command. Exception: {1}

Explanation

The xscmd tool encountered an unanticipated exception.

User response

Contact IBM Software Support for further investigation.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0012E

PARSE_EXCEPTION_CWXSI0012=CWXSI0012E: An exception occurred while parsing the {0} command. Exception: {1}

Explanation

The xscmd tool encountered an unanticipated parsing exception.

User response

Contact IBM Software Support for further investigation.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0013E

COREGROUP_XML_NULL_ERROR_CWXSI0013=CWXSI0013E: The task cannot run because the core group XML file was null.

Explanation

The runtime environment cannot find the core group XML content for the specified command.

User response

Verify that the catalog server is running and accessible and that the catalog service host name and port numbers are correct.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0014E

PLACEMENT_STATUS_RETRIEVE_ERROR_CWXSI0014=CWXSI0014E: The PlacementServiceMBean MBean did not return an ObjectGrid placement status.

Explanation

The runtime placement service cannot retrieve status as this time.

User response

Verify that any xscmd command-line options include the appropriate JMX host name and JMX port number values (or respective default values are set), and that the MBean server is running.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0015E

SHARD_MVMT_SUMMARY_RETRIEVE_ERROR_CWXSI0015=CWXSI0015E: A shard movement summary cannot display because the JMXContainer URL was not found.

Explanation

The runtime placement service cannot retrieve a shard movement summary.

User response

Examine the options you entered on the command line to ensure that a valid data grid name and map set name have been specified.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0016E

JMX_CONTAINER_NOT_FOUND_ERROR_CWXSI0016=CWXSI0016E: A shard movement summary cannot display because the JMXContainer URL was not found.

Explanation

The runtime placement service cannot retrieve a shard movement summary.

User response

Examine the options you entered on the command line to ensure that a valid data grid name and map set name have been specified.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0017E

SWAPPING_SHARD_W_PRIMARY_ERROR_CWXSI0017=CWXSI0017E: Swapping shard {0} with the primary shard failed with the following exception: {1}

Explanation

The runtime environment is unable to swap the replica shard with the specified primary shard.

User response

Examine the options you entered on the command line to ensure that valid arguments have been specified for the command.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0018E

JMX_CONN_CLOSE_ERROR_CWXSI0018=CWXSI0018E: The Java Management Extensions (JMX) connection cannot close.

Explanation

The runtime environment cannot close the Java Management Extensions (JMX) connection.

User response

Verify that the catalog server is running and accessible and that the catalog service host name and port numbers are correct.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0019E

SUSPEND_BAL_ERROR_CWXSI0019=CWXSI0019E: The placement service attempt to suspend shard balancing did not complete successfully.

Explanation

The PlacementServiceMBean MBean did not return results from an attempt to suspend balancing.

User response

Examine command-line options to ensure that valid Java Management Extensions (JMX) host and port values are specified.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0020E

RESUME_BAL_ERROR_CWXSI0020=CWXSI0020E: The placement service attempt to resume shard balancing did not complete successfully.

Explanation

The PlacementServiceMBean MBean did not return results from an attempt to resume balancing.

User response

Examine the command-line options to ensure that valid Java Management Extensions (JMX) host and port values are specified.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0021E

CONTAINER_SVC_MBEAN_ERROR_CWXSI0021=CWXSI0021E: Cannot connect to the container server because container service MBeans are not available.

Explanation

The Java Management Extensions (JMX) MBean server could not connect to the container service.

User response

Examine your xscmd command-line options to ensure that valid JMX host and port values are specified.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0022E

CLI_GRID_MS_NOT_FOUND_ERRROR_CWXSI0022=CWXSI0022E: The specified data grid name {0} and map set name {1} were not found.

Explanation

The runtime environment could not locate the data grid and map set that were specified on the command line.

User response

Use the xscmd listObjectGridNames command to list available data grids and map sets.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0023E

CLI_MS_NOT_FOUND_ERRROR_CWXSI0023=CWXSI0023E: The specified map set name {0} was not found.

Explanation

The runtime environment could not locate the map set that was specified on the command line.

User response

Use the xscmd listObjectGridNames command to list available data grids and map sets.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0024E

CLI_GRID_NOT_FOUND_ERRROR_CWXSI0024=CWXSI0024E: The specified data grid name {0} was not found.

Explanation

The runtime environment could not locate the data grid specified on the command line.

User response

Use the xscmd listObjectGridNames command to list available data grids and map sets.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0025E

CLI_SHOW_QUORUM_SERVER_RETRIEVE_ERRROR_CWXSI0025=CWXSI0025E: Server names cannot be retrieved due to a connection issue.

Explanation

The runtime environment could not detect any active servers.

User response

Examine your xscmd command-line options to ensure that valid Java Management Extensions (JMX) host and port values are specified.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0026W

CLI_TEARDOWN_SERVER_NON_MATCH_CWXSI0026=CWXSI0026W: The arguments to the specified parameters did not match any known servers.

Explanation

The runtime environment could not detect the specified servers.

User response

Confirm that the specified servers are active before running the command.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0027W

CLI_ROUTETABLE_CONN_FAILED_CWXSI0027=CWXSI0027W: Could not connect to the catalog server at {0}:{1}.

Explanation

The runtime environment could not retrieve route table information using the specific host name and bootstrap port.

User response

No action is required. Retrying the connection using the listener port.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0028I

CLI_ROUTETABLE_RETRY_CONN_CWXSI0028=CWXSI0028I: Setting the catalog server host: {0} and port: {1}.

Explanation

The runtime environment is attempting to connect to the catalog server using the specified listener port.

User response

No action is required.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0029E

CLI_MEDIATION_SERVICE_CONN_ERROR_CWXSI0029=CWXSI0029E: Mediation service MBean not available.

Explanation

The runtime environment could not connect to the Mediation service management bean.

User response

Contact IBM Software Support for further investigation.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0030E

CLI_DOMAIN_LINK_NA_ERROR_CWXSI0030=CWXSI0030E: This catalog service domain does not support multi-data center linking. The catalog service domain must be at Version 7.1 or later.

Explanation

The attempted operation is only applicable to catalog service domains that are at Version 7.1 or later.

User response

Upgrade your catalog service domain to Version 7.1. For more information about updating eXtreme Scale servers, see the information center.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0031W

CLI_SET_SPEC_ERROR_CWXSI0031=CWXSI0031W: Cannot set trace or statistic specification due to the following exception: {0}

Explanation

The runtime environment was unable to set the provided trace or statistic specification.

User response

Verify that you are using a valid trace or statistic specification.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0032W

CLI_GET_TRACE_SPEC_ERROR_CWXSI0032=CWXSI0032W: Cannot retrieve the {2} for server {0} due to the following exception: {1}

Explanation

The runtime environment could not retrieve the requested information for the specified server.

User response

Verify that the server is running.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0033W

CLI_GET_TRACE_SPEC_UNKNOWN_ERROR_CWXSI0033=CWXSI0033W: Cannot retrieve the trace specification for server {0} because the server is running software before Version 7.1.

Explanation

Retrieving the trace specification is only applicable to catalog service domains that are at Version 7.1 or later.

User response

Upgrade your servers to Version 7.1. For more information about updating eXtreme Scale servers, see the information center.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0034E

PLACEMENT_XML_GRID_NULL_ERROR_CWXSI0034=CWXSI0034E: The task cannot run because the placement XML file for the data grid {0} was null.

Explanation

The runtime environment cannot find a Placement XML file for the specified data grid.

User response

Examine your objectgrid.xml file to ensure that an ObjectGrid instance is specified.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0035W

CLI_SERVER_MBEAN_MISMATCH_WARNING_CWXSI0035=CWXSI0035W: The queried Server MBean server name {0} did not match the server that was specified as an option: {1}

Explanation

The name of the server that is associated with the queried MBean is not the server that was specified on the command line.

User response

Verify that the JMX host name and port are correct and can be reached.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0036W

CLI_SERVER_MBEAN_NULL_WARNING_CWXSI0036=CWXSI0036W: The Server MBean returned a null name.

Explanation

The name of the server associated with the MBean is null.

User response

Verify that the specified JMX host name and port are correct and can be reached.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0037I

CLI_CLEARGRID_CONN_RETRY_CWXSI0037=CWXSI0037I: Retrying the connection to the catalog server host with -jh {0} and port with -lp {1}.

Explanation

The runtime environment could not connect to the catalog server with the specified Java Management Extensions (JMX) host name and JMX port.

User response

No action is required. The runtime environment is attempting to connect again using the configured Object Request Broker (ORB) listener port.

Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0038I

CLI_CLEARGRID_VERBOSE_MSG_CWXSI0038=CWXSI0038I: For more information about the command you are running, use the verbose option, -v.

Explanation

Use the verbose option to see more information about the command you are running.

User response

No action is required.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0039E

UNRECOGNIZED_COMMAND_GROUP_EXCEPTION_CWXSI0039=CWXSI0039E: The command group {0} is not defined in the xscmd tool.

Explanation

The xscmd command group that is being queried is not registered with the xscmd tool.

User response

Verify that any xscmd command group name being specified is spelled correctly. Use the -cg option see a list of all valid command groups.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0040I

COMMAND_SUCCESSFUL_MSG_CWXSI0040=CWXSI0040I: The {0} command completed successfully.

Explanation

The command that was run completed successfully.

User response

No action is required.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0041E

UNRECOGNIZED_SF_ARGUMENT_ERROR_CWXSI0041=CWXSI0041E: Unexpected argument for -sf option.

Explanation

The argument to the xscmd option -sf was not recognized.

User response

Consult the usage summary to verify the valid arguments of this option.
Related tasks:
Administering with the xscmd utility

Querying and invalidating data
Troubleshooting administration

CWXSI0042E

CATALOG_SERVER_CONN_ERROR_CWXSI0042=CWXSI0042E: Cannot connect to the catalog server at: {0}

Explanation

The runtime environment could not connect to the container service.

User response

Examine your xscmd command-line options to ensure that valid JMX host and port values are specified.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0043E

OSGI_UPDATE_ERROR_CWXSI0043=CWXSI0043E: OSGi update operation failed for the following reason: {0}

Explanation

The runtime environment could not update the OSGi service.

User response

Examine your xscmd command-line options to ensure that the ObjectGrid and map set specified have the corresponding OSGi service.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0044E

SERVER_MBEAN_CONN_ERROR_CWXSI0044=CWXSI0044E: Exception occurred while connecting to JMX server connection at JMX URL: {0}

Explanation

The runtime environment could not obtain connection to the MBean server

User response

Examine your xscmd command-line options to ensure that the specified host name and port are correct and can be reached.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0045E

OSGI_SR_SERVICE_MISSING_ERROR_CWXSI0045=CWXSI0045E: The service ranking list part ''{0}'' does not contain a service value.

Explanation

The runtime environment could not interpret the provided OSGi service ranking.

User response

Examine the syntax of your command to ensure that you provided a correct OSGi service value.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0046E

OSGI_SR_SERVICE_RANK_MISSING_ERROR_CWXSI0046=CWXSI0046E: The service ranking list part ''{0}'' does not contain service ranking value.

Explanation

The runtime environment could not interpret the provided OSGi service ranking.

User response

Examine the syntax of your command to ensure that you provided a correct OSGi service ranking value.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0047E

OSGI_SR_SERVICE_REPEATED_ERROR_CWXSI0047=CWXSI0047E: The service ''{0}'' is repeated in the service ranking list: ''{1}''

Explanation

The runtime environment detected a duplicate OSGi service specified in the command line

User response

Correct your xscmd command-line argument to remove repeated instances of OSGi services
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0048E

OSGI_SR_NOT_INTEGER_ERROR_CWXSI0048=CWXSI0048E: The service ranking ''{0}'' of OSGi service ''{1}'' is not an integer.

Explanation

The runtime environment expects an integer a value to be provided.

User response

Correct your xscmd command-line argument to use an integer as the OSGi service ranking.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0049E

PLACEMENT_XML_NULL_OGMS_ERROR_CWXSI0049=CWXSI0049E: The placement XML for ObjectGrid ''{0}'' and map set name ''{1}'' was null. The task cannot continue.

Explanation

The runtime environment cannot find placement XML file for the specified object grid or map set name.

User response

Specify a different object grid and map set pair on the command line.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0050W

OSGI_RETRIEVE_SERIVCE_RANKING_MSG_CWXSI0050=CWXSI0050W: An exception occurred when getting current service rankings from OSGi service name {0}, and service URL: {1}, with exception: {2}. Stack trace: {3}

Explanation

The runtime environment was unable to obtain service rankings from the container server specified by the URL.

User response

Examine the content of the exception stack. Then verify that the server referred to by the URL is running and network accessible before running the command again.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0051I

OSGI_CONTINUE_MSG_CWXSI0051=CWXSI0051I: Ignoring the exception from this container server, and continuing processing with the next container server.

Explanation

The runtime environment was unable to connect to a container server to obtain OSGi service rankings. Attempting to connect to another container server.

User response

None required. This is an informational message.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0052W

OSGI_RETRIEVE_ALL_SERIVCE_RANKING_MSG_CWXSI0052=CWXSI0052W: Exception occurred when getting service rankings from OSGi services at service URL: {0}, with exception: {1}. Stack trace: {2}

Explanation

The runtime environment was unable to obtain service rankings from the OSGi sevices in the container server specified by the URL

User response

Examine the content of the exception stack. Then make sure that the server referred to by the URL is running and network accessible before running the command again.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0053W

OSGI_CANNOT_FIND_SERVER_MSG_CWXSI0053=CWXSI0053W: Unable to find server referred to by service URL: {0}

Explanation

The runtime environment was unable to locate the container server that was specified by the URL.

User response

Verify that the server referred to by the URL is running and network is accessible before running the command again.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data

CWXSI0054W

OSGI_SERVER_RETRIEVE_SERIVCE_RANKING_MSG_CWXSI0054=CWXSI0054W: Exception occurred when getting service rankings from server {0}, and service URL: {1}, with exception: {2}. Stack trace: {3}

Explanation

The runtime environment was unable to obtain service rankings from the container server specified

User response

Examine the content of the exception stack. Then make sure that the server referred to by the URL is running and network accessible before running the command again.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0055W

OSGI_SERVER_RETRIEVE_SERIVCE_RANKING_ERROR_LIST_CWXSI0055=CWXSI0055W: The following servers failed to retrieve OSGi service rankings, with the following exception messages:

Explanation

The runtime environment was unable to retrieve service rankings from the specified servers due to the exceptions listed

User response

Ensure that the servers specified have OSGi services set before running this command again.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0056E

OSGI_CURRENT_ERROR_MSG_CWXSI0056=CWXSI0056E: The following errors have been discovered during processing.

Explanation

The runtime environment encountered error during processing.

User response

None.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0057E

OSGI_SERVER_RETRIEVE_MBEAN_ERROR_CWXSI0057=CWXSI0057E: Error occurred when getting the Java Management Extensions (JMX) OSGi MBean from server {0} with service URL {1}, with exception {2}. Stack trace: {3}

Explanation

The runtime environment was unable to obtain a JMX OSGi-related MBean for the specified server.

User response

Ensure that the specified server is running and the network is accessible. Examine command line options to ensure valid host name and port values.
Related tasks:

Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0058I

OSGI_SERVER_RETRIEVE_MBEAN_CONTINUE_CWXSI0058=CWXSI0058I: Continue processing with next available MBean.

Explanation

The runtime environment continues its processing by obtaining the next available MBean.

User response

No action is required.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0059E

JMX_CONNECTION_ERROR_CWXSI0059=CWXSI0059E: An error might have occurred when attempting to connect to the JMX connector server with service URL: {0}. Exception: {1}.

Explanation

The runtime environment failed to connect to JMX connector server.

User response

Ensure that server corresponding to the URL is running and network accessible. Examine connection-related command line options to ensure valid host name and port values
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0060W

OSGI_CONTINUE_MSG_CWXSI0060=CWXSI0060W: Could not obtain server MBean for this container server. Continue with processing of results for the next server.

Explanation

The runtime environment was unable to obtain the JMX connection and JMX MBean corresponding to the provided service URL

User response

Examine command-line connection options to ensure correct host name and port values.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0061I

CATALOG_SVR_CONN_MSG_CWXSI0061=CWXSI0061I: Connecting to catalog service at {0}:{1}

Explanation

The runtime environment is attempting to connect to a catalog service located at the specified host:port

User response

No action is required.

Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0062E

CATALOG_SVR_UNAVAILABLE_CWXSI0062=CWXSI0062E: Catalog service is unavailable at the {0} endpoint.

Explanation

The runtime environment was unable to connect to a catalog service at the specified endpoint.

User response

Ensure catalog service is running and accessible on the specified endpoint. If the catalog service is within a catalog service domain, re-run this command by selecting another catalog service endpoint in the catalog service domain.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0063W

CLI_CLEARGRID_MAP_NONMATCH_CWXSI0063=CWXSI0063W: No maps were found.

Explanation

The runtime environment was unable to clear the specified grids since they did not contain any maps, and no dynamic maps were specified to clear.

User response

Ensure that the grids you are trying to clear contain maps. If attempting to clear dynamic maps, specify them on the command line.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0064W

CLI_SSL_REQ_ON_CONT_WITHOUT_PORT_CWXSI0064=CWXSI0064W: Ran a command that contains SSL options on an environment that includes a server that was not started with the JMXServicePort property specified.

Explanation

The command might not complete correctly. Because this command is an SSL request, the JMXServicePort property must be specified during the startup of the container server.

User response

Either restart all of the container servers with the JMXServicePort property specified or remove the SSL-related options from this command.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0065E

REPEATED_OPTION_EXCEPTION_CWXSI0065=CWXSI0065E: Repeated option {0} was detected

Explanation

A command-line options was specified more than one time.

User response

Remove the repeated option from the command line, and run the command again.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0066E

UNMATCHED_ARGUMENT_EXCEPTION_CWXSI0066=CWXSI0066E: Unmatched argument {0} was detected

Explanation

A command-line argument was not matched with any specified command-line options

User response

Remove the unmatched argument from the command line, and run the command again.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0067E

BAL_STATUS_RETRIEVE_ERROR_CWXSI0067=CWXSI0067E: The PlacementServiceMBean MBean did not return balance status results.

Explanation

The runtime placement service was unable to retrieve the balance status at this time.

User response

Verify that any xscmd command-line options include the appropriate JMX host name and JMX port number values (or respective default values are set), a valid Grid and Map set name was provided, and that the MBean server is running.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0068I

EXECUTING_CMD_INFO_CWXSI0068=CWXSI0068I: Executing command: {0}

Explanation

The runtime environment is running the specified command.

User response

None. Informational message only.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0069W

MAPSET_FOR_GRID_MAP_NOT_FOUND_CWXSI0069=CWXSI0069W: The map set name for the specified data grid name {0} and map name {1} was not found. Execution continues.

Explanation

The Placement Service could not locate the map set corresponding to the grid name and map name specified. Some rows of data might be missing from the output of the command as a result.

User response

Upgrade your catalog service domain to Version 7.1.1. For more information about upgrading eXtreme Scale servers, see the information center.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0070I

CLI_ROUTETABLE_LPORT_MISSING_CWXSI0070=CWXSI0070I: The -lp option was not specified as a command-line option. Using {0} as the bootstrap port value.

Explanation

If you do not specify a bootstrap port for the running command, the default is used. Defaulting to port {0}.

User response

If the command does not complete successfully, use the -lp option to specify a bootstrap port.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0071I

COMMAND_LINK_FINISHED_MSG_CWXSI0071I=CWXSI0071I: The link operation was submitted. Use {0} command to verify the results of the link operation.

Explanation

The link operation was submitted successfully. The result of the link operation is unknown at this time. Run the suggested command to determine the result.

User response

Run the suggested command.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0072E

SWAP_SHARD_NOT_FOUND_CWXSI0072=CWXSI0072E: A shard that matches the specified objectGrid name, map set name or partition number was not found. Verify that the arguments are correct and the {0} objectGrid is available. An objectGrid name of {0}, a map set name of {1} and a partition number of {2} were provided.

Explanation

A shard matching the specific arguments was not found on the catalog service. The specified operation cannot complete without a valid shard.

User response

Review the objectGrid name, the map set name and the partition number. If the arguments are correct, use the showObjectGridPlacement command to verify placement and that the requested shard is on a container.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0073W

CLI_GET_SPEC_ERROR_CWXSI0073=CWXSI0073W: Exception occurred when getting specification for service URL: {0}, with exception: {1}. Stack trace: {2}

Explanation

The runtime environment was unable to get the specification from the service URL

User response

Examine the content of the exception stack. Then make sure that the server referred to by the URL is running and network accessible before running the command again.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0074W

CLI_SET_SPEC_ERROR_CWXSI0074=CWXSI0074W: Exception occurred when setting specification for service URL: {0}, with exception: {1}. Stack trace: {2}

Explanation

The runtime environment was unable to set the specification for the service URL

User response

Examine the content of the exception stack. Then make sure that the server referred to by the URL is running and network accessible before running the command again.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0075E

BAL_SHARD_TYPE_ERROR_CWXSI0075=CWXSI0075E: PlacementServiceMBean did not return results from attempt to balance shard types

Explanation

The runtime placement service was unable to balance shard types at this time

User response

Verify network connectivity with your catalog server and that the MBean server is running.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0076E

CLI_SPEC_INVALID_ARG_ERROR_CWXSI0076=CWXSI0076E: The specification string "{0}" is not valid

Explanation

The format of specification string was not valid

User response

Verify the format of the specification string is valid
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0077E

CLI_REMOVE_PROFILE_ERROR_CWXSI0077=CWXSI0077E: Error removing profile {0}.

Explanation

The xscmd tool was unable to remove the profile.

User response

Verify the profile exists.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0078E

BAL_SHARD_TYPE_XML_ERROR_CWXSI0078=CWXSI0078E: PlacementServiceMBean returned invalid XML

Explanation

The XML returned from balance shard type was invalid

User response

Verify that you are connected to a server running 7.1.1 or greater
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0079E

ERROR_ARG_FILE_EXISTS_CWXSI0079=CWXSI0079E: The argument for parameter {0}: {1} does not exist.

Explanation

The specified file path does not exist.

User response

Verify the specified file path exists.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0080E

RELEASE_SHARD_ERROR_CWXSI0080=CWXSI0080E: Releasing shard {0} failed with the following exception: {1}

Explanation

The runtime environment is unable to release the shard with the specified partition ID.

User response

Examine the options you entered on the command line to ensure that a valid partition ID was specified.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0081E

RESERVE_SHARD_ERROR_CWXSI0081=CWXSI0081E: Reserving shard {0} failed with the following exception: {1}

Explanation

The runtime environment is unable to reserve the shard with the specified partition ID.

User response

Examine the options you entered on the command line to ensure that a valid, unreserved partition ID was specified.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0082E

CLI_QUORUM_ERROR_CWXSI0082=CWXSI0082E: Command {0} failed. Quorum is enabled and catalog server is waiting for quorum. Check your environment to determine whether it is undergoing a brown out or black out. If you detect a brown out, try the command again at a later time. If you detect a black out, consider overriding quorum.

Explanation

During quorum loss execution of commands is not predictable.

User response

Override quorum and attempt the command again.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0083E

CLI_MAP_NOT_FOUND_ERRROR_CWXSI0083=CWXSI0083E: The specified map name {0} was not found.

Explanation

The runtime environment could not locate the map that was specified on the command line.

User response

Specify a map name that exists.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0084E

CLI_CG_NOT_FOUND_ERRROR_CWXSI0084=CWXSI0084E: The specified core group name {0} was not found.

Explanation

The runtime environment could not locate the core group that was specified on the command line.

User response

Use the xscmd listCoreGroups command to list available core groups.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0085W

CLI_METHOD_NOT_AVAILABLE_CWXSI0085=CWXSI0085W: The specified command {0} was not available on server {1}. The specified command requires WebSphere eXtreme Scale version {2} or later.

Explanation

The command was not found on the remote server. It is an older version than the xscmd version.

User response

Run the command again after all servers are upgraded to the new version, otherwise no action is required.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSI0086W

CLI_NO_RESULTS_RETURNED_CWXSI0086=CWXSI0086W: The specified command {0} did not return any results for {1}.

Explanation

The expected result was either empty or null after running the expected command.

User response

Review the command and options supplied. Review the catalog and container server side JVM logs and FFDC logs for an exception during the server side command execution.
Related tasks:
Administering with the xscmd utility
Querying and invalidating data
Troubleshooting administration

CWXSR: WebSphere eXtreme Scale messages for the log analyzer component

When you encounter a message in a log or other parts of the product interface, look up the message by its message ID to find out more information.

- CWXSR0001I
  AGGREGATION_BEGIN_CWXSR0001I=CWXSR0001I: All log files processed, starting global summary analysis
- CWXSR0002E
  ROOT_DIRECTORY_NOT_FOUND_CWXSR0002E=CWXSR0002E: Root directory not found
- CWXSR0003I
  PROCESS_FILE_CWXSR0003I=CWXSR0003I: Found log file to process {0}
- CWXSR0004I
  START_PROCESS_REPLAY_CWXSR0004I=CWXSR0004I: Start analysis of log files for process {0}
- CWXSR0005I
  END_PROCESS_REPLAY_CWXSR0005I=CWXSR0005I: Finished analysis of log files for process {0}
- CWXSR0006I
  START_FILE_REPLAY_CWXSR0006I=CWXSR0006I: Start analysis of log file {0}
- CWXSR0007I
  END_FILE_REPLAY_CWXSC0007I=CWXSR0007I: End analysis of log file {0}
- CWXSR0008I
  AGGREGATION_END_CWXSR0008I=CWXSR0008I: Finished global summary analysis
- CWXSR0009E
  ERROR_UNZIPPING_CWXSR0009E=CWXSR0009E: The report was not completely generated because archive {0} was not extracted.
- CWXSR0010W
  ERROR_UNZIPPING_CWXSR0010W=CWXSR0010W: The output stream was not closed successfully.
- CWXSR0011I
  INFO_REPORT_LOCATION_CWXSR0011I=CWXSR0011I: The report generated successfully and can be found in the following path: {0}.
- CWXSR0012E
  ERROR_REPORT_LOCATION_CWXSR0012E=CWXSR0012E: The report generation failed.
- CWXSR0013W
  WARNING_REPORT_LOCATION_CWXSR0013W=CWXSR0013W: The report generated with errors and can be found in the following path: {0}.
- CWXSR0500I
  WRITING_REPORT_CWXSR0500I=CWXSR0500I: Writing out report: {0}

Related tasks:
Analyzing log and trace data
Troubleshooting log analysis

CWXSR0001I

AGGREGATION_BEGIN_CWXSR0001I=CWXSR0001I: All log files processed, starting global summary analysis

Explanation

This message is for informational purposes only.

User response

No action is required.
Related tasks:
Analyzing log and trace data
Troubleshooting log analysis

CWXSR0002E

ROOT_DIRECTORY_NOT_FOUND_CWXSR0002E=CWXSR0002E: Root directory not found

Explanation

Directory specified by the logsRoot parameter could not be found.

User response

Check to make sure the directory exists and try the command again.
Related tasks:
Analyzing log and trace data
Troubleshooting log analysis

CWXSR0003I

PROCESS_FILE_CWXSR0003I=CWXSR0003I: Found log file to process {0}

Explanation

This message is for informational purposes only.

User response

No action is required.
Related tasks:
Analyzing log and trace data
Troubleshooting log analysis

CWXSR0004I

START_PROCESS_REPLAY_CWXSR0004I=CWXSR0004I: Start analysis of log files for process {0}

Explanation

This message is for informational purposes only.

User response

No action is required.
Related tasks:
Analyzing log and trace data
Troubleshooting log analysis

CWXSR0005I

END_PROCESS_REPLAY_CWXSR0005I=CWXSR0005I: Finished analysis of log files for process {0}

Explanation

This message is for informational purposes only.

User response

No action is required.
Related tasks:
Analyzing log and trace data
Troubleshooting log analysis

CWXSR0006I

START_FILE_REPLAY_CWXSR0006I=CWXSR0006I: Start analysis of log file {0}

Explanation

This message is for informational purposes only.

User response

No action is required.
Related tasks:
Analyzing log and trace data
Troubleshooting log analysis

CWXSR0007I

END_FILE_REPLAY_CWXSC0007I=CWXSR0007I: End analysis of log file {0}

Explanation

This message is for informational purposes only.

User response

No action is required.
Related tasks:
Analyzing log and trace data
Troubleshooting log analysis

CWXSR0008I

AGGREGATION_END_CWXSR0008I=CWXSR0008I: Finished global summary analysis

Explanation

This message is for informational purposes only.

User response

No action is required.
Related tasks:
Analyzing log and trace data
Troubleshooting log analysis

CWXSR0009E

ERROR_UNZIPPING_CWXSR0009E=CWXSR0009E: The report was not completely generated because archive {0} was not extracted.

Explanation

This archive contains static content. Not having this content will cause issues viewing the report.

User response

Check to make sure the path to this archive exists. If the path exists then contact IBM Software Support for further investigation.
Related tasks:
Analyzing log and trace data
Troubleshooting log analysis

CWXSR0010W

ERROR_UNZIPPING_CWXSR0010W=CWXSR0010W: The output stream was not closed successfully.

Explanation

After extracting an archive, an issue was encountered by closing the output stream.

User response

No action is required.
Related tasks:
Analyzing log and trace data
Troubleshooting log analysis

CWXSR0011I

INFO_REPORT_LOCATION_CWXSR0011I=CWXSR0011I: The report generated successfully and can be found in the following path: {0}.

Explanation

This message is for informational purposes only.

User response

No action is required.
Related tasks:
Analyzing log and trace data
Troubleshooting log analysis

CWXSR0012E

ERROR_REPORT_LOCATION_CWXSR0012E=CWXSR0012E: The report generation failed.

Explanation

The report could not be created.

User response

Look for other messages to see why it failed. If no other messages are found or help is needed with the messages that are found contact IBM Software Support for further investigation.
Related tasks:
Analyzing log and trace data
Troubleshooting log analysis

CWXSR0013W

WARNING_REPORT_LOCATION_CWXSR0013W=CWXSR0013W: The report generated with errors and can be found in the following path: {0}.

Explanation

The report generated successfully but encountered some non critical errors.

User response

Look for other messages to see what non critical errors occurred. If help is needed with the messages that are found contact IBM Software Support for further investigation.
Related tasks:
Analyzing log and trace data
Troubleshooting log analysis

CWXSR0500I

WRITING_REPORT_CWXSR0500I=CWXSR0500I: Writing out report: {0}

Explanation

This message is for informational purposes only.

User response

No action is required.
Related tasks:
Analyzing log and trace data
Troubleshooting log analysis

SESN: WebSphere eXtreme Scale messages for the WebSphere eXtreme Scale HTTP Session manager

When you encounter a message in a log or other parts of the product interface, look up the message by its message ID to find out more information.

- SESN0006E
  SessionContext.createWhenStop=SESN0006E: Attempted to create a session while the WebSphere Application Server was stopping.
- SESN0007E
  SessionContext.accessWhenStop=SESN0007E: Attempted to access a session while the WebSphere Application Server was stopping.
- SESN0008E
  SessionContext.unauthAccessError=SESN0008E: A user authenticated as {0} has attempted to access a session owned by {1}.
- SESN0012E
  SessionData.putValErr1=SESN0012E: Null key entered. The HttpSession.putValue or HttpSession.setAttribute method was called from a servlet or JSP with a null key.
- SESN0013E
  SessionData.putValErr2=SESN0013E: Null value entered for key {0}. The HttpSession.putValue method was called from a servlet or JSP with a null value.
- SESN0066E
  SessionContext.responseAlreadyCommitted=SESN0066E: The response is already committed to the client. The session cookie cannot be set.
- SESN0116W
  SessionContext.maxSessionIdLengthExceeded=SESN0116W: Session identifier {0} has exceeded the max length limit of {1}.
- SESN0117I
  SessionContextRegistry.globalSessionsEnabled=SESN0117I: Global sessions is enabled for Web modules running with the Web container-level session management configuration.
- SESN0118W
  SessionContextRegistry.globalSessionTBWWarning=SESN0118W: Time-based write is enabled with global sessions. Accessing a global session from more than one server or cluster may result in loss of session data integrity.
- SESN0119W
  SessionContextRegistry.globalSessionM2MWarning=SESN0119W: Memory-to-memory replication is enabled with global sessions. Accessing a global session from more than one server or cluster may result in loss of session data integrity.
- SESN0120I
  SessionContextRegistry.SessionNotGlobalForWebApp=SESN0120I: Web module {0} will not participate in global sessions because the Web container-level session management configuration has been overridden.
- SESN0121E
  SessionContext.CrossoverOnRetrieve=SESN0121E: Session crossover detected in Web application {0}. Session {1} was retrieved when session {2} was expected -
- SESN0122E
  SessionContext.CrossoverOnReference=SESN0122E: Session crossover detected in Web application {0}. Session {1} was referenced by method {2} when session {3} was expected -
- SESN0123E
  SessionContext.CrossoverOnReturn=SESN0123E: Session crossover detected in Web application {0}. Session {1} was returned to the client when session {2} was expected -
- SESN0124W
  SessionContext.DebugCrossoverEnabled=SESN0124W: Session crossover detection is enabled.
- SESN0169I
  SessionContext.propertyFound=SESN0169I: Session Manager found the custom property {0} with value {1}.
- SESN0170W
  SessionContext.invalidPropertyFound=SESN0170W: Session Manager found the custom property {0} with a non-numeric value {1} so it will be ignored.
- SESN0171W
  SessionContext.valueOutOfRange=SESN0171W: Session Manager found the custom property {0} with out-of-range value {1} so it will use {2}.
- SESN0172I
  SessionIdGeneratorImpl.UsingDefaultSecureRandom=SESN0172I: Using the default SecureRandom implementation for ID generation.
- SESN0175I
  SessionContextRegistry.existingContext=SESN0175I: Will use an existing session context for application key {0}
- SESN0176I
  SessionContextRegistry.newContext=SESN0176I: Will create a new session context for application key {0}
- SESN0194W
  SessionProperties.serverLevelConfigOnly=SESN0194W: Session Manager found custom property {0} with value {1}. It cannot override server level configuration with value {2}. It will be ignored.
- SESN0195I
  SessionProperties.propertyFoundButAlreadySet=SESN0195I: Session Manager found custom property {0} with value {1}. Because it is the same as the server level configuration property, it will be used.
- SESN0196W
  Store.createSessionIdAlreadyExists=SESN0196W: The Id Generator generated an id that already exists.

Related tasks:
Configuring HTTP session managers

SESN0006E

SessionContext.createWhenStop=SESN0006E: Attempted to create a session while the WebSphere Application Server was stopping.

Explanation

This error occurs when a session request is received while the Application Server is stopping.

User response

Restart the Application Server.
Related tasks:
Configuring HTTP session managers

SESN0007E

SessionContext.accessWhenStop=SESN0007E: Attempted to access a session while the WebSphere Application Server was stopping.

Explanation

This error occurs when a session request is received while the Application Server is stopping.

User response

Retart the Application Server.
Related tasks:
Configuring HTTP session managers

SESN0008E

SessionContext.unauthAccessError=SESN0008E: A user authenticated as {0} has attempted to access a session owned by {1}.

Explanation

The Session Security Integration feature has detected an attempt to access a session by an unauthorized user.

User response

No user action is required.
Related tasks:
Configuring HTTP session managers

SESN0012E

SessionData.putValErr1=SESN0012E: Null key entered. The HttpSession.putValue or HttpSession.setAttribute method was called from a servlet or JSP with a null key.

Explanation

The HttpSession.putValue or HttpSession.setAttribute method cannot be called with a null key.

User response

Fix the servlet or JSP to pass a non-null key.
Related tasks:
Configuring HTTP session managers

SESN0013E

SessionData.putValErr2=SESN0013E: Null value entered for key {0}. The HttpSession.putValue method was called from a servlet or JSP with a null value.

Explanation

The HttpSession.putValue method cannot be called with a null value.

User response

Fix the servlet or JSP to pass a non-null value.
Related tasks:
Configuring HTTP session managers

SESN0066E

SessionContext.responseAlreadyCommitted=SESN0066E: The response is already committed to the client. The session cookie cannot be set.

Explanation

The response is already committed to client so the session cookie cannot be sent to client.

User response

Correct the application to access the HTTP session before writing anything to the response.
Related tasks:
Configuring HTTP session managers

SESN0116W

SessionContext.maxSessionIdLengthExceeded=SESN0116W: Session identifier {0} has exceeded the max length limit of {1}.

Explanation

The value specified for the SessionIdentifierMaxLength property has been exceeded.

User response

Only set this property if absolutely necessary. If this property is required, set it to the largest value that your installation can tolerate. If still experiencing this problem, it is likely due to repeated failovers. Investigate and fix the root cause of the failovers.
Related tasks:
Configuring HTTP session managers

SESN0117I

SessionContextRegistry.globalSessionsEnabled=SESN0117I: Global sessions is enabled for Web modules running with the Web container-level session management configuration.

Explanation

This message is for informational purposes only.

User response

If global sessions is not desired, disable global sessions by setting the Servlet21SessionCompatibility property to false.
Related tasks:
Configuring HTTP session managers

SESN0118W

SessionContextRegistry.globalSessionTBWWarning=SESN0118W: Time-based write is enabled with global sessions. Accessing a global session from more than one server or cluster may result in loss of session data integrity.

Explanation

The time-based write feature is enabled with global sessions. Unless all Web modules that access the session are in the same server or cluster, session data integrity may be lost.

User response

Disable the time-based write if Web modules that access global sessions are split across servers or clusters.
Related tasks:
Configuring HTTP session managers

SESN0119W

SessionContextRegistry.globalSessionM2MWarning=SESN0119W: Memory-to-memory replication is enabled with global sessions. Accessing a global session from more than one server or cluster may result in loss of session data integrity.

Explanation

The memory-to-memory replication feature is enabled with global sessions. Unless all Web modules that access the session are in the same server or cluster, session data integrity may be lost.

User response

Disable memory-to-memory replication if Web Modules that access global sessions are split across servers or clusters.
Related tasks:
Configuring HTTP session managers

SESN0120I

SessionContextRegistry.SessionNotGlobalForWebApp=SESN0120I: Web module {0} will not participate in global sessions because the Web container-level session management configuration has been overridden.

Explanation

The specified Web module will not participate in global sessions because the Web container-level session management configuration has been overridden either at the enterprise application- or Web module-level.

User response

If you want the Web module to participate in global sessions, disable the session management configuration that is specified at the enterprise application- or the Web module-level and restart the server.
Related tasks:
Configuring HTTP session managers

SESN0121E

SessionContext.CrossoverOnRetrieve=SESN0121E: Session crossover detected in Web application {0}. Session {1} was retrieved when session {2} was expected -

Explanation

A call to the request.getSession method returned a session other than the requested session.

User response

If the problem persists, see problem determination information on the WebSphere Application Server Support page at http://www.ibm.com/software/webservers/appserv/was/support/.
Related tasks:
Configuring HTTP session managers

SESN0122E

SessionContext.CrossoverOnReference=SESN0122E: Session crossover detected in Web application {0}. Session {1} was referenced by method {2} when session {3} was expected -

Explanation

An application referenced a session other than the session associated with the request.

User response

If the problem persists, see problem determination information on the WebSphere Application Server Support page at http://www.ibm.com/software/webservers/appserv/was/support/.
Related tasks:
Configuring HTTP session managers

SESN0123E

SessionContext.CrossoverOnReturn=SESN0123E: Session crossover detected in Web application {0}. Session {1} was returned to the client when session {2} was expected -

Explanation

A cookie or URL was returned to the client containing a session ID that is not associated with the request.

User response

If the problem persists, see problem determination information on the WebSphere Application Server Support page at http://www.ibm.com/software/webservers/appserv/was/support/.
Related tasks:
Configuring HTTP session managers

SESN0124W

SessionContext.DebugCrossoverEnabled=SESN0124W: Session crossover detection is enabled.

Explanation

Checks for session crossover are being initiated.

User response

For better performance, you may choose to disable these checks by setting the Web container custom property DebugSessionCrossover=false.
Related tasks:
Configuring HTTP session managers

SESN0169I

SessionContext.propertyFound=SESN0169I: Session Manager found the custom property {0} with value {1}.

Explanation

Session Manager will use the specified property and value to control behavior.

User response

Verify that the specified property and value will result in the desired behavior.
Related tasks:
Configuring HTTP session managers

SESN0170W

SessionContext.invalidPropertyFound=SESN0170W: Session Manager found the custom property {0} with a non-numeric value {1} so it will be ignored.

Explanation

Session Manager expected the specified property to contain a numeric value. The property will be ignored.

User response

Correct the specified property value to make it a valid number.
Related tasks:
Configuring HTTP session managers

SESN0171W

SessionContext.valueOutOfRange=SESN0171W: Session Manager found the custom property {0} with out-of-range value {1} so it will use {2}.

Explanation

Session Manager expected the specified property to have a value within a certain range. Session Manager will use the closest valid value.

User response

Correct the specified property value to make it within the documented range.
Related tasks:
Configuring HTTP session managers

SESN0172I

SessionIdGeneratorImpl.UsingDefaultSecureRandom=SESN0172I: Using the default SecureRandom implementation for ID generation.

Explanation

Session Manager is using the java default SecureRandom implementation for session ID generation.

User response

No action is required.
Related tasks:
Configuring HTTP session managers

SESN0175I

SessionContextRegistry.existingContext=SESN0175I: Will use an existing session context for application key {0}

Explanation

An existing session context is going to be shared for this application key.

User response

No action is required.
Related tasks:
Configuring HTTP session managers

SESN0176I

SessionContextRegistry.newContext=SESN0176I: Will create a new session context for application key {0}

Explanation

A new session context will be created for this application key.

User response

No action is required.
Related tasks:
Configuring HTTP session managers

SESN0194W

SessionProperties.serverLevelConfigOnly=SESN0194W: Session Manager found custom property {0} with value {1}. It cannot override server level configuration with value {2}. It will be ignored.

Explanation

The custom property can only be set at the server level configuration. The property will be ignored.

User response

Remove the custom property from the application/module level configuration. Change the server level configuration if applicable.
Related tasks:
Configuring HTTP session managers

SESN0195I

SessionProperties.propertyFoundButAlreadySet=SESN0195I: Session Manager found custom property {0} with value {1}. Because it is the same as the server level configuration property, it will be used.

Explanation

The custom property can only be set at the server level configuration. Since it is the same as the server level configuration property, it will be used.

User response

No action is required.
Related tasks:
Configuring HTTP session managers

SESN0196W

Store.createSessionIdAlreadyExists=SESN0196W: The Id Generator generated an id that already exists.

Explanation

The Id Generator generated an id that already exists. We will create another id.

User response

No action is required.
Related tasks:
Configuring HTTP session managers

SSLC: WebSphere eXtreme Scale messages for SSL channel security

When you encounter a message in a log or other parts of the product interface, look up the message by its message ID to find out more information.

- SSLC0001W
  invalid.security.level=SSLC0001W: {0} in an invalid security level. Default of high will be used.
- SSLC0002E
  invalid.security.properties=SSLC0002E: The SSL channel cannot be started due to the following incorrect settings: {0}
- SSLC0003E
  security.repertoire.not.found=SSLC0003E: Alias {0} does not map to a known security repertoire.
- SSLC0004W
  provider.not.fips.enabled=SSLC0004W: FIPS support has been requested, but the specified provider {0} might not support it.
- SSLC0005E
  no.pkcs.keystore=SSLC0005E: Unable to get a PKCS keystore.
- SSLC0006E
  unable.to.read.config=SSLC0006E: Error reading the SSL channel configuration, exception: {0}
- SSLC0007W
  security.ssl.config.initialization.warning.invalidkeystoretype=SSLC0007W: The keystore or truststore type specified is invalid. Automatically adjusting to use the correct type, however, please correct the SSL configuration for performance reasons.
- SSLC0008E
  handshake.failure=SSLC0008E: Unable to initialize SSL connection. Unauthorized access was denied or security settings have expired. Exception is {0}.
- SSLC0009E
  hwcrypto.not.supported=SSLC0009E: Hardware crypto is not supported.
- SSLC0010E
  empty.keystore=SSLC0010E: Keystore file exists, but is empty: {0}
- SSLC0011E
  no.keystore.and.hwcrypto=SSLC0011E: No keystore specified and no hardware crypto defined.
- SSLC0012E
  null.truststore=SSLC0012E: Invalid truststore name of null.
- SSLC0013E
  invalid.keystore.password=SSLC0013E: Invalid password for keystore {0}. Internal exception {1}.

Related concepts:
Security

SSLC0001W

invalid.security.level=SSLC0001W: {0} in an invalid security level. Default of high will be used.

Explanation

Valid values for security level are low, medium and high.

User response

Ensure that the security level is set to a valid value.
Related concepts:
Security

SSLC0002E

invalid.security.properties=SSLC0002E: The SSL channel cannot be started due to the following incorrect settings: {0}

Explanation

One or more settings for security in the SSL channel were not valid.

User response

The SSL channel security settings should be modified to the correct values.
Related concepts:
Security

SSLC0003E

security.repertoire.not.found=SSLC0003E: Alias {0} does not map to a known security repertoire.

Explanation

If a configuration alias is specified on the SSL channel, it must map to a security repertoire.

User response

The SSL channel configuration must be updated to reference a valid security repertoire.
Related concepts:
Security

SSLC0004W

provider.not.fips.enabled=SSLC0004W: FIPS support has been requested, but the specified provider {0} might not support it.

Explanation

Not enough information is available to determine if the specified provider is FIPS enabled.

User response

Verify that the specified provider has the appropriate FIPS support.
Related concepts:
Security

SSLC0005E

no.pkcs.keystore=SSLC0005E: Unable to get a PKCS keystore.

Explanation

The token information provided did not identify a valid PKCS keystore.

User response

Review and update the token information to identify a valid PKCS keystore.
Related concepts:

Security

SSLC0006E

unable.to.read.config=SSLC0006E: Error reading the SSL channel configuration, exception: {0}

Explanation

An exception was created when reading the SSL channel configuration.

User response

Problems in the SSL channel configuration should be fixed.
Related concepts:
Security

SSLC0007W

security.ssl.config.initialization.warning.invalidkeystoretype=SSLC0007W: The keystore or truststore type specified is invalid. Automatically adjusting to use the correct type, however, please correct the SSL configuration for performance reasons.

Explanation

The keystore or truststore type specified is not valid.

User response

Modify the SSL configuration so that the keystore or truststore type is a valid type. You can check the keystore and truststore types by loading them in WebSphere's IKeyMan tool.
Related concepts:
Security

SSLC0008E

handshake.failure=SSLC0008E: Unable to initialize SSL connection. Unauthorized access was denied or security settings have expired. Exception is {0}.

Explanation

A new connection failed to complete a successful secure handshake.

User response

Check security settings. The certificate might have expired. Alternatively, an unauthorized client connection might have been denied.
Related concepts:
Security

SSLC0009E

hwcrypto.not.supported=SSLC0009E: Hardware crypto is not supported.

Explanation

Support for hardware cryptography cards is not available.

User response

Change the security settings to disable the use of hardware cryptography.
Related concepts:
Security

SSLC0010E

empty.keystore=SSLC0010E: Keystore file exists, but is empty: {0}

Explanation

The keystore from the configuration was found, but had no contents.

User response

Replace the current keystore file with valid keystore.
Related concepts:
Security

SSLC0011E

no.keystore.and.hwcrypto=SSLC0011E: No keystore specified and no hardware crypto defined.

Explanation

A keystore or hardware cryptography setting is required in the security configuration, but neither exist.

User response

Update the security settings to specify a keystore or to enable hardware cryptography.
Related concepts:
Security

SSLC0012E

null.truststore=SSLC0012E: Invalid truststore name of null.

Explanation

The truststore name is not present in the security configuration.

User response

Update the security settings to specify a valid truststore file.
Related concepts:
Security

SSLC0013E

invalid.keystore.password=SSLC0013E: Invalid password for keystore {0}. Internal exception {1}.

Explanation

The password provided for the keystore is not compatible.

User response

Check the keystore password in the security settings, or specify a new keystore and password.
Related concepts:
Security

TCPC: WebSphere eXtreme Scale messages for the TCP channel component

When you encounter a message in a log or other parts of the product interface, look up the message by its message ID to find out more information.

- TCPC0001I
  TCP_CHANNEL_STARTED=TCPC0001I: TCP Channel {0} is listening on host {1} port {2}.
- TCPC0002I
  TCP_CHANNEL_STOPPED=TCPC0002I: TCP Channel {0} has stopped listening on host {1} port {2}.
- TCPC0003E
  BIND_ERROR=TCPC0003E: TCP Channel {0} initialization failed. The socket bind failed for host {1} and port {2}. The port may already be in use.
- TCPC0004W
  MAX_CONNS_EXCEEDED=TCPC0004W: TCP Channel {0} has exceeded the maximum number of open connections {1}.
- TCPC0005W
  THREAD_DISPATCH_FAILED=TCPC0005W: TCP Channel {0} could not obtain thread from thread pool {1}.

- TCPC0006E
  LOCAL_HOST_UNRESOLVED=TCPC0006E: TCP Channel {0} initialization failed. The host {1} and port {2} could not be resolved.
- TCPC0007E
  PORT_NOT_ACCEPTING=TCPC0007E: TCP Channel {0} listening on host {1} port {2} has stopped accepting connections.
- TCPC0801E
  MAX_OPEN_CONNECTIONS_INVALID=TCPC0801E: The maximum number of open connections {0} is not valid. Valid values must be no less than {1} and no greater than {2}.
- TCPC0802E
  INACTIVITY_TIMEOUT_INVALID=TCPC0802E: The inactivity time out of {0} is not valid. Valid values must be no less than {1} and no greater than {2}.
- TCPC0803E
  ADDRESS_EXCLUDE_LIST_INVALID=TCPC0803E: An entry in the address exclude list for the TCP Channel {0} was not valid. Valid values consist of a valid String.
- TCPC0804E
  ADDRESS_INCLUDE_LIST_INVALID=TCPC0804E: An entry in the address include list for the TCP Channel {0} was not valid. Valid values consist of a valid String.
- TCPC0805E
  HOST_NAME_EXCLUDE_LIST_INVALID=TCPC0805E: An entry in the host name exclude list for the TCP Channel {0} was not valid. Valid values consist of a valid String.
- TCPC0806E
  HOST_NAME_INCLUDE_LIST_INVALID=TCPC0806E: An entry in the host name include list for the TCP Channel {0} was not valid. Valid values consist of a valid String.
- TCPC0807E
  NOT_VALID_CUSTOM_PROPERTY=TCPC0807E: TCPChannel {0} custom property {1} has an value of {2}. This value is not valid.
- TCPC0808E
  CONFIG_VALUE_NUMBER_EXCEPTION=TCPC0808E: TCP Channel {0} has been configured with an incorrect number value for a property, Property Name: {1} Value: {2}
- TCPC0809E
  CONFIG_VALUE_NOT_VALID_STRING=TCPC0809E: TCP Channel {0} has been constructed with incorrect configuration property value, Name: {1} Value: {2}
- TCPC0810E
  CONFIG_VALUE_NOT_VALID_NULL_STRING=TCPC0810E: TCP Channel {0} has been constructed with a null configuration property value, Name: {1}
- TCPC0811E
  CONFIG_VALUE_NOT_VALID_BOOLEAN2=TCPC0811E: TCP Channel {0} has been constructed with incorrect configuration property value, Name: {1} Value: {2} Valid Range: 0 - False, 1 - True
- TCPC0812E
  CONFIG_VALUE_NOT_VALID_INT=TCPC0812E: TCP Channel {0} has been constructed with incorrect configuration property value, Name: {1} Value: {2} Valid Range: Minimum {3}, Maximum {4}
- TCPC0813W
  CONFIG_KEY_NOT_VALID=TCPC0813W: TCP Channel {0} has been constructed with incorrect configuration property, Property Name: {1} Value: {2}
- TCPC0814W
  UNRECOGNIZED_CUSTOM_PROPERTY=TCPC0814W: The TCP Channel {0} has a custom property configured which is not a recognized property, Property Name: {1}
- TCPC0815E
  NEW_CONFIG_VALUE_NOT_EQUAL=TCPC0815E: An attempt to update TCP Channel {0} failed because a property which can not be updated at runtime has been given a new value that is different from the current value. Propery Name: {1} Current Value: {2} Failed Updated Value: {3}
- TCPC0816E
  UPDATED_CONFIG_NOT_IMPLEMENTED=TCPC0816E: An attempt to update TCP Channel {0} failed.
- TCPC0817E
  NO_ENDPOINT_NAME=TCPC0817E: No Endpoint Name was assigned to TCP Channel {0}.
- TCPC0818E
  USER_ID_NOT_VALID=TCPC0818E: Configured user ID to switch to after starting was not valid. User ID: {0}
- TCPC0819E
  GROUP_ID_NOT_VALID=TCPC0819E: Configured group ID to switch to after starting was not valid. Group ID: {0}

Related reference:
Trace options

TCPC0001I

TCP_CHANNEL_STARTED=TCPC0001I: TCP Channel {0} is listening on host {1} port {2}.

Explanation

The specified TCP Channel has been started and is now listening for requests.

User response

Informational message only, no action required.
Related reference:
Trace options

TCPC0002I

TCP_CHANNEL_STOPPED=TCPC0002I: TCP Channel {0} has stopped listening on host {1} port {2}.

Explanation

The specified TCP Channel has been stopped and is no longer listening for requests.

User response

Information message only, no action required.
Related reference:
Trace options

TCPC0003E

BIND_ERROR=TCPC0003E: TCP Channel {0} initialization failed. The socket bind failed for host {1} and port {2}. The port may already be in use.

Explanation

The socket bind operation failed. Common cause is that the port number is already in use.

User response

Check that the TCP Channel has been configured to use the correct port number.
Related reference:
Trace options

TCPC0004W

MAX_CONNS_EXCEEDED=TCPC0004W: TCP Channel {0} has exceeded the maximum number of open connections {1}.

Explanation

The specified TCP Channel has exceeded the maximum number of open connections and is refusing some connections.

User response

If more connections are required, use either the admin application or the command line scripting tool to update the maximum number of connections allowed.
Related reference:
Trace options

TCPC0005W

THREAD_DISPATCH_FAILED=TCPC0005W: TCP Channel {0} could not obtain thread from thread pool {1}.

Explanation

The specified TCP Channel could not obtain a thread from the specified thread pool. Processing of subsequent requests may be delayed.

User response

If CPU is not fully utilized, increase the maximum number of threads in the thread pool. Otherwise, reduce the number of incoming requests.
Related reference:
Trace options

TCPC0006E

LOCAL_HOST_UNRESOLVED=TCPC0006E: TCP Channel {0} initialization failed. The host {1} and port {2} could not be resolved.

Explanation

The host name specified could not be resolved. The host name was specified incorrectly or has not been defined on this system.

User response

Check that the TCP Channel has been configured to use the correct host name.
Related reference:
Trace options

TCPC0007E

PORT_NOT_ACCEPTING=TCPC0007E: TCP Channel {0} listening on host {1} port {2} has stopped accepting connections.

Explanation

The specified TCP Channel can no longer accept connections for a port. The port appears to be disabled.

User response

To use the port, enable the port and restart the application.
Related reference:
Trace options

TCPC0801E

MAX_OPEN_CONNECTIONS_INVALID=TCPC0801E: The maximum number of open connections {0} is not valid. Valid values must be no less than {1} and no greater than {2}.

Explanation

No additional information.

User response

Correct the configuration error.
Related reference:
Trace options

TCPC0802E

INACTIVITY_TIMEOUT_INVALID=TCPC0802E: The inactivity time out of {0} is not valid. Valid values must be no less than {1} and no greater than {2}.

Explanation

No additional information.

User response

Correct the configuration error.
Related reference:
Trace options

TCPC0803E

ADDRESS_EXCLUDE_LIST_INVALID=TCPC0803E: An entry in the address exclude list for the TCP Channel {0} was not valid. Valid values consist of a valid String.

Explanation

No additional information.

User response

Correct the configuration error.
Related reference:
Trace options

TCPC0804E

ADDRESS_INCLUDE_LIST_INVALID=TCPC0804E: An entry in the address include list for the TCP Channel {0} was not valid. Valid values consist of a valid String.

Explanation

No additional information.

User response

Correct the configuration error.
Related reference:
Trace options

TCPC0805E

HOST_NAME_EXCLUDE_LIST_INVALID=TCPC0805E: An entry in the host name exclude list for the TCP Channel {0} was not valid. Valid values consist of a valid String.

Explanation

No additional information.

User response

Correct the configuration error.
Related reference:
Trace options

TCPC0806E

HOST_NAME_INCLUDE_LIST_INVALID=TCPC0806E: An entry in the host name include list for the TCP Channel {0} was not valid. Valid values consist of a valid String.

Explanation

No additional information.

User response

Correct the configuration error.
Related reference:
Trace options

TCPC0807E

NOT_VALID_CUSTOM_PROPERTY=TCPC0807E: TCPChannel {0} custom property {1} has an value of {2}. This value is not valid.

Explanation

A custom property was specified with a value that is outside the range of valid values.

User response

Correct the configuration error.
Related reference:
Trace options

TCPC0808E

CONFIG_VALUE_NUMBER_EXCEPTION=TCPC0808E: TCP Channel {0} has been configured with an incorrect number value for a property, Property Name: {1} Value: {2}

Explanation

A TCP Channel configuration property value is not a valid number

User response

Correct the configuration error.
Related reference:
Trace options

TCPC0809E

CONFIG_VALUE_NOT_VALID_STRING=TCPC0809E: TCP Channel {0} has been constructed with incorrect configuration property value, Name: {1} Value: {2}

Explanation

A TCP Channel property value is not within the valid range for this property

User response

Correct the configuration error.
Related reference:
Trace options

TCPC0810E

CONFIG_VALUE_NOT_VALID_NULL_STRING=TCPC0810E: TCP Channel {0} has been constructed with a null configuration property value, Name: {1}

Explanation

A TCP Channel property value is not within the valid range for this property

User response

Correct the configuration error.
Related reference:
Trace options

TCPC0811E

CONFIG_VALUE_NOT_VALID_BOOLEAN2=TCPC0811E: TCP Channel {0} has been constructed with incorrect configuration property value, Name: {1} Value: {2} Valid Range: 0 - False, 1 - True

Explanation

A TCP Channel property value is not within the valid range for this property

User response

Correct the configuration error.
Related reference:
Trace options

TCPC0812E

CONFIG_VALUE_NOT_VALID_INT=TCPC0812E: TCP Channel {0} has been constructed with incorrect configuration property value, Name: {1} Value: {2} Valid Range: Minimum {3}, Maximum {4}

Explanation

A TCP Channel property value is not within the valid range for this property

User response

Correct the configuration error.
Related reference:
Trace options

TCPC0813W

CONFIG_KEY_NOT_VALID=TCPC0813W: TCP Channel {0} has been constructed with incorrect configuration property, Property Name: {1} Value: {2}

Explanation

A TCP Channel configuration property which is not recognized has been used.

User response

Correct the configuration error.
Related reference:
Trace options

TCPC0814W

UNRECOGNIZED_CUSTOM_PROPERTY=TCPC0814W: The TCP Channel {0} has a custom property configured which is not a recognized property, Property Name: {1}

Explanation

A custom property was specified incorrectly.

User response

Correct the configuration error.
Related reference:
Trace options

TCPC0815E

NEW_CONFIG_VALUE_NOT_EQUAL=TCPC0815E: An attempt to update TCP Channel {0} failed because a property which can not be updated at runtime has been given a new value that is different from the current value. Propery Name: {1} Current Value: {2} Failed Updated Value: {3}

Explanation

A TCP Channel property value was given a new value during an attempt to update the configuration, but this property can not be updated with a new value at runtime.

User response

Correct the configuration error.
Related reference:
Trace options

TCPC0816E

UPDATED_CONFIG_NOT_IMPLEMENTED=TCPC0816E: An attempt to update TCP Channel {0} failed.

Explanation

A TCP Channel could not be updated because the new configuration was not a valid configuration.

User response

Correct the configuration error.
Related reference:
Trace options

TCPC0817E

NO_ENDPOINT_NAME=TCPC0817E: No Endpoint Name was assigned to TCP Channel {0}.

Explanation

No additional information.

User response

Correct the configuration error.
Related reference:
Trace options

TCPC0818E

USER_ID_NOT_VALID=TCPC0818E: Configured user ID to switch to after starting was not valid. User ID: {0}

Explanation

No additional information.

User response

Change configured user id to a new value.
Related reference:
Trace options

TCPC0819E

GROUP_ID_NOT_VALID=TCPC0819E: Configured group ID to switch to after starting was not valid. Group ID: {0}

Explanation

No additional information.

User response

Change configured group id to a new value.
Related reference:
Trace options

WSBB: WebSphere eXtreme Scale messages for the XsByteBuffer component

When you encounter a message in a log or other parts of the product interface, look up the message by its message ID to find out more information.

- WSBB0861E
  NOT_VALID_CUSTOM_PROPERTY=WSBB0861E: The custom property {0} has an value of {1}. This value is not valid.
- WSBB0862W
  UNRECOGNIZED_CUSTOM_PROPERTY=WSBB0862W: The custom property {0} specified for the XsByteBuffer Component is not valid.
- WSBB0863E
  CONFIG_VALUE_NUMBER_EXCEPTION=WSBB0863E: The XsByteBuffer Component caught a NumberFormatException processing property, Property Name: {0} Value: {1}
- WSBB0864E
  POOL_MISMATCH=WSBB0864E: The XsByteBuffer Pool Sizes and Pool Depths specification do not have the same number of entries, Sizes: {0} Depths: {1}

WSBB0861E

NOT_VALID_CUSTOM_PROPERTY=WSBB0861E: The custom property {0} has an value of {1}. This value is not valid.

Explanation

A custom property was specified with a value that is outside the range of valid values.

User response

Correct the configuration error.

WSBB0862W

UNRECOGNIZED_CUSTOM_PROPERTY=WSBB0862W: The custom property {0} specified for the XsByteBuffer Component is not valid.

Explanation

A custom property was specified incorrectly.

User response

Correct the configuration error.

WSBB0863E

CONFIG_VALUE_NUMBER_EXCEPTION=WSBB0863E: The XsByteBuffer Component caught a NumberFormatException processing property, Property Name: {0} Value: {1}

Explanation

A XsByteBuffer configuration property value is not a valid number

User response

Correct the configuration error.

WSBB0864E

POOL_MISMATCH=WSBB0864E: The XsByteBuffer Pool Sizes and Pool Depths specification do not have the same number of entries, Sizes: {0} Depths: {1}

Explanation

The Pool Sizes or Pool Depths were specified incorrectly.

User response

Correct the configuration error.

XSMI: WebSphere eXtreme Scale messages for migration

When you encounter a message in a log or other parts of the product interface, look up the message by its message ID to find out more information.

- XSMI0000E
  XSMI0000E=XSMI0000E: Could not find text for message ID {0}.
- XSMI0001I
  XSMI0001I=XSMI0001I: Migrating {0} configuration from {1} to {2}.
- XSMI0002I
  XSMI0002I=XSMI0002I: Migrating from {0}
- XSMI0003I
  XSMI0003I=XSMI0003I: Migrating to {0}
- XSMI0004I
  XSMI0004I=XSMI0004I: Reading actions from {0}
- XSMI0005I
  XSMI0005I=XSMI0005I: Logging level is {0}
- XSMI0006I
  XSMI0006I=XSMI0006I: Logging to {0}
- XSMI0007I
  XSMI0007I=XSMI0007I: Set variable {0} = {1}
- XSMI0008I
  XSMI0008I=XSMI0008I: Executing {0} - action {1} of {2}
- XSMI0009I
  XSMI0009I=XSMI0009I: End of {0} migration. RC={1}
- XSMI0010I
  XSMI0010I=XSMI0010I: {0}
- XSMI0011I
  XSMI0011I=XSMI0011I: {0} has already been migrated in profile {1}
- XSMI0012I
  XSMI0012I=XSMI0012I: No pending migration actions found.
- XSMI2001I
  XSMI2001I=XSMI2001I: Backing up configuration located at {0} for profile {1}.
- XSMI2002I
  XSMI2002I=XSMI2002I: Removing backup of configuration for profile {0} .
- XSMI2003I
  XSMI2003I=XSMI2003I: {0}

- XSMI2004I
  XSMI2004I=XSMI2004I: XDPreUpgrade Setting variable {0} = {1}
- XSMI2005I
  XSMI2005I=XSMI2005I: Executing {0} - action {1} of {2}
- XSMI2006I
  XSMI2006I=XSMI2006I: Reading actions from {0}
- XSMI2007I
  XSMI2007I=XSMI2007I: Logging level is {0}
- XSMI2008I
  XSMI2008I=XSMI2008I: Logging to {0}
- XSMI2009I
  XSMI2009I=XSMI2009I: End of backup of configuration for profile. RC={1}
- XSMI2010I
  XSMI2010I=XSMI2010I: End of removal of configuration backup for profile. RC={1}
- XSMI9000E
  XSMI9000E=XSMI9000E: An error occurred during migration. See {0}.
- XSMI9001E
  XSMI9001E=XSMI9001E: Option {0} is unknown.
- XSMI9002E
  XSMI9002E=XSMI9002E: Parameter required for option: {0}
- XSMI9003E
  XSMI9003E=XSMI9003E: Parameter on {0} option is not valid.
- XSMI9004E
  XSMI9004E=XSMI9004E: Directory {0} is not a valid WebSphere profile.
- XSMI9005E
  XSMI9005E=XSMI9005E: Profile {0} contains more than one cell.
- XSMI9006E
  XSMI9006E=XSMI9006E: Profile {0} does not contain a WebSphere cell.
- XSMI9007E
  XSMI9007E=XSMI9007E: Profile {0} does not contain a dmgr node.
- XSMI9008E
  XSMI9008E=XSMI9008E: An error occurred during backup of configuration. See {0}.
- XSMI9009E
  XSMI9009E=XSMI9009E: An error occurred during the removal of a configuration backup. See {0}.
- XSMI9010E
  XSMI9010E=XSMI9010E: Source cell name {0} does not match target cell name {1}.
- XSMI9011E
  XSMI9011E=XSMI9011E: Source node name {0} does not match target node name {1}.
- XSMI9012E
  XSMI9012E=XSMI9012E: Source Node {0} contains a DMGR and target node {1} does not.
- XSMI9013E
  XSMI9013E=XSMI9013E: Target Node {0} contains a DMGR and source node {1} does not.
- XSMI9014E
  XSMI9014E=XSMI9014E: WebSphere eXtreme Scale is not installed in the Source install location {0}.
- XSMI9015E
  XSMI9015E=XSMI9015E: {0} does not contain a valid WebSphere installation.
- XSMI9016E
  XSMI9016E=XSMI9016E: When entering userid and password, both parameters must be entered.

Related concepts:
Upgrading and migrating WebSphere eXtreme Scale

XSMI0000E

XSMI0000E=XSMI0000E: Could not find text for message ID {0}.

Explanation

WebSphere could not find the message text for the indicated message ID.

User response

Open a problem report with IBM service.
Related concepts:
Upgrading and migrating WebSphere eXtreme Scale

XSMI0001I

XSMI0001I=XSMI0001I: Migrating {0} configuration from {1} to {2}.

Explanation

Migration of a WebSphere eXtreme Scale configuration has been initiated by the user.

User response

None
Related concepts:
Upgrading and migrating WebSphere eXtreme Scale

XSMI0002I

XSMI0002I=XSMI0002I: Migrating from {0}

Explanation

Indicates the install directory from which a WebSphere eXtreme Scale configuration is being migrated.

User response

None
Related concepts:
Upgrading and migrating WebSphere eXtreme Scale

XSMI0003I

XSMI0003I=XSMI0003I: Migrating to {0}

Explanation

Indicates the profile directory to which the WebSphere eXtreme Scale configuration is being migrated.

User response

None
Related concepts:
Upgrading and migrating WebSphere eXtreme Scale

XSMI0004I

XSMI0004I=XSMI0004I: Reading actions from {0}

Explanation

Indicates the directory from which migration actions are being read.

User response

None
Related concepts:
Upgrading and migrating WebSphere eXtreme Scale

XSMI0005I

XSMI0005I=XSMI0005I: Logging level is {0}

Explanation

Indicates the logging level being used.

User response

None
Related concepts:
Upgrading and migrating WebSphere eXtreme Scale

XSMI0006I

XSMI0006I=XSMI0006I: Logging to {0}

Explanation

Indicates location and name of XDUpgrade log.

User response

None
Related concepts:
Upgrading and migrating WebSphere eXtreme Scale

XSMI0007I

XSMI0007I=XSMI0007I: Set variable {0} = {1}

Explanation

The indicated XDUpgrade variable {0} is set to the indicated value {1}

User response

None
Related concepts:
Upgrading and migrating WebSphere eXtreme Scale

XSMI0008I

XSMI0008I=XSMI0008I: Executing {0} - action {1} of {2}

Explanation

XDUpgrade is executing action {0}, which is action {1} of a total of {2}.

User response

None
Related concepts:
Upgrading and migrating WebSphere eXtreme Scale

XSMI0009I

XSMI0009I=XSMI0009I: End of {0} migration. RC={1}

Explanation

Migration of a WebSphere eXtreme Scale configuration has completed with the indicated return code.

User response

None
Related concepts:
Upgrading and migrating WebSphere eXtreme Scale

XSMI0010I

XSMI0010I=XSMI0010I: {0}

Explanation

Displays correct syntax for XDUpgrade command.

User response

None
Related concepts:
Upgrading and migrating WebSphere eXtreme Scale

XSMI0011I

XSMI0011I=XSMI0011I: {0} has already been migrated in profile {1}

Explanation

The specified package has already been migrated.

User response

None
Related concepts:
Upgrading and migrating WebSphere eXtreme Scale

XSMI0012I

XSMI0012I=XSMI0012I: No pending migration actions found.

Explanation

XDUpgrade was invoked, but no packages that require migration were found.

User response

None
Related concepts:
Upgrading and migrating WebSphere eXtreme Scale

XSMI2001I

XSMI2001I=XSMI2001I: Backing up configuration located at {0} for profile {1}.

Explanation

Backup of configuration been initiated by the user.

User response

None
Related concepts:
Upgrading and migrating WebSphere eXtreme Scale

XSMI2002I

XSMI2002I=XSMI2002I: Removing backup of configuration for profile {0} .

Explanation

Removal of a backup of configuration been initiated by the user.

User response

None
Related concepts:
Upgrading and migrating WebSphere eXtreme Scale

XSMI2003I

XSMI2003I=XSMI2003I: {0}

Explanation

Displays correct syntax for XDPreUpgrade command.

User response

None
Related concepts:
Upgrading and migrating WebSphere eXtreme Scale

XSMI2004I

XSMI2004I=XSMI2004I: XDPreUpgrade Setting variable {0} = {1}

Explanation

XDPreUpgrade has set the specified variable to the specified value

User response

None
Related concepts:
Upgrading and migrating WebSphere eXtreme Scale

XSMI2005I

XSMI2005I=XSMI2005I: Executing {0} - action {1} of {2}

Explanation

XDPreUpgrade is executing action {0}, which is action {1} of a total of {2}.

User response

None
Related concepts:
Upgrading and migrating WebSphere eXtreme Scale

XSMI2006I

XSMI2006I=XSMI2006I: Reading actions from {0}

Explanation

Indicates the directory from which migration actions are being read.

User response

None
Related concepts:
Upgrading and migrating WebSphere eXtreme Scale

XSMI2007I

XSMI2007I=XSMI2007I: Logging level is {0}

Explanation

Indicates the logging level being used.

User response

None
Related concepts:

Upgrading and migrating WebSphere eXtreme Scale

XSMI2008I

XSMI2008I=XSMI2008I: Logging to {0}

Explanation

Indicates location and name of XDUpgrade log.

User response

None
Related concepts:
Upgrading and migrating WebSphere eXtreme Scale

XSMI2009I

XSMI2009I=XSMI2009I: End of backup of configuration for profile. RC={1}

Explanation

Backup of a WebSphere eXtreme Scale configuration has completed with the indicated return code.

User response

None
Related concepts:
Upgrading and migrating WebSphere eXtreme Scale

XSMI2010I

XSMI2010I=XSMI2010I: End of removal of configuration backup for profile. RC={1}

Explanation

Removal of a WebSphere eXtreme Scale configuration backup has completed with the indicated return code.

User response

None
Related concepts:
Upgrading and migrating WebSphere eXtreme Scale

XSMI9000E

XSMI9000E=XSMI9000E: An error occurred during migration. See {0}.

Explanation

An error occurred during execution of XDUpgrade. Details of the problem should show in the XDUpgrade log file.

User response

Contact IBM Service.
Related concepts:
Upgrading and migrating WebSphere eXtreme Scale

XSMI9001E

XSMI9001E=XSMI9001E: Option {0} is unknown.

Explanation

Command issued with incorrect options.

User response

Reissue command with correct options.
Related concepts:
Upgrading and migrating WebSphere eXtreme Scale

XSMI9002E

XSMI9002E=XSMI9002E: Parameter required for option: {0}

Explanation

Required parameter not specified for indicated option.

User response

Reissue command with required parameter.
Related concepts:
Upgrading and migrating WebSphere eXtreme Scale

XSMI9003E

XSMI9003E=XSMI9003E: Parameter on {0} option is not valid.

Explanation

Command issued with parameter that was not valid.

User response

Reissue command with valid parameter.
Related concepts:
Upgrading and migrating WebSphere eXtreme Scale

XSMI9004E

XSMI9004E=XSMI9004E: Directory {0} is not a valid WebSphere profile.

Explanation

Specified profile name is not an actual WebSphere profile.

User response

Review available profile names and reissue command with a valid profile name.
Related concepts:
Upgrading and migrating WebSphere eXtreme Scale

XSMI9005E

XSMI9005E=XSMI9005E: Profile {0} contains more than one cell.

Explanation

Specified profile contains more than one cell.

User response

This is not a valid WebSphere configuration. Remove cell directories from profile until there is only one cell and reissue command.
Related concepts:
Upgrading and migrating WebSphere eXtreme Scale

XSMI9006E

XSMI9006E=XSMI9006E: Profile {0} does not contain a WebSphere cell.

Explanation

Specified profile does not contain a WebSphere cell.

User response

Review available profile names and reissue command with a valid profile name.
Related concepts:
Upgrading and migrating WebSphere eXtreme Scale

XSMI9007E

XSMI9007E=XSMI9007E: Profile {0} does not contain a dmgr node.

Explanation

Specified profile is not a deployment manager profile.

User response

Review available profile names and reissue command with a valid deployment manager profile name.
Related concepts:
Upgrading and migrating WebSphere eXtreme Scale

XSMI9008E

XSMI9008E=XSMI9008E: An error occurred during backup of configuration. See {0}.

Explanation

An error occurred during execution of XDPreUpgrade. Details of the problem should show in the XDPreUpgrade log file.

User response

Contact IBM Service.
Related concepts:
Upgrading and migrating WebSphere eXtreme Scale

XSMI9009E

XSMI9009E=XSMI9009E: An error occurred during the removal of a configuration backup. See {0}.

Explanation

An error occurred during execution of XDPreUpgrade. Details of the problem should show in the XDPreUpgrade log file.

User response

Contact IBM Service.
Related concepts:
Upgrading and migrating WebSphere eXtreme Scale

XSMI9010E

XSMI9010E=XSMI9010E: Source cell name {0} does not match target cell name {1}.

Explanation

The target and source cell names are different.

User response

Determine correct cell and reissue command.
Related concepts:
Upgrading and migrating WebSphere eXtreme Scale

XSMI9011E

XSMI9011E=XSMI9011E: Source node name {0} does not match target node name {1}.

Explanation

The target and source node names are different.

User response

Determine correct nodes and reissue command.
Related concepts:
Upgrading and migrating WebSphere eXtreme Scale

XSMI9012E

XSMI9012E=XSMI9012E: Source Node {0} contains a DMGR and target node {1} does not.

Explanation

The target and source node names are different.

User response

Determine correct nodes and reissue command.
Related concepts:
Upgrading and migrating WebSphere eXtreme Scale

XSMI9013E

XSMI9013E=XSMI9013E: Target Node {0} contains a DMGR and source node {1} does not.

Explanation

The target and source node names are different.

User response

Determine correct nodes and reissue command.
Related concepts:
Upgrading and migrating WebSphere eXtreme Scale

XSMI9014E

XSMI9014E=XSMI9014E: WebSphere eXtreme Scale is not installed in the Source install location {0}.

Explanation

WebSphere eXtreme Scale is not installed in the Source install location.

User response

Determine correct source install location.
Related concepts:
Upgrading and migrating WebSphere eXtreme Scale

XSMI9015E

XSMI9015E=XSMI9015E: {0} does not contain a valid WebSphere installation.

Explanation

WebSphere eXtreme Scale is not installed in the Source install.

User response

Determine correct source install location.
Related concepts:
Upgrading and migrating WebSphere eXtreme Scale

XSMI9016E

XSMI9016E=XSMI9016E: When entering userid and password, both parameters must be entered.

Explanation

Either the userid or the password parameter was not found.

User response

Enter correct userid and password parameters.
Related concepts:
Upgrading and migrating WebSphere eXtreme Scale

User interface settings

This reference information describes settings that you can view and configure on the pages of the WebSphere® Application Serveradministrative console and elsewhere.

To open the Table of Contents to the location of this reference information, click the Show in table of contents () button on the information center border. For example, if you found this page from the information center search or from an internet search engine, such as Google, click the button to make the information center show you the location of this page in the information center navigation tree. You will be able to browse the contents indented under this navigation entry.

- eXtreme Scale session management settings
  You can configure your WebSphere Application Server applications to use WebSphere eXtreme Scale or a WebSphere DataPower® XC10 Appliance for session persistence.
- Catalog service domain collection
  Use this page to manage catalog service domains. Catalog service domains define a group of catalog servers that manage the placement of shards and monitors the health of container servers in your data grid.
- Catalog service domain settings
  Use this page to manage the settings for a specific catalog service domain. Catalog service domains define a group of catalog servers that manage the placement of shards and monitors the health of container servers in your data grid. You can define a catalog service domain that is in the same cell as your deployment manager. You can also define remote catalog service domains if your WebSphere eXtreme Scale configuration is in a different cell or your data grid is made up of Java™ SE processes.
- Client security properties
  Use this page to configure client security for a catalog service domain. These settings apply to all the servers in your catalog service domain. These properties can be overridden by specifying a splicer.properties file with the com.ibm.websphere.xs.sessionFilterProps custom property or by splicing the application EAR file.
- Catalog service domain custom properties
  You can further edit the configuration of the catalog service domain by defining custom properties.

eXtreme Scale session management settings

You can configure your WebSphere® Application Server applications to use WebSphere eXtreme Scale or a WebSphere DataPower® XC10 Appliance for session persistence.

You can edit these settings in the enterprise application installation wizard, or on the application or server detail pages:

- Version 6.1: Applications > Install new application
- Version 6.1: Applications > Enterprise Applications > *application_name*
- Version 6.1: Servers > Application servers > *server_name* > Web container settings > Session management
- Version 7.0: Applications > New application > New Enterprise Application, and choose the Detailed path for creating the application.
- Version 7.0: Applications > Application Types > WebSphere enterprise applications > *application_name* > Web Module properties > Session management > Session management settings
- Version 7.0: Servers > Server Types > WebSphere application servers > *server_name* > Container settings > Session management settings

Enable session management

Enables session management to use WebSphere eXtreme Scale embedded or remote data grid or a WebSphere DataPower XC10 Appliance for session persistence.

Manage session persistence by

Specifies how session persistence is managed. You can choose one of the following options:

- WebSphere DataPower XC10 Appliance
- Remote eXtreme Scale data grid
- Embedded eXtreme Scale data grid

The remaining settings that you configure depend on the session persistence mechanism you choose.

WebSphere DataPower XC10 Appliance specific settings

The following settings are specific to configuring the WebSphere DataPower XC10 Appliance for session persistence.

IP or host name of the WebSphere DataPower XC10 Appliance

Specifies the IP or host name of the appliance to use for persisting sessions.

IBM® WebSphere DataPower XC10 Appliance administrative credentials

Specifies the User name and Password that you use to log in to the DataPower XC10 Appliance user interface. Click Test Connection... to test the connection to your appliance.

Session persistence preference

Specifies the data grid on which sessions are persisted. You can choose one of the following options:

- Persist sessions in a new data grid on the IBM WebSphere DataPower XC10 Appliance. You can then specify a Data grid name.
- Persist sessions in an existing data grid on the IBM WebSphere DataPower XC10 Appliance. You can then enter or browse for an Existing data grid name.

Remote eXtreme Scale data grid configuration

The following settings are specific to configuring the remote eXtreme Scale grid for session persistence.

Catalog service domain that manages the remote session data grid

Specifies the catalog service domain that you want to use to manage your sessions.

If no catalog service domains are displayed, or you want to create a new catalog service domain, click System administration > WebSphere eXtreme Scale > Catalog service domains.

Remote data grid in which to store session information

Specifies the name of the data grid in the catalog service domain in which you want to store your session information. The list of active remote grids is populated when you select a catalog service. The remote data grid must already exist in the eXtreme Scale configuration.

Embedded eXtreme Scale data grid configuration

The following settings are specific to configuring an embedded eXtreme Scale configuration. In the embedded eXtreme Scale scenario, the eXtreme Scale processes are hosted by WebSphere Application Server processes.

eXtreme Scale embedded data grid configuration

- Use default ObjectGrid configuration
- Specify custom ObjectGrid configuration files

Full path to objectgrid.xml file to copy into configuration
: Specifies the full path to the objectgrid.xml file for the configuration that you want to use.

Full path to objectgriddeployment.xml file to copy into configuration
: Specifies the full path to the objectgriddeployment.xml file for the configuration that you want to use.

Catalog service domain collection

Use this page to manage catalog service domains. Catalog service domains define a group of catalog servers that manage the placement of shards and monitors the health of container servers in your data grid.

To view this administrative console page, click System administration > WebSphere eXtreme Scale > Catalog service domains. To create a new catalog service domain, click New. To delete a catalog service domain, select the catalog service domain you want to remove and click Delete.

Test Connection

When you click the Test connection button, all of the defined catalog service domain end points are queried one by one, if any one end point is available, returns a message that indicates that the connection to the catalog service domain was successful. You can use this button to test that you have configured the connection and security information correctly.

Set Default

Defines the catalog service domain that is used as the default. Select one catalog service domain as the default and click Set default. Only one catalog service domain can be selected as the default.

Name

Specifies the name for the catalog service domain.

Default

Specifies which catalog service domain in the list is the default. The default catalog service domain is indicated with the following icon:  .

Catalog service domain settings

Use this page to manage the settings for a specific catalog service domain. Catalog service domains define a group of catalog servers that manage the placement of shards and monitors the health of container servers in your data grid. You can define a catalog service domain that is in the same cell as your deployment manager. You can also define remote catalog service domains if your WebSphere® eXtreme Scale configuration is in a different cell or your data grid is made up of Java™ SE processes.

To view this administrative console page, click System administration > WebSphere eXtreme Scale > Catalog service domains > *catalog_service_domain_name*.

Related tasks:
Configuring eXtreme Scale connection factories
Configuring catalog servers and catalog service domains
Related reference:
Client properties file

Test connection

When you click the Test connection button, all of the defined catalog service domain end points are queried one by one, if any one end point is available, returns a message that indicates that the connection to the catalog service domain was successful. You can use this button to test that you have configured the connection and security information correctly.

Name

Specifies the name of the catalog service domain.

Enable this catalog service domain as the default unless another catalog service domain is explicitly specified

If you select this check box, the selected catalog service domain becomes the default catalog service domain for the cell. Each server profile in the cell that is augmented with the WebSphere eXtreme Scale profile belongs to the selected catalog service domain.

For WebSphere eXtreme Scale, all eXtreme Scale containers that are embedded in Java EE application modules connect to the default domain. Clients can connect to the default domain using the ServerFactory.getServerProperties().getCatalogServiceBootstrap() API to retrieve the catalog service endpoints to use when calling the ObjectGridManager.connect() API.

If you change the default domain to point to a different set of catalog servers, then all containers and clients refer to the new domain after they are restarted.

Catalog servers

Specifies a list of catalog servers that belong to this catalog service domain.

Click New to add a catalog server to the list. This catalog server must already exist in the eXtreme Scale configuration. You can also edit or delete a server from the list by selecting the endpoint and then clicking Edit or Delete. Define the following properties for each catalog server endpoint:

Catalog server endpoint
: Specifies the name of the existing application server or remote server on which the catalog service is running. A catalog service domain cannot contain a mix of existing application servers and remote server endpoints.

  - Existing application server: Specifies the path of an application server, node agent, or deployment manager in the cell. A catalog service starts automatically in the selected server. Select from the list of the existing application servers. All of the application servers that you define within the catalog service domain must be in the same core group.
  - Remote server: Specifies the host name of the remote catalog server.
    For WebSphere eXtreme Scale remote endpoints: Specifies the host name of the remote catalog server process. You must start the remote servers with the **startOgServer** script or the embedded server API.

Client Port
: Specifies the port that is used for communication between the catalog servers in the catalog service domain. This value is required for catalog servers that are running in WebSphere Application Server processes. You can set the value to any port that is not being used by another process.

Listener Port
: Specifies the port that is used for communication with clients. This value is required for remote endpoints and must match the value used when the catalog service was started. The listener port is used by clients and containers to communicate with the catalog service.
For WebSphere eXtreme Scale remote endpoints: Defines the Object Request Broker (ORB) listener port for containers and clients to communicate with the catalog service through the ORB. For WebSphere Application Server endpoints, the listener port value is inherited from the BOOTSTRAP_ADDRESS port configuration.

Status

Table 1. Catalog server endpoint status

Icon	Definition
	Unknown
	Started
	Stopped

Client security properties

Use this page to configure client security for a catalog service domain. These settings apply to all the servers in your catalog service domain. These properties can be overridden by specifying a splicer.properties file with the com.ibm.websphere.xs.sessionFilterProps custom property or by splicing the application EAR file.

To view this administrative console page, click System administration > WebSphere eXtreme Scale > Catalog service domains > *catalog_service_domain_name* > Client security properties.

Enable client security

Specifies that client security is enabled for the catalog server. The server properties file that is associated with the selected catalog server must have a matching securityEnabled setting in the server properties file. If these settings do not match, an exception results.

Credential authentication

Indicates if credential authentication is enforced or supported.

Never
: No client credential authentication is enforced.

Required
: Credential authentication is always enforced. If the server does not support credential authentication, the client cannot to connect to the server.

Supported
: Credential authentication is enforced only if both the client and server support credential authentication.

Authentication retry count

Specifies the number of times that authentication gets tried again if the credential is expired.

If you do not want to try authentication again, set the value to `0`.

Credential generator class

Indicates the com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator implementation class, so the client retrieves the credential from the CredentialGenerator object.

You can choose from two predefined credential generator classes, or you can specify a custom credential generator. If you choose a custom credential generator, you must indicate the name of the credential generator class.

- com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredentialGenerator
- com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredentialGenerator

- Custom credential generator

Subject type

Specifies if you are using the J2EE caller or the J2EE runAs subject type. You must specify this value when you choose the WSTokenCredentialGenerator credential generator.

- **runAs**: The subject contains the principal of the J2EE run as identity and the J2EE run as credential.
- **caller**: The subject contains the principal of the J2EE caller and the J2EE caller credential.

User ID

Specify a user ID when you are using the UserPasswordCredentialGenerator credential generator implementation.

Password

Specify a password when you are using the UserPasswordCredentialGenerator credential generator implementation.

Credential generator properties

Specifies the properties for a custom CredentialGenerator implementation class. The properties are set in the object with the setProperties(String) method. The credential generator properties value is used only when a value is specified for the Credential generator class field.

Catalog service domain custom properties

You can further edit the configuration of the catalog service domain by defining custom properties.

To view this administrative console page, click System administration > WebSphere eXtreme Scale > Catalog service domains > Custom properties. To create a new custom property, click New.

Name

Specifies the name of the custom property for the catalog service domain.

Value

Specifies a value for the custom property for the catalog service domain.

Glossary

This glossary includes terms and definitions for various IBM business process management products.

The following cross-references are used in this glossary:

- See refers you from a term to a preferred synonym, or from an acronym or abbreviation to the defined full form.
- See also refers you to a related or contrasting term.

To view glossaries for other IBM products, go to www.ibm.com/software/globalization/terminology (opens in new window).

A

A2A
: See application to application.

abstract schema
: Part of the deployment descriptor for an entity bean that is used to define the bean relationships, persistent fields, or query statements.

abstract test
: A component or unit test that is used to test Java interfaces, abstract classes, and superclasses; that cannot be run on its own; and that does not include a test suite. See also component test.

abstract type
: A type that can never be instantiated and whose members are exposed only in instances of concrete types that are derived from it.

Abstract Window Toolkit (AWT)
: In Java programming, a collection of GUI components that were implemented using native-platform versions of the components. These components provide that subset of functionality which is common to all operating system environments. (Sun) See also Standard Widget Toolkit, Swing Set.

access bean
: An enterprise bean wrapper that is typically used by client programs, such as JSP files and servlets. Access beans hide the complexity of using enterprise beans and improve the performance of reading and writing multiple EJB properties.

access control

In computer security, the process of ensuring that users can access only those resources of a computer system for which they are authorized.

access control list (ACL)
In computer security, a list associated with an object that identifies all the subjects that can access the object and their access rights.

access ID
The unique identification of a user used during authorization to determine if access is permitted to the resource.

access intent
Metadata that optimizes and controls the runtime behavior of an entity bean with respect to concurrency control, resource management, and database access strategies.

access intent policy
A grouping of access intents that governs a type of data access pattern for enterprise bean persistence.

accessor
In computer security, an object that uses a resource. Users and groups are accessors.

access point group
A collection of core groups that defines the set of core groups in the same cell or in different cells that communicate with each other.

access privilege
An authority that relates to a request for a type of access to data.

access right
See permission.

accounting
The process of collecting and reporting information about the use of services to apportion cost.

ACID transaction
A transaction involving multiple resource managers using the two-phase commit process to ensure atomic, consistent, isolated, and durable (ACID) properties.

ACL
See access control list.

action

1. An activity that is run on a transition or a transaction. See also processing action.
2. In a business rule, the event that results from the evaluation of the condition.
3. A business process that is generated in response to the processing of an event or a rule.
4. A series of processing steps, such as document validation and transformation.

Action class
In Struts, the superclass of all action classes.

action column
The action part of a decision table.

action mapping
A Struts configuration file entry that associates an action name with an Action class, a form bean, and a local forward.

action object
A subset of fields in the definition of an action.

action phrase
In the vocabulary, a phrase that specifies an action to be executed. An action phrase corresponds to a method that has no return value in the business object model (BOM).

action rule

1. A rule in which the action is always performed. See also if-then rule, rule set.
2. A business rule that can be edited in the rule editor. Action rules, decision tables, and decision trees are different representations of business rules.

action rule template
A partly completed action rule that can be used to create a series of rules with the same structure.

action service
A service that triggers a process or notification to inform users about a situation.

action service handler
An entity that is responsible for the invocation mechanism of one or more action services.

action set

1. The leaf of a branch in a decision tree. Action sets consist of one or more actions to be carried out when the conditions defined in the rule are met.
2. In Eclipse, a group of commands that a perspective contributes to the main toolbar and menu bar.

action task
In a rule flow, a task that contains rule action statements. These action statements are executed each time the task is called.

activation
In Java, the process of transferring an enterprise bean from secondary storage to memory. (Sun) See also passivation.

activation condition
A Boolean expression in a node within a business process that specifies when processing is to begin.

active change set
A change that is in the Draft, Pending, or Approved state.

active option set
In an option set group, the option set that a new scenario uses or that a scenario in progress switches to, if switching becomes necessary.

active site analytics
The instrumentation of pages with metadata that is embedded in themes and skins to provide data to website analytics and search engine optimization tools.

activity

1. An action designed to achieve a particular business process. An activity is performed on a set of targets on a specific schedule.
2. An element of a process, such as a task, a subprocess, a loop, or a decision. Activities are represented as nodes in process diagrams.
3. A unit of work or a building block that performs a specific, discrete task. See also task.

4. Work that a company or organization performs using business processes. An activity can be atomic or non-atomic (compound). The types of activities that are a part of a process model are process, subprocess, and task.
5. A logical unit of work that can be completed by a person or a system while the process runs.

Activity Decision Flow (ADF)
: The format in which models are exported from WebSphere Business Integration Workbench into WebSphere Business Modeler.

actuator
: A device that causes mechanical motion.

adapter
: An intermediary software component that allows two other software components to communicate with one another.

adapter foundation classes (AFC)
: A common set of services for all resource adapters. The adapter foundation classes conform to, and extend, the Java 2 Connector Architecture JCA 1.5 specification.

adapter object
: An object used in the TX Programming Interface that represents a resource adapter.

Address Resolution Protocol (ARP)
: A protocol that dynamically maps an IP address to a network adapter address in a local area network.

ADF
: See Activity Decision Flow.

ad hoc start event
: An event that is triggered by a user's interaction with the process, such as through the process portal. The ad hoc start event requires an active process to be triggered. See also start event.

administrative agent
: A program that provides administrative support without requiring a direct connection to a database.

administrator
: A person responsible for administrative tasks such as access authorization and content management. Administrators can also grant levels of authority to users.

Advanced Integration service
: A service that represents and interacts with a corresponding service in Integration Designer. See also integration service, service.

Advanced Program-to-Program Communication (APPC)
: An implementation of the SNA LU 6.2 protocol that allows interconnected systems to communicate and share the processing of programs.

AFC
: See adapter foundation classes.

after-image
: A business object that contains all of the entity data after changes have been made to it during an update operation. An after-image contains the complete business object rather than only the primary key and those elements that were changed. See also delta business object.

agenda
: A logical workspace where rule instances that have conditions matching objects in the working memory are put.

agent
: A process that performs an action on behalf of a user or other program without user intervention or on a regular schedule, and reports the results back to the user or program.

aggregate interface
: The logical grouping of ethernet interfaces, connected to the same subnet, that provide higher levels of availability and bandwidth from the networking substrate. See also link aggregation.

aggregate metric
: A metric that is calculated by finding the average, maximum, minimum, sum, or number of occurrences of an instance metric across multiple runs of a process. Examples of aggregate metrics are an average order amount, a maximum order amount, a minimum order amount, the total order amount, or the number of occurrences of $500 for an order amount. See also measure, metric.

aggregation
: The structured collection of data objects for subsequent presentation within a portal.

alarm listener
: A type of asynchronous bean that is called when a high-speed transient alarm expires.

alert
: A message or other indication that signals an event or an impending event.

algorithm mapping
: A process by which service providers can define the mapping of Uniform Resource Identifier (URI) algorithms to cryptographic algorithms that are used for XML digital signature and XML encryption.

alias
: An assumed or actual association between two data entities, or between a data entity and a pointer.

annotate
: To add metadata to an object to describe services and data.

annotation
: An added descriptive comment or explanatory note.

anonymous user
: A user who does not use a valid user ID and password to log into a site. See also authenticated user, registered user.

AP
: See application program.

APAR
: See authorized program analysis report.

API
: See application programming interface.

API content model
: A model that describes how XML documents, and their extended metadata, are represented.

API stub
: A piece of glue code that enables the binder to resolve zRule Execution Server for z/OS API calls that COBOL applications make. For example, HBRBSTUB is an API stub that is used for COBOL applications that run as batch applications and HBRCSTUB is an API stub that is used for COBOL applications that

run as CICS applications. See also glue code.

APPC

See Advanced Program-to-Program Communication.

applet

A program that performs a specific task and is typically portable between operating systems. Often written in Java, applets can be downloaded from the Internet and run in a web browser.

applet client

A client that runs within a browser-based Java runtime environment, and is capable of interacting with enterprise beans directly instead of indirectly through a servlet.

appliance

A drop-in network device, including hardware and firmware, that simplifies IT deployment for a specific set of business requirements.

application

One or more computer programs or software components that provide a function in direct support of a specific business process or processes. See also application server.

application assembly

The process of creating an enterprise archive (EAR) file containing all the files related to an application as well as an Extensible Markup Language (XML) deployment descriptor for the application.

application client

In Java EE, a first-tier client component that runs in its own Java virtual machine. Application clients have access to some Java EE platform APIs, for example JNDI, JDBC, RMI-IIOP, and JMS. (Sun)

application client module

A Java archive (JAR) file that contains a client that accesses a Java application. The Java application runs inside a client container and can connect to remote or client-side Java EE resources.

Application Client project

A structure and hierarchy of folders and files that contain a first-tier client component that runs in its own Java virtual machine.

application delivery notification

A delivery notification that is passed to an application. Typically, an application delivery notification is based on a network delivery notification, but has been modified in some way by the service that exchanges data directly with the application. See also network delivery notification.

application edition

A unique deployment of a particular application. Multiple editions of the same application have the same application name, while edition names are unique.

application edition manager

An autonomic manager that manages interruption-free production application deployments.

application infrastructure virtualization

The pool of application server resources that separates applications from the physical infrastructure on which they run. As a result, workload can be dynamically placed and migrated across the application server pool.

application LT

A logical terminal (LT) that is used by one or more applications, but that is not used for LT sessions.

application placement controller

An autonomic manager that can start and stop application instances on servers to meet the fluctuating demand of work requests and varying service policy definitions.

application policy

A collection of policies and attributes governing access to applications.

application program (AP)

A complete, self-contained program, such as a text editor or a web browser, that performs a specific task for the user, in contrast to system software, such as the operating system kernel, server processes, and program libraries.

application programming interface (API)

An interface that allows an application program that is written in a high-level language to use specific data or functions of the operating system or another program.

Application Response Measurement (ARM)

An application programming interface (API), developed by a group of technology vendors, that can be used to monitor the availability and performance of business transactions within and across diverse applications and systems.

Application Response Measurement agent (ARM agent)

An agent that monitors software that is implemented using the Application Response Measurement standard.

application server

A server program in a distributed network that provides the execution environment for an application program. See also application.

application-specific component

The component of a connector that contains code tailored to a particular application or technology. The application-specific component can respond to requests and implement an event-notification mechanism that detects and responds to events initiated by an application or external programmatic entity.

application-specific information

Part of the metadata of a business object that enables the connector to interact with its application (for example, Ariba Buyer) or a data source (for example, a web servlet). See also metadata.

application to application (A2A)

A data transformation from the output of one application to the input of another application.

application virtualization

The separation of an application from the underlying operating environment, which improves portability, compatibility, and manageability of the application.

area

A representation of the physical space within the location to be monitored. Areas are the container for all zones. See also location.

ARFM

See autonomic request flow manager.

ARM

1. See Application Response Measurement.
2. See automatic restart manager.

ARM agent
See Application Response Measurement agent.
ARP
See Address Resolution Protocol.
artifact

1. A graphical object that provides supporting information about the process or elements within the process without directly affecting the semantics of the process.
2. An entity that is used or produced by a software development process. Examples of artifacts are models, source files, scripts, and binary executable files.

aspect-oriented connectivity
A form of connectivity that implements or enforces cross-cutting aspects in service-oriented architecture (SOA), such as security, management, logging, and auditing, by removing such aspects from the concern of the service requesters and providers.
assertion

1. A concept in the meta-model that is used to specify a policy requirement and evaluating endpoints at run time. An assertion is also used to describe the capabilities of an endpoint.
2. A logical expression specifying a program state that must exist or a set of conditions that program variables must satisfy at a particular point during program execution.

asset
A collection of artifacts that provide a solution to a specific business problem. Assets can have relationships and variability or extension points to other assets.
assisted lifecycle server
A representation of a server that is created outside of the administrative domain but can be managed in the administrative console.
assistive technology
Hardware or software that is used to increase, maintain, or assist the functional capabilities of people with disabilities.
associated type
An object that refers to a source object. See also referenced type.
association

1. In enterprise beans, a relationship that exists between two container-managed persistence (CMP) entity beans. There are two types of association: one-to-one and one-to-many.
2. For XML documents, the linkage of the document itself to the rules that govern its structure, which might be defined by a Document Type Definition (DTD) or an XML schema.
3. A connecting object that is used to link information and artifacts with flow objects. An association is represented as a dotted graphical line with an arrowhead to represent the direction of flow.

asymmetric algorithm
See public key algorithm.
asymmetric cryptography
See public key cryptography.
asynchronous
Pertaining to events that are not synchronized in time or do not occur in regular or predictable time intervals.
asynchronous bean
A Java object or an enterprise bean that a Java Platform, Enterprise Edition (Java EE) application can run asynchronously.
asynchronous messaging
A method of communication between programs in which a program places a message on a message queue, then proceeds with its own processing without waiting for a reply to its message.
asynchronous replica
A shard that receives updates after the transaction commits. This method is faster than a synchronous replica, but introduces the possibility of data loss because the asynchronous replica can be several transactions behind the primary shard. See also synchronous replica.
atomic activity
An activity that is not broken down to a finer level of process model detail. It is a leaf in the tree-structure hierarchy of process activities.
attribute

1. A set of factors that are used as variables to determine the score of an entity. The value of an attribute can be a natural number, a floating point number, a Boolean value, a character, or a character string. An attribute can be the result of the execution of another rule or a combination of other attributes.
2. A characteristic or trait of an entity that describes the entity; for example, the telephone number of an employee is one of the employee attributes. See also entity, identity.
3. In markup languages such as SGML, XML, and HTML, a name-value pair within a tagged element that modifies features of the element.
4. A property, quality, or characteristic whose value contributes to the specification of an element or program function. For example, "cost" or "location" are attributes that can be assigned to a resource.

attribute list
A linked list that contains extended information that is used to make authorization decisions. Attribute lists consist of a set of name = value pairs.
audit log
A log file containing a record of system events and responses.
augment
To convert a profile to another kind of profile. For example, a server profile can be modified to become a bus profile. See also unaugment.
authenticated user
A portal user who has logged in to the portal with a valid account (user ID and password). Authenticated users have access to all public places. See also anonymous user, registered user.
authentication

A security service that provides proof that a user of a computer system is genuinely who that person claims to be. Common mechanisms for implementing this service are passwords and digital signatures.

authentication alias
An alias that authorizes access to resource adapters and data sources. An authentication alias contains authentication data, including a user ID and password.

authenticator key
A set of alphanumeric characters used for the authentication of a message sent via the SWIFT network.

authorisation
A document that authorizes one SWIFTNet destination to send messages to or receive messages from another SWIFTNet destination.

authorization

1. The process of granting a user, system, or process either complete or restricted access to an object, resource, or function.
2. In computer security, the right granted to a user to communicate with or make use of a computer system.

authorization policy
A policy whose policy target is a business service and whose contract contains one or more assertions that grant permission to run a channel action.

authorization table
A table that contains the role to user or group mapping information that identifies the permitted access of a client to a particular resource.

authorized program analysis report (APAR)
A request for correction of a defect in a supported release of a program supplied by IBM.

autodiscovery
The discovery of service artifacts in a file system, external registry, or another source.

automatic application installation project
A monitored directory to which the addition of a fully composed EAR, WAR, EJB JAR, or stand-alone RAR file triggers automatic deployment and publication to a target server. Deletion of an EAR or Java EE module file from this directory triggers automatic uninstalling. See also monitored directory.

automatic restart management
The facilities that detect failures and manage server restarts.

automatic restart manager (ARM)
A z/OS recovery function that can automatically restart batch jobs and started tasks after they or the system on which they are running end unexpectedly.

automatic transition
A transition that occurs on completion of the activity within the originating state.

automatic variable
A variable that a user can declare as an instance of a specific business object model (BOM) class.

autonomic request flow manager (ARFM)
An autonomic manager that controls request prioritization in the on-demand router.

availability

1. The time periods during which a resource is accessible. For example, a contractor might have an availability of 9 AM to 5 PM every weekday, and 9 AM to 3 PM on Saturdays.
2. The condition allowing users to access and use their applications and data.

AWT
See Abstract Window Toolkit.

Axis
An implementation of SOAP on which Java web services can be implemented.

B

B2B
See business-to-business.

B2C
See business-to-consumer.

B2E
See business-to-employee.

BA
See basic authentication.

back-end system
An IMS in a multisystem environment that accepts transactions from the front-end system, calls application programs for transaction processing, and routes replies back to the front-end system for response to the terminal.

background processing
A mode of program execution in which the shell does not wait for program completion before prompting the user for another command.

BAL
See Business Action Language.

BAM
See business activity monitoring.

bank identifier code (BIC)
A code used to uniquely identify a bank, logical terminal, or branch within a SWIFT network.

BAPI
See Business Application Programming Interface.

base
The core product, for which features can be separately ordered and installed.

base classes
See adapter foundation classes.

base configuration
The part of a storage management subsystem (SMS) configuration that contains general storage management attributes, such as the default management class, default unit, and default device geometry. It also identifies the systems, system groups, or both the systems and system groups that an SMS

configuration manages.

baseline
: See snapshot.

base object
: An object that defines a common set of attributes; more complex objects are then built from a base object, inheriting the common attribute set.

basic analysis
: A type of analysis that displays a report for the values of one or more business measures during a specific period of time.

basic authentication (BA)
: An authentication method that uses a user name and a password.

basic type
: A type whose values have no identity (that is, they are pure values). Basic types include Integer, Boolean, and Text.

batch application
: An application that is implemented as part of a bundle or Java archive file and deployed as an archive file.

batch job
: A predefined group of processing actions submitted to the system to be performed with little or no interaction between the user and the system.

bean
: A definition or instance of a JavaBeans component. See also enterprise bean, JavaBeans.

bean class
: In Enterprise JavaBeans (EJB) programming, a Java class that implements a javax.ejb.EntityBean class or javax.ejb.SessionBean class.

bean-managed messaging
: A function of asynchronous messaging that gives an enterprise bean complete control over the messaging infrastructure.

bean-managed persistence (BMP)
: The mechanism whereby data transfer between an entity bean's variables and a resource manager is managed by the entity bean. (Sun) See also container-managed persistence.

bean-managed transaction (BMT)
: The capability of the session bean, servlet, or application client component to manage its own transactions directly, instead of through a container.

bearer token
: A Security Assertion Markup Language (SAML) token that uses the bearer subject confirmation method. In a bearer subject confirmation method, a sender of SOAP messages is not required to establish correspondence that binds a SAML token with contents of the containing SOAP message.

BEL
: See Business Events Language.

BIC
: See bank identifier code.

bidi
: See bidirectional.

bidirectional (bidi)
: Pertaining to scripts such as Arabic and Hebrew that generally run from right to left, except for numbers, which run from left to right.

big endian
: A format for storage or transmission of binary data in which the most significant value is placed first. See also little endian.

binary format
: Representation of a decimal value in which each field must be 2 or 4 bytes long. The sign (+ or -) is in the far left bit of the field, and the number value is in the remaining bits of the field. Positive numbers have a 0 in the sign bit and are in true form. Negative numbers have a 1 in the sign bit and are in twos complement form.

binary large object (BLOB)
: A block of bytes of data (for example, the body of a message) that has no discernible meaning, but is treated as one entity that cannot be interpreted.

bind
: To establish a connection between software components on a network using an agreed-to protocol. In web services, the bind operation occurs when the service requester invokes or initiates an interaction with the service at run time using the binding details in the service description to locate, contact, and invoke the service.

binding
: A temporary association between a client and both an object and a server that exports an interface to the object. A binding is meaningful only to the program that sets it and is represented by a bound handle.

black box
: A pool in which no content can be seen.

BLOB
: See binary large object.

block decryption
: Symmetric algorithms that decrypt a block of data at one time.

block encryption
: Symmetric algorithms that encrypt a block of data at one time.

BMP
: See bean-managed persistence.

BMT
: See bean-managed transaction.

BOM
: See business object model.

BOM property
: A property added to a type in a business object model (BOM). Business properties extend the original type without altering its source.

BOM-to-XOM mapping
: A mechanism that defines how business elements are mapped to the Execution Object Model.

Boolean
: Characteristic of an expression or variable that can only have a value of true or false.

boot BOM
: A set of files that define the system types, such as string or number, for the Business Action Language (BAL).

bootstrap

A small program that starts a computer by loading the operating system and other basic software.

bootstrap authorisation
: An authorization that has been recorded but not yet processed by an relationship management application (RMA).

bootstrap member
: An application server or cluster that is configured to accept application initialization requests into the service integration bus. The bootstrap member authenticates the request and directs the connection request to a bus member.

bootstrap period
: The period during which relationship management (RM) data is recorded and converted into authorization records.

bootstrapping
: The process by which an initial reference of the naming service is obtained. The bootstrap setting and the host name form the initial context for Java Naming and Directory Interface (JNDI) references.

bottleneck
: A place in the system where contention for a resource is affecting performance.

bottom-up development
: In web services, the process of developing a service from an existing artifact such as JavaBeans or an enterprise bean rather than a Web Services Description Language (WSDL) file. See also top-down development.

bottom-up mapping
: In Enterprise JavaBeans (EJB) programming, an approach for mapping enterprise beans to database tables, in which the schema is first imported from an existing database and then enterprise beans and mappings are generated.

boundary event
: An intermediate event that is attached to the boundary of an activity. A boundary event can be triggered only while the activity is running, either leaving the activity running or interrupting the activity.

boundary zone
: A zone that is used for implementing access control to areas that are not covered by event devices and therefore cannot be controlled completely or directly.

bound component
: In the Type Designer, a component for which each occurrence of the data can be identified without considering the context in which that occurrence is placed.

bound type
: In the Type Designer, a type whose data object can be identified without considering the context in which that data object is placed.

BPD
: See business process definition.

BPEL
: See Business Process Execution Language.

BPM
: See business process management.

BPML
: See Business Process Modeling Language.

BPML activity
: A step in a business process that provides directions for how data should be handled.

BPMN
: See Business Process Modeling Notation.

branch
: 1. A distinct path leading to or originating from an element in a process model or UML diagram.
  2. In the CVS team development environment, a separate line of development where changes can be isolated. When a programmer changes files on a branch, those changes do not appear on the main trunk or other branches.
  3. In a rule flow, a node that organizes conditional transitions. Several transitions can go to and from a branch node. All transitions created from a branch must have a condition, except the Else transition.

breadcrumb trail
: A navigation technique used in a user interface to give users a way to keep track of their location within the program or documents.

breakpoint
: A marked point in a process or programmatic flow that causes that flow to pause when the point is reached, typically to allow debugging or monitoring.

bridge
: In the connection of local loops, channels, or rings, the equipment and techniques used to match circuits and to facilitate accurate data transmission. See also web application bridge.

bridge interface
: A node and a server that run a core group bridge service.

BRLDF
: See Business Rule Language Definition Framework.

BRMS
: See business rule management system.

broker archive
: A file that is the unit of deployment to the broker that can contain any number of compiled message flow and message set files and a single deployment descriptor. A separate broker archive file is required for each configuration that is deployed.

brokerlist section
: A section in a customization definition document (CDD) that describes which BAR files are deployed, which execution group and broker the files are deployed to, and which tuning parameters the files use.

broker topology definition (BTD)
: A description of the brokers, execution groups, and broker archive (BAR) files that are used in a runtime environment, and the actions that are required to implement the current broker topology (for example, deploying the BAR files for a new service).

broker topology definition document (BTDD)
: An XML document that describes a broker topology definition.

browser
: A client program that initiates requests to a web server and displays the information that the server returns.

brute force collision
: A programming style that relies on computing power to try all the possibilities with a known hash until the solution is found.

BTD
: See broker topology definition.

BTDD
: See broker topology definition document.

bucket
: One or more fields that accumulate the result of an operation.

build
: To create or modify resources, typically based on the state of other resources. A Java builder converts Java source files into executable class files, for example, and a web link builder updates links to files whose name or location has changed.

build definition file
: An XML file that identifies components and characteristics for a customized installation package (CIP).

build path
: The path that is used during compilation of Java source code, in order to find referenced classes that reside in other projects.

build plan
: An XML file that defines the processing necessary to build generation outputs and that specifies the machine where processing takes place.

build time data
: Objects that are not used by the translator, such as EDI standards, record oriented data document types, and maps.

bulk decryption
: See block decryption.

bulk encryption
: See block encryption.

bulk resource
: A resource that is taken in quantity from a pool of generic resources. For example, a task might require 10 landscapers or 10 liters of water.

bundle
: 1. In the OSGi service platform, a Java archive 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. See also bundle cache, bundle repository, enterprise bundle archive.
  2. A set of tokens that are transferred between nodes in a simulation as a complete group.

bundle cache
: A cell-wide store, or server-wide store for single-server systems, of bundles that OSGi applications refer to and that have been downloaded from both internal and external repositories. See also bundle, bundle repository.

bundle repository
: A common store of OSGi bundles that can be shared by multiple OSGi applications. See also bundle, bundle cache.

bus
: Interconnecting messaging engines that manage bus resources.

Business Action Language (BAL)
: A business rule language that uses an intuitive and natural language-like syntax for writing business rules.

business activity monitoring (BAM)
: The collection and presentation of real-time information that describes a business process or a series of activities spanning multiple systems and applications.

business analyst
: 1. A decision management user role that is responsible for modeling rule application projects.
  2. A specialist who analyzes business needs and problems, consults with users and stakeholders to identify opportunities for improving business return through information technology, and transforms requirements into a technical form.

Business Application Programming Interface (BAPI)
: A programming interface that is used to access SAP databases from with SAP or other development platforms. BAPI is used to achieve integration between the R/3 System and external applications and legacy systems.

business calendar
: A calendar that is used to model noncontiguous time intervals (intervals that do not proceed in a sequential manner). For example, a business calendar that defines regular working hours might refer to the non-overtime regular working hours of Monday to Friday, 9:00 a.m. to 5:00 p.m.

business component
: A component that defines the structure, behavior, and information displayed by a particular subject, such as a product, contact, or account, in Siebel Business Applications.

business ecosystem
: A business community supported by a foundation of interacting organizations and individuals. This community produces goods and services of value to customers, who are themselves members of the ecosystem. A business ecosystem contains business services networks, which contain business process, relevant to the transactions in that network.

business event
: 1. A significant occurrence in a business process, generally identified by a business analyst, that warrants monitoring over time to reveal a key performance indicator (KPI).
  2. An event that occurs during a business process.

Business Event Language
: A business rule language that expresses event rules. See also business rule language.

Business Events Language (BEL)
: A business rule language that uses an intuitive and natural language-like syntax for writing event rules.

business graph
: A wrapper that is added around a simple business object or a hierarchy of business objects to provide additional capabilities, such as carrying change summary and event summary information related to the business objects in the business graph. See also business object.

business group

A place to collect any elements to group together. Different business groups can be created for companies, processes, parts of processes, or any other grouping.

business integration system
: An integration broker and a set of integration adapters that allow heterogeneous business applications to exchange data through the coordinated transfer of information in the form of business objects.

business item
: A business document, work product, or commodity that is used in business operations. Examples of business items are a manufacturing order, system board, power supply, and memory chip (in a PC assembly process), itinerary and customer information record (in a trip reservation process), and passenger (in a transportation process). See also business object.

business item instance
: A particular occurrence or example of a business item. If there is a business item called Invoice, then an example of a business item instance would be "Invoice #1473.

business item template
: A category used to model a group of business items that share common properties. After these properties are defined in the template, they are inherited by all business items using the template. For example, an organization may define a number of forms to be used in human resource processes, all of which have fields for date, employee number, HR form number, and HR administrator.

business logic tier
: The set of components that reside between the presentation and database tiers. This logic tier hosts the enterprise bean containers, which run the business logic.

business measure
: A description of a performance management characteristic that you want to monitor. Business measures include instance metrics, aggregate metrics (also called measures), and key performance indicators (KPI).

business method

1. A method of an enterprise bean that implements the business logic or rules of an application. (Sun)
2. A method added to a type in a business object model. Business methods extend the original type without altering its source.

business object

1. A software entity that represents a business entity, such as an invoice. A business object includes persistent and nonpersistent attributes, actions that can be performed on the business object, and rules that the business object is governed by. See also binding, business graph, business item, data object, private business object, Service Data Objects.
2. An abstract representation of the fields that belong to the event and action definitions.

business objective
: A high-level business goal. Because business objectives are typically abstract, they are difficult to measure and are therefore translated into more measurable lower-level business goals.

business object map
: An artifact that assigns values to the target business objects based on the values in the source business objects.

business object model (BOM)

1. A representation of the core concepts of a business and their logical connections. The business object model is the basis for the vocabulary used in business rules. The elements of a business object model (BOM) map to those of a corresponding execution object model.
2. A model that defines how a system organizes its processes when interacting with business objects. An example of a business object model is the Enterprise JavaBeans (EJB) component model.

business operations
: The ways in which an organization operates, including its processes and organizational structure. For example, an organization might have a management structure and processes defined for everything from taking vacation days to submitting travel expenses.

business policy

1. A set of rules that define business processes, industry practices, or the scope and characteristics of business offerings.
2. A policy that is attached to an object in the ontology known as the business policy target. It optionally specifies a set of conditions that must be met for the business policy to apply. The policies declare a set of assertions that must be satisfied when the conditions are met.

business policy target
: An object in the ontology suitable for attaching business policies.

business process
: A defined set of business activities that represent the required steps to achieve a business objective. A business process includes the flow and use of information and resources.

business process container
: A process engine that contains process modules.

business process definition (BPD)
: A reusable model of a process that defines the common aspects of all runtime instances of that process model.

Business Process Execution Language (BPEL)
: An XML-based language for the formal specification of business processes and business interaction protocols. BPEL extends the web services interaction model and enables it to support business transactions.

business process management (BPM)
: The services and tools that support process management (for example, process analysis, definition, processing, monitoring and administration), including support for human and application-level interaction. BPM tools can eliminate manual processes and automate the routing of requests between departments and applications.

Business Process Modeling Language (BPML)
: An XML-based language that describes business processes designed by the Business Process Management Initiative (www.bpmi.org).

Business Process Modeling Notation (BPMN)
: A standardized graphical notation for creating diagrams of business processes.

business protocol

A set of rules and instructions (protocol) used to format and transmit information across a computer network. Examples include RosettaNet, cXML, and EDI-X12.

business rule

1. A policy, constraint, or required operation that applies to a specific set of business conditions or dependencies. An example of a business rule for a bank is that a credit check is not required when opening an account for an existing customer.
2. A representation of how business policies or practices apply to a business activity.

business rule application
: An application in which a business policy has been implemented by using business rules.

business rule group
: A set of scheduled business rules that are available as a service that can be invoked. The business rule group also provides the organizational structure for managing the set of business rules.

business rule language
: A language for expressing rules with natural language terms and syntax. See also Business Event Language.

Business Rule Language Definition Framework (BRLDF)
: A framework for defining custom business rule languages, using XML schemas and property files.

business rule management
: The practices that control and manage business rules through their life cycles.

business rule management system (BRMS)
: A system designed to modify and manage business logic independently from the applications within an organization.

business service
: An abstract representation of a business function, hiding the specifics of the function interfaces.

business service definition
: A representation of the WSDL PortTypes in a business service. A business service definition describes a specific set of business service operations that are used to perform related business functionality.

business service object
: A representation of an XML schema file (.xsd). There are inline XML schemas and schema types within WSDL files. A business service object is a collection of business service object definitions and business service object templates.

business service object definition
: A representation of the WSDL ComplexType in an inline schema, or the XML schema type (SimpleType, ComplexType, Anonymous ComplexType, or Anonymous SimpleType) in an XML schema file. There are inline XML schemas and schema types within WSDL files. A business service object definition is similar to a business item and is used to define the business data that is required when a business service operation is invoked.

business service operation
: A representation of the WSDL Operation in a business service definition. A business service operation describes a business function and includes the business service object definitions that are required when the operation is invoked. A business service operation also describes the business service object definitions that result from completing the business service operation. For example, a Product Search business service operation requires a Product name (a business service object definition) and returns a Product business service object definition. Business service operations can be added to process diagrams as non-editable services.

business services network
: A collection of business processes, services, subscribers, and policies that enable, control, or consume a portfolio of business services. The business services network can span enterprise boundaries and geographies or be confined to a single physical network or entity.

business situation
: A condition that might require business action. Examples of business situations are a declining sales volume or an unacceptable amount of time to respond to a customer.

business space
: A collection of related web content that conveys insight into the business and gives users the ability to react to changes in the business.

business subtype
: A subtype of a type in a business object model (BOM). Business subtypes are used to extend an object model using business methods and business properties.

business-to-business (B2B)
: Refers to Internet applications that exchange information or run transactions between businesses. See also business-to-consumer.

business-to-consumer (B2C)
: Refers to the subset of Internet applications that exchange information or run transactions between businesses and consumers. See also business-to-business.

business-to-employee (B2E)
: A business model that supports electronic communications between a business and its employees.

bus member
: An application server or server cluster that hosts one or more messaging engines in a service integration bus.

bus topology
: A physical arrangement of application servers, messaging engines and queue managers and the pattern of bus connections and links between them.

bytecode
: Machine-independent code generated by the Java compiler and executed by the Java interpreter. (Sun)

C

C2A
: See Click-to-Action.

CA
: See certificate authority.

cache
: A buffer that contains frequently accessed instructions and data; it is used to reduce access time.

cache instance resource
: A location where any Java Platform, Enterprise Edition (Java EE) application can store, distribute, and share data.

cache replication

The sharing of cache IDs, cache entries, and cache invalidations with other servers in the same replication domain.

calculation strategy
: The strategy used to calculate the final score of a scorecard table.

callback handler
: A mechanism that uses a Java Authentication and Authorization Service (JAAS) interface to pass a security token to the web service security run time for propagation in the web service security header.

callout
: The action of bringing a computer program, a routine, or a subroutine into effect.

callout node
: The connection point in a mediation request flow from which a service message is sent to a target. There must be one callout node for each target operation.

callout response node
: The starting point for a mediation response flow. There must be one callout response node for each target.

call stack
: A list of data elements that is constructed and maintained by the Java virtual machine (JVM) for a program to successfully call and return from a method.

candidate endpoint
: A known service endpoint that implements an interface for a particular request. The set of candidates is then filtered by the dynamic assembler to select the best endpoint out of all the candidates.

capability
: A group of functions and features that can be hidden or revealed to simplify the user interface. Capabilities can be enabled or disabled by changing preference settings, or they can be controlled through an administration interface.

capability list
: A list of associated resources and their corresponding privileges per user.

card
: 1. A Wireless Markup Language (WML) document that provides user-interface and navigational settings to display content on mobile devices. See also deck.
  2. In the Map Designer, a data object. There are two types of map cards: input and output.

cardinality
: The number of elements in a set.

card object
: An object used in the TX Programming Interface that represents an input or output card of a map in program memory.

Cascading Style Sheets (CSS)
: A language that defines a hierarchical set of style rules for controlling the rendering of HTML or XML files in browsers, viewers, or in print.

case
: A process instance and related set of documents, properties, roles, and tasks that are used to facilitate the collaboration of people to achieve a business outcome. See also task.

case property
: A property, such as a name or date, that is defined at the solution level and reused in a case type, document type, task, or step in that solution. See also case type, document type, step, task.

case type
: The definition of a case. The information that a case type provides is similar to that information that a process definition or template provides for a process instance. See also case property.

catalog
: A container that, depending on the container type, holds processes, data, resources, organizations, or reports in the project tree.

catalog service
: A service that controls placement of shards and discovers and monitors the health of containers.

catalog service domain
: A highly available collection of catalog service processes.

catching message intermediate event
: An intermediate event that is triggered when a specific message is received. See also intermediate event.

category
: 1. A classification of elements for documentation or analyses.
  2. A property that is set on an element of the business object model (BOM) and can be applied to business classes and filtered in business rules. This property allows the user to specify whether a business class and its members are visible in a rule.
  3. A container used in a structure diagram to group elements based on a shared attribute or quality.
  4. A type class that is used to organize types in a type tree in the Type Designer. Categories organize types that have common properties.

category filter
: A filter that is set on a business rule and removes the business element to which a category was attached from the completion menu.

CBPDO
: See Custom-built Product Delivery Option.

CBR
: See content based routing.

CBS
: See composite business service.

CCD table
: See consistent-change-data table.

CCI
: See common client interface.

CCSID
: See coded character set identifier.

CDD
: See customization definition document.

CD table
: See change-data table.

CEI
: See Common Event Infrastructure.

CEI event
: An event generated over the Common Event Infrastructure (CEI) and logged in a CEI data store.

CEI target
: An application server or server cluster where the Common Event Infrastructure (CEI) server is enabled.

cell
: 1. A group of managed processes that are federated to the same deployment manager and can include high-availability core groups.
  2. One or more processes that each host runtime components. Each has one or more named core groups.

cell-scoped binding
: A binding scope where the binding is not specific to, and not associated with any node or server. This type of name binding is created under the persistent root context of a cell.

center cell
: The only cell in a star topology with the ability to make autonomic decisions.

centralized installation manager
: A component that remotely installs and uninstalls product and maintenance packages in server environments.

certificate
: In computer security, a digital document that binds a public key to the identity of the certificate owner, thereby enabling the certificate owner to be authenticated. A certificate is issued by a certificate authority and is digitally signed by that authority.

certificate authority (CA)
: A trusted third-party organization or company that issues the digital certificates. The certificate authority typically verifies the identity of the individuals who are granted the unique certificate. See also certificate, Globus certificate service, Secure Sockets Layer.

certificate revocation list (CRL)
: A list of certificates that have been revoked before their scheduled expiration date. Certificate revocation lists are maintained by the certificate authority and used, during a Secure Sockets Layer (SSL) handshake to ensure that the certificates involved have not been revoked.

certificate set
: A set of primary and secondary certificates that can be associated to a participant connection.

certificate signing request (CSR)
: An electronic message that an organization sends to a certificate authority (CA) in order to obtain a certificate. The request includes a public key and is signed with a private key; the CA returns the certificate after signing with its own private key. See also certificate, keystore.

CGI
: See Common Gateway Interface.

chain
: The name of a channel framework connection that contains an endpoint definition.

chameleon schema
: A schema that inherits a target namespace from a schema that includes the chameleon schema.

change basis version
: The version of the server partition that changes were made against.

change-data table (CD table)
: In SQL replication, a replication table on the Capture control server that contains changed data for a replication source table.

change management
: 1. The process of planning for and executing changes to configuration items in the information technology environment. The primary objective of change management is to enable beneficial changes to be made with minimum disruption to services.
  2. The process of planning (for example, scheduling) and controlling (for example, distributing, installing, and tracking) software changes over a network.

change record
: A recorded instance that is created with each write action to the repository. The change record contains metadata about all repository changes (such as who was responsible for a commit action) and can be used as a version history view of the repository.

change set
: A cohesive unit consisting of a number of related changes that need to be made together.

channel
: 1. A communication path through a chain to an endpoint.
  2. A WebSphere MQ object that defines a communication link between two queue managers (message channel) or between a client and a queue manager (MQI channel). See also message channel.

channel action
: A business function that can be issued on a channel. Channel actions are role specific and an authorization policy makes it possible to control which role can perform which action in a channel.

channel framework
: A common model for connection management, thread usage, channel management, and message access within an application server.

character conversion
: The process of changing data from one character coding representation to another.

character encoding
: The mapping from a character (a letter of the alphabet) to a numeric value in a character code set. For example, the ASCII character code set encodes the letter "A" as 65, while the EBCIDIC character set encodes this letter as 43. The character code set contains encodings for all characters in one or more language alphabets.

chart series
: A selection of a category of data that will be represented by a chart in a report. A chart can have multiple chart series to represent multiple types of data.

chassis
: The metal frame in which various electronic components are mounted.

cheat sheet
: An interface that guides users through the wizards and steps required to perform a complex task, and that links to relevant sections of the online help.

check in
: In certain software configuration management (SCM) systems, to copy files back into the repository after changing them.

check out
: In certain software configuration management (SCM) systems, to copy the latest revision of a file from the repository so that it can be modified.

child node
: A node within the scope of another node.

choice type
: A group type with a subclass equal to choice that is used to define a selection from a set of components. A choice type defines a choice group, which is valid when the data matches one of the components in the choice group.

choreography
: An ordered sequence of message exchanges between two or more participants. In a choreography there is no central controller, responsible entity, or observer of the process.

CICS
: An IBM licensed program that provides online transaction-processing services and management for business applications.

CIP
: See customized installation package.

cipher
: A cryptographic algorithm used to encrypt data that is unreadable until converted into plain data with a predefined key.

cipher specifications
: Specifications that indicate the data encryption algorithm and key size to use for secure connections.

circular reference
: A series of objects where the last object refers to the first object, which can cause the series of references to be unusable.

class
: 1. In object-oriented design or programming, a model or template that can be used to create objects with a common definition and common properties, operations, and behavior. An object is an instance of a class.
  2. A basic unit of the classification hierarchy used in the Type Designer. There are three classes: item, group, and category.

class file
: A compiled Java source file.

class hierarchy
: The relationships between classes that share a single inheritance.

classification hierarchy
: The hierarchy of a type tree in the Type Designer. The deeper the subtype, the more specific the data characteristics are. See also compositional hierarchy.

classifier
: A specialized attribute used for grouping and color-coding process elements.

class loader
: Part of the Java virtual machine (JVM) that is responsible for finding and loading class files. A class loader affects the packaging of applications and the runtime behavior of packaged applications deployed on application servers.

class path
: A list of directories and JAR files that contain resource files or Java classes that a program can load dynamically at run time.

cleanup period
: The time period during which a database record that has reached its final state or condition is to remain in the database. After the cleanup period expires for such a record, database cleanup causes the record to be deleted from the database.

CLI
: See command-line interface.

Click-to-Action (C2A)
: A method for implementing cooperative portlets, whereby users can click an icon on a source portlet to transfer data to one or more target portlets. See also cooperative portlets, wire.

client
: A software program or computer that requests services from a server. See also host, server.

client application
: An application, running on a workstation and linked to a client, that gives the application access to queuing services on a server.

client message
: A message from a client application that is to be sent by means of a network to its destination, or a message that is routed to a client application to acknowledge the receipt of a client message by a network.

client project for RuleApps
: A predefined project for Eclipse that contains a class to execute a rule set within a RuleApp.

client proxy
: An object on the client side of a network connection that provides a remote procedure call interface to a service on the server side.

client reroute
: A method that allows a client application, upon the loss of communication with a database server and the predefinition of an alternative server, to continue working with the original database server or the alternative server with only minimal interruption of the work.

client/server
: Pertaining to the model of interaction in distributed data processing in which a program on one computer sends a request to a program on another computer and awaits a response. The requesting program is called a client; the answering program is called a server. See also distributed application.

client type detection
: A process in which a servlet determines the markup language type required by a client and calls the appropriate JavaServer Pages file.

clock event
: A special system event that is used to initiate a system-generated event.

cloud computing
: A computing platform where users can have access to applications or computing resources, as services, from anywhere through their connected devices. A simplified user interface and application programming interface (API) makes the infrastructure supporting such services transparent to users.

cloud group

A collection of hypervisors from a single vendor.

cluster
A group of application servers that collaborate for the purposes of workload balancing and failover.

CMP
See container-managed persistence.

CO
See configuration object.

coach
A user interface that can be created to collect input that is required for an underlying service.

COA report
See confirm-on-arrival report.

coarse-grained
Pertaining to viewing a group of objects from an abstract or high level. See also fine-grained.

cobrowsing
The interaction of multiple users sharing information about their individual web interactions. With this interaction users can share a view of the same web page simultaneously and share further interactions with the web page they are jointly viewing.

code assist
See content assist.

coded character set identifier (CCSID)
A 16-bit number that includes a specific set of encoding scheme identifiers, character set identifiers, code page identifiers, and other information that uniquely identifies the coded graphic-character representation.

code list

1. One or many dynamic pairs of code values that contains sender code and receiver code. Each code pair has one description and up to four additional codes relating to the pair.
2. A table, supplied by Data Interchange Services or defined by the user, that contains all acceptable values for a single data field.

code list table
A repository for lists of codes that can further define fields.

COD report
See confirm-on-delivery report.

coexistence
The ability of two or more entities to function in the same system or network.

coherent cache
Cache that maintains integrity so that all clients see the same data.

cold start
The process of starting an existing data replication configuration without regard for prior replication activity, causing reinitialization of all subscriptions.

collaboration

1. A diagram that shows the exchange of messages between two or more participants in a BPMN model.
2. The ability to connect customers, employees, or business partners to the people and processes in a business or organization, in order to facilitate improved decision-making. Collaboration involves two or more individuals with complementary skills interacting together to resolve a business problem.

collaborative components
UI-neutral API methods and tag libraries that allow developers to add collaborative functionality to their portlets.

collaborative filtering
Personalization technology that calculates the similarity between users based on the behaviors of a number of other people and uses that information to make recommendations for the current user.

collaborative unit
The configuration of the part of a deployment environment that delivers required behavior to an application module. For example, a messaging collaborative unit includes the host of the messaging engine and deployment targets of the application module, and provides messaging support to the application module.

collapsed subprocess
A subprocess that hides its flow details. The collapsed subprocess object has a marker that distinguishes it as a subprocess, rather than a task. The marker is a small square with a plus sign inside.

collection certificate store
A collection of intermediate certificates or certificate revocation lists (CRL) that are used by a certificate path to build up a certificate chain for validation.

collection page
A type of page in the administrative console that displays a collection list of administrative objects. From this type of page, you can typically select objects to act on or to display other pages for.

collective
A set of WebSphere DataPower XC10 appliances that are grouped together for scalability and management purposes.

collision arbiter
A plug-in that specifies how to handle change collisions in map entries.

comma delimited file
A file whose records contain fields that are separated by a comma.

command bean
A proxy that can invoke a single operation using an execute() method.

command line
The blank line on a display where commands, option numbers, or selections can be entered.

command-line interface (CLI)
A type of computer interface in which the input command is a string of text characters.

commit
To apply all the changes made during the current unit of recovery (UR) or unit of work (UOW). After the operation is complete, a new UR or UOW can begin.

common area

In a web page that is based on a page template, the fixed region of the page.

Common Base Event
: A specification based on XML that defines a mechanism for managing events, such as logging, tracing, management, and business events, in business enterprise applications. See also situation.

common client interface (CCI)
: A standard interface that allows developers to communicate with enterprise information systems (EISs) through specific resource adapters, using a generic programming style. The generic CCI classes define the environment in which a J2EE component can send and receive data from an EIS.

Common Criteria
: A framework for independent assessment, analysis, and testing of IT products to a set of security requirements.

Common Event Infrastructure (CEI)
: The implementation of a set of APIs and infrastructure for the creation, transmission, persistence, and distribution of business, system, and network Common Base Events. See also event emitter.

Common Gateway Interface (CGI)
: An Internet standard for defining scripts that pass information from a web server to an application program, through an HTTP request, and vice versa.

Common Object Request Broker Architecture (CORBA)
: An architecture and a specification for distributed object-oriented computing that separates client and server programs with a formal interface definition. See also Internet Inter-ORB Protocol.

Common Secure Interoperability Version 2
: An authentication protocol developed by the Object Management Group (OMG) that supports interoperability, authentication delegation and privileges.

communications enabled application
: A software application that uses an IP network and communications technology to accomplish business objectives. Enterprise applications can be communications enabled with web telephony components and collaborative web services that allow users to dynamically interact through shared browser sessions over a secure network.

community manager
: See internal partner.

community operator
: The service provider who has a restricted set of typical day-to-day administrative responsibilities for the hub.

compensation
: The means by which operations in a process that have successfully completed can be undone if an error occurs, to return the system to a consistent state.

compensation flow
: Flow that defines the set of activities that are performed while the transaction is being rolled back to compensate for activities that were performed during the normal flow of the process. A compensation flow can also be called from a compensate end or intermediate event.

compensation service
: The operation that is performed to compensate for a successful operation when a process generates a fault (which is not handled within the process).

compilation unit
: A portion of a computer program sufficiently complete to be compiled correctly.

compiled map component
: An Integration Flow Designer object that references an executable map in compiled file format.

compile time
: The time period during which a computer program is being compiled into an executable program.

complete lifecycle server
: A server that the user can create and manage within the administrative console.

complete type name
: The name of a type that represents its hierarchical structure within a type tree, which includes the names of all the types in the path from the root type down.

complex change
: A single operation that impacts one or more ontologies and spans multiple repository versions. Examples of a complex change are ontology content pack (OCP) imports and ontology deletions.

complex scorecard
: A collection of multiple scorecards within one rule flow. The rule flow is used to identify the dependency, the flow and the order in which the score from each of the scorecards will be included in the overall score.

complex type
: A type that contains elements and can include attributes. See also simple type.

component
: 1. A reusable object or program that performs a specific function and works with other components and applications.
2. In Eclipse, one or more plug-ins that work together to deliver a discrete set of functions.

component element
: An entity in a component where a breakpoint can be set, such as an activity or Java snippet in a business process, or a mediation primitive or node in a mediation flow.

component instance
: A running component that can be running in parallel with other instances of the same component.

component rule
: An expression about one or more components, which is defined in the Type Designer. A component rule is used for validating data and specifies what must be true for the data that is defined by that component to be valid.

component test
: An automated test of one or more components of an enterprise application, which may include Java classes, EJB beans, or web services. See also abstract test, test pattern.

composer
: In Java, a class used to map a single complex bean field to multiple database columns. Composition is needed for complex fields that are themselves objects with fields and behavior.

composite
: 1. A group of related data elements used in EDI transactions.
2. A Service Component Architecture (SCA) element that contains components, services, references, and wires that connect them.

composite business policy
: A runtime aggregation of business policies based on context, content and contract of a service request.

composite business service (CBS)
: A collection of business services that work together, along with the existing applications of a client, to provide a specific business solution.

composite service
: In service-oriented architecture, a unit of work accomplished by an interaction between computing devices.

composite state
: In a business state machine, an aggregate of one or more states that is used to decompose a complex state machine diagram into a simple hierarchy of state machines.

compositional hierarchy
: A hierarchy in which the composition of the data is reflected in the structure of the group type in the group window. See also classification hierarchy.

composition unit
: A unit that represents a configured asset and enables the asset contents to interact with other assets in the application.

compound activity
: An activity that has detail that is defined as a flow of other activities. A compound activity is a branch (or trunk) in the tree-structure hierarchy of process activities. Graphically, a compound activity is a process or subprocess.

compound element
: An item in the source or target document that contains child items, such as EDI Segments and EDI composite data elements, ROD records and ROD structures in record oriented data, and XML elements.

concept
: A class of entities that are represented by general metadata definitions rather than physical document standards.

concrete portlet
: A logical representation of a portlet object distinguished by a unique configuration parameter (PortletSettings).

concrete type
: A type that can be instantiated and is derived from an abstract type.

concurrency control
: The management of contention for data resources.

Concurrent Versions System (CVS)
: An open-source, network-transparent version control system.

condition
: 1. A test of a situation or state that must be in place for a specific action to occur.
  2. In a business state machine, an expression that guards the transition and allows transition to the next state only when and if the incoming operation evaluates to 'True'. Otherwise, the current state is maintained.

conditional mean
: An alternative method for choosing a reason code assignment. It is a reasoning strategy that the user specifies in the scorecard requirements, which evaluates every attribute’s possible value ranges. The reason codes are determined based on the expected value of the attributes and rank order. Typically, the four lowest expected and their corresponding reason codes are chosen. A conditional mean can be used in neural nets and fused scorecards.

condition column
: The condition part of a decision table.

condition node
: A node in a decision tree that defines a rule condition and groups a set of branches.

configuration
: In a broker domain, the brokers, execution groups, deployed message sets, and deployed message flows, and the defined topics and access control lists.

configuration administration
: The administration of the configuration object types (CTs), configuration objects (COs), and configuration object sets (COSs) that comprise the configuration data of organizational units (OUs). This is carried out after the product has been installed and customized.

configuration database
: The Data Interchange Services client database that stores parameters necessary for running Data Interchange Services client, including database definitions, messages, queries, and preferences.

configuration entity
: Entities used to model an organization and to specify how messages are processed. These entities include configuration object types (CTs), organizational units (OUs), configuration object sets (COSs), configuration objects (COs).

configuration object (CO)
: An instance of a configuration object type (CT) that represents an object in an organizational unit (OU). Which attributes can be added to a CO is determined by the definition of the CT on which the CO is based.

configuration object set (COS)
: A set of configuration objects, used to limit the scope of configuration data provided to message flows.

configuration object type (CT)
: A description of the class of configuration objects, including the attributes that each member of this class can have.

configuration repository
: A storage area of configuration data that is typically located in a subdirectory of the product installation root directory.

configured name binding
: Persistent storage of an object in the name space that is created using either the administrative console or the wsadmin program.

confirm-on-arrival report (COA report)
: A WebSphere MQ report message type created when a message is placed on that queue. It is created by the queue manager that owns the destination queue.

confirm-on-delivery report (COD report)
: A WebSphere MQ report message type created when an application retrieves a message from the queue in a way that causes the message to be deleted from the queue. It is created by the queue manager.

conflict
: A result that occurs when two simultaneous edit submissions are processed for the same object and where the intended outcome of the edit is unclear.

connection
: A link between two process elements. Connections can be used to specify the chronological sequence of activities in a process.

connection factory

A set of configuration values that produces connections that enable a Java EE component to access a resource. Connection factories provide on-demand connections from an application to an enterprise information system (EIS) and allow an application server to enroll the EIS in a distributed transaction.

connection handle
A representation of a connection to a server resource.

connection pool
A group of host connections that are maintained in an initialized state, ready to be used without having to create and initialize them.

connection pooling
A technique used for establishing a pool of resource connections that applications can share on an application server.

connectivity
The capability of a system or device to be attached to other systems or devices without modification.

connector

1. In Java EE, a standard extension mechanism for containers to provide connectivity to enterprise information systems (EISs). A connector consists of a resource adapter and application development tools (Sun). See also container.
2. A servlet that provides a portlet access to external sources of content, for example, a news feed from a website of a local television station.

connector packet
The set of data that is passed between the event processing server (runtime server) and external systems using the technology connectors.

consistent-change-data table (CCD table)
In data replication, a type of replication target table that is used for storing history, auditing data, or staging data. A CCD table can also be a replication source.

console
A user interface that allows you to list and manage objects or entities, such as catalogs, hierarchies, and items. See also module.

constant
In a business object model (BOM), a vocabulary element that verbalizes the public static final attribute of a class with the same type as the BOM class. See also verbalization.

constraint
A rule that limits the values that can be inserted, deleted, or updated in a table. See also foreign key, primary key.

consumer portal
A portal that uses the portlets that a producer portal provides. See also producer definition, producer portal.

container

1. An item that can contain other items. Tags that are added to a container inherit the position of the container.
2. An entity that provides life-cycle management, security, deployment, and runtime services to components. (Sun) See also connector, resource adapter.

container-managed persistence (CMP)
The mechanism whereby data transfer between an entity bean's variables and a resource manager is managed by the entity bean's container. (Sun) See also bean-managed persistence.

container-managed transaction
A transaction whose boundaries are defined by an EJB container. An entity bean must use container-managed transactions. (Sun)

container server
A server instance that can host multiple shards. One Java virtual machine (JVM) can host multiple container servers.

container transaction
See container-managed transaction.

containment hierarchy
A namespace hierarchy consisting of model elements, and the containment relationships that exist between them. A containment hierarchy forms an acyclic graph.

containment relationship
A relationship between two objects where one object is contained within the other. The destination is nested within the source.

content
The data semantics of a message that is received by the dynamic assembler.

content area
In a web page that is based on a page template, the editable region of the page.

content assist
A feature of some source editors that prompts the user with a list of valid alternatives for completing the current line of code or input field.

content based routing (CBR)
An optional feature of the caching proxy that provides intelligent routing to back-end application servers. This routing is based on HTTP session affinity and a weighted round-robin algorithm.

contention
A situation in which a transaction attempts to lock a row or table that is already locked.

content management
Software designed to help businesses manage and distribute content from diverse sources.

content model
The representation of any data that may be contained inside an XML element. There are four kinds of content models: element content, mixed content, EMPTY content and ANY content.

content provider
A source for content that can be incorporated into a portal page as a portlet.

content spot
A class file that is added to a JSP file to designate display of personalized data or content. Each content spot has a name and will accept a specific type of data from a rule.

context
An object created for a service request in the business service model. The object contains one or more of the following details of information captured from the metadata: a business process, organization, role, channel, and domain specific information. See also context propagation.

context propagation

In a multiple service transaction, the information about the details of a service request that passes from one invocation to another via the message header. See also context.

context root
: The web application root, which is the top-level directory of an application when it is deployed to a web server.

context-scoped business object
: A summary object or accumulating array object that is used to share data from events across events in a context.

contract
: The set of business policy assertions that have to be met by service provider at run time based on the context and content.

contracted component
: In the Integration Flow Designer, a component that does not display the sources and targets associated with it. See also expanded component.

contribution
: The primary asset that can contain Service Component Definition Language (SCDL) with composite definitions, as well as artifacts such as Java classes and Web Services Description Language (WSDL) and XML Schema Definitions (XSD).

control
: See widget.

control analysis
: A type of analysis that displays variations in values of the business measures over a specific period of time. This type of analysis reduces data variation, and is often used for quality control. Allowable variation is three times the standard deviation of the data.

controlled flow
: A flow that proceeds from one flow object to another through a sequence flow link but is subject to either conditions or dependencies from another flow as defined by a gateway. Typically, a controlled flow is a sequence flow between two activities, with a conditional indicator or a sequence flow that is connected to a gateway.

controller
: A component or a set of virtual storage processes that schedules or manages shared resources.

control link
: An object in a process that links nodes and determines the order in which they run.

control number
: A number that is used to identify an interchange, group, or EDI document.

control region adjunct
: A servant that interfaces with service integration buses to provide messaging services.

control string
: One of several compiled objects, which consist primarily of map control strings and document definition control strings.

control structure
: The beginning and ending segments (header and trailer) of EDI-enveloped documents.

conversational processing
: An optional IMS facility with which an application program can accumulate information acquired through multiple interchanges with a terminal, even if the program stops between interchanges. See also IMS conversation.

converter
: In Enterprise JavaBeans (EJB) programming, a class that translates a database representation to an object type and back.

cooperative portlets
: Two or more portlets on the same web page that interact by sharing information. See also Click-to-Action, wire.

Coordinated Universal Time (UTC)
: The international standard of time that is kept by atomic clocks around the world.

copy helper
: An access bean that contains a local copy of attributes from a remote entity bean. Unlike bean wrappers, copy helpers are optimized for use with a single instance of an entity bean.

CORBA
: See Common Object Request Broker Architecture.

core group
: A group of processes that is directly accessible to each other and is connected using a local area network (LAN).

core group access point
: A definition of a set of servers that provides access to the core group.

core group bridge
: The means by which core groups communicate.

core group member
: A server included in the cluster of a core group.

correlation
: 1. Data that enables a user to record document-specific correlation parameters generated during translation, by the correlation service, or by document tracking functions.
  2. A mechanism that bridges a point in a process flow between two or more process instances.
  3. The relationship, captured in a correlation expression, that describes how an incoming event is matched with one or more monitoring context instances to which it will be delivered.
  4. A record used with business processes and state machines to allow two partners to initialize a transaction, temporarily suspend an activity, and then recognize each other again when that activity resumes.

correlation property
: Data in an event that the runtime server uses to determine which instance of a task, process, or business state machine should receive the input at run time.

correspondent
: An institution to which your institution sends and from which it receives messages.

COS
: See configuration object set.

cost
: A number that is used as a weighting mechanism to differentiate one resource from another where a smaller value is always preferred.

counter

A specialized metric used to keep track of the number of occurrences of a specific situation or event. For example, you can use a counter to track the number of times that a task is started within a process, where that task is contained in a loop.

coupling
: The dependency that components have on one another.

create method
: In enterprise beans, a method defined in the home interface and invoked by a client to create an enterprise bean. (Sun)

credential
: 1. In the Java Authentication and Authorization Service (JAAS) framework, a subject class that owns security-related attributes. These attributes can contain information used to authenticate the subject to new services.
  2. Information acquired during authentication that describes a user, group associations, or other security-related identity attributes, and that is used to perform services such as authorization, auditing, or delegation. For example, a user ID and password are credentials that allow access to network and system resources.

critical path
: The processing path that takes the longest time to complete of all parallel paths in a process instance, where each path considered begins at a start node or an input to the process and ends at a terminate node.

CRL
: See certificate revocation list.

cross-cell communication
: The process of information sharing and request routing between cells.

cross-cell environment
: A production environment in which one or more servers in one cell can receive events from another server or set of servers in another cell.

cross-cutting concern
: A software concern (synchronization, logging, memory allocation, and so forth) that is external and orthogonal to the problem that a software component is designed to address.

cryptographic token
: A logical view of a hardware device that performs cryptographic functions and stores cryptographic keys, certificates, and user data.

CSR
: See certificate signing request.

CSS
: See Cascading Style Sheets.

CSV file
: A text file that contains comma-separated values. A CSV file is commonly used to exchange files between database systems and applications that use different formats.

CT
: See configuration object type.

cube
: A multidimensional representation of data needed for online analytical processing, multidimensional reporting, or multidimensional planning applications.

current customization definition
: A customization definition that describes an instance for which the corresponding resources have already been deployed and are running.

custom action
: 1. In JSP programming, an action described in a portable manner by a tag library descriptor and a collection of Java classes and imported into a JSP page by a taglib directive. (Sun)
  2. A Java or non-Java process definition that a user can define as a part of a health policy action plan or elasticity operation.

Custom-built Product Delivery Option (CBPDO)
: A software delivery package consisting of uninstalled products and unintegrated service. Installation requires the use of SMP/E. CBPDO is one of the two entitled methods for installing z/OS; the other method is ServerPac.

custom finder
: See finder method.

customization definition document (CDD)
: An XML document that describes the layout of an instance (that is, its organizational units (OUs) and servers, and which service bundles are assigned to each server-OU combination). The Customization Definition Program (CDP) uses a CDD to determine which deployment data to produce for an instance.

customization definition report
: A report that describes the servers, organizational units (OUs), and services of an instance, and how they are distributed within the instance.

customization time data
: See build time data.

customized installation package (CIP)
: A customized installation image that can include one or more maintenance packages, a configuration archive file from a stand-alone server profile, one or more enterprise archive files, scripts, and other files that help customize the resulting installation.

customizer
: A Java class (implementing the java.beans.Customizer interface) that is associated with a bean to provide a richer user interface for the properties of that bean.

custom profile
: A profile that describes an empty node, which becomes operational, as a managed node, when federated into a network deployment cell.

custom relationship
: An association between two or more data entities as provided by the user.

custom screen record
: A runtime view of the screen that allows access to available screen fields.

custom service
: A configurable service that defines a hook that runs when the server starts and shuts down when the server stops.

custom tag
: An extension to the JavaServer Pages (JSP) language that performs a specialized task. Custom tags are typically distributed in the form of a tag library, which also contains the Java classes that implement the tags.

custom user registry
: A customer-implemented user registry that implements the UserRegistry Java interface. This registry type can support virtually any kind of accounts repository from a relational database and can provide flexibility in adapting product security to various environments.

CVS
: See Concurrent Versions System.

cycle time
: The time required for a process instance in a process simulation run to finish processing its inputs. Cycle time includes idle time when an activity in the process is waiting for a resource to become available.

D

DAD
: See document access definition.

DAD script
: A file that is used by the DB2 XML Extender, either to compose XML documents from existing DB2 data or to decompose XML documents into DB2 data.

DADX
: See document access definition extension.

DADX group
: A folder that contains database connection (JDBC and JNDI) and other information that is shared between DADX files within the group.

DADX runtime environment
: The DADX runtime environment provides information to the DADX web service, including the HTTP GET and POST bindings, the test page, WSDL generation, and the translation of DTD data into XML schema data.

daemon
: A program that runs unattended to perform continuous or periodic functions, such as network control.

dashboard
: A web page that can contain one or more widgets that graphically represent business data.

data access bean
: A class library that provides a rich set of features and functions, while hiding the complexity associated with accessing relational databases.

database cleanup
: The act of deleting from a database those records for which the cleanup period has expired.

database definition
: A Data Interchange Services definition that contains information used by Data Interchange Services Client to connect to a database.

database management system (DBMS)
: See database manager.

database manager
: A program that manages data by providing centralized control, data independence, and complex physical structures for efficient access, integrity, recovery, concurrency control, privacy, and security.

database request module (DBRM)
: A data set member that is created by the DB2 for z/OS precompiler and that contains information about SQL statements. DBRMs are used in the bind process.

data binding
: A component that converts protocol-specific local data to and from a business object.

data catalog
: A collection of models representing objects, such as business items and notifications, to be used as inputs and outputs in process modeling.

data class
: An access bean that provides data storage and access methods for caching enterprise bean properties. Unlike copy helpers, data class access beans work with enterprise beans that have local client views as well as remote client views.

data connection
: A connection to a repository of data (for example, a DB2 database) with which the runtime server can retrieve data in order to enhance the event being processed.

data definition
: A data object that defines a database or table.

Data Definition Language (DDL)
: A language for describing data and its relationships in a database.

data dictionary
: A grouping of logically related components of a particular syntax type, such as ROD dictionaries, EDI dictionaries, and XML dictionaries.

data element delimiter
: A character, such as an asterisk (*), that follows the EDI segment identifier and separates each EDI data element in an EDI segment. See also segment ID separator.

Data Encryption Standard (DES)
: A cryptographic algorithm designed to encrypt and decrypt data using a private key.

Data Exchange SPI architecture (DESPI)
: The interface that resource adapters and runtime components use to exchange business object data. The Data Exchange SPI architecture, which is based on the concept of cursors and accessors, abstracts the data type so that an adapter can be written only once and then work on runtime environments that support different data types, such as data objects and JavaBeans.

datagram
: A form of asynchronous messaging in which an application sends a message, but does not require a response. See also request/reply.

data graph
: A set of Service Data Objects (SDO) interconnected with relationships.

data grid
: A system of data that dynamically caches, partitions, replicates, and manages application data and business logic across multiple servers.

data handler
: A Java class or library of classes that a process uses to transform data into and from specific formats. In the business integration environment, data handlers transform text data of specified formats into business objects, and transform business objects into text data of specified formats.

Data Interchange Services client (DIS client)

The Data Interchange Services tool used to document metadata and map documents to one another.

Data Interchange Services database
: The database that contains all Data Interchange Services objects.

Data Interchange Services translator
: The Data Interchange Services component responsible for transforming a document from one format to another.

data model
: A model defining the structure of business artifacts that are operated upon by business operations.

data object

1. A portion of data in a data stream that can be recognized as belonging to a specific type.
2. An object that provides information about required activities. Data objects can represent one object or a collection of objects.
3. Any object (such as tables, views, indexes, functions, triggers, and packages) that can be created or manipulated using SQL statements. See also business object.

data object filter
: A control that allows the exclusion of data objects (such as tables and schemas) from the tree view of the database.

data source

1. The means by which an application accesses data from a database.
2. In JDBC, an interface that provides a logical representation of a pool of connections to a physical data source. Data source objects provide application portability by making it unnecessary to supply information specific to a particular database driver.

data store

1. A data structure where documents are kept in their parsed form.
2. A place (such as a database system, file, or directory) where data is stored.

data store profile
: An object that defines properties used by the default data store plug-in, which is used to persistently store events received by the event server.

data structure
: The composition of the data, including repeating sub-structures, nested groupings, sequences, and choices.

Data Transformation Framework (DTF)
: An infrastructure that includes data bindings and function selectors, which enables an adapter to convert native data formats to business objects and to convert business objects back to native data formats, such as XML.

data transformation map
: A set of mapping instructions that describes how to translate data from a source document into a target document. Both the source and target documents can be one of several supported document types. A data transformation map is one of three supported map types.

Data Universal Numbering System (DUNS)
: A system in which internationally recognized nine-digit numbers are assigned and maintained by Dun & Bradstreet to uniquely identify worldwide businesses. See also partner profile.

DB2
: A family of IBM licensed programs for relational database management.

DB2 XML Extender
: A program that is used to store and manage XML documents in DB2 tables. Well-formed and validated XML documents can be generated from existing relational data, stored as column data, and the content of XML elements and attributes can be stored in DB2 tables.

DBMS
: See database management system.

DBRM
: See database request module.

DDL
: See Data Definition Language.

dead-letter queue (DLQ)
: A queue to which a queue manager or application sends messages that cannot be delivered to their correct destination.

deadlock
: A condition in which two independent threads of control are blocked, each waiting for the other to take some action. Deadlock often arises from adding synchronization mechanisms to avoid race conditions.

debug engine
: The server component of the debugger, whose client/server design enables both local and remote debugging. The debug engine runs on the same system as the program being debugged.

debugger
: A tool used to detect and trace errors in computer programs.

debugging session
: The debugging activities that occur between the time that a developer starts a debugger and the time that the developer exits from it.

decimal notation
: In an EDI Standard, the character that represents a decimal point.

decision

1. A gateway within a business process where the sequence flow can take one of several alternative paths.
2. A gateway that routes an input to one of several alternative outgoing paths, depending on its condition. A decision is like a question that determines the exact set of activities during the execution of a process. Questions might include: What type of order? Or How will the order be shipped?

Decision Center console
: A designated workspace where business users can work collaboratively to author, edit, organize, and search for business rules.

decision table
: A form of business rule that captures multi-conditional decision-making business logic in a table where the rows and columns intersect to determine the appropriate action. See also rule set.

decision tree

A way of representing business rules in a tree form. Decision trees provide a structure for laying out options and investigating the possible outcomes of choosing those options.

Decision Validation Services (DVS)
A set of testing and simulation capabilities with which business users and policy managers can verify the rules they have written, and determine if potential changes will have the intended outcome.

Decision Warehouse
A warehouse that saves execution traces to a database so that users can query the data store to get information on particular executions or transactions.

deck
An XML document that contains a collection of Wireless Markup Language (WML) cards. See also card.

declaration
In Java programming, a statement that establishes an identifier and associates attributes with it, without necessarily reserving its storage or providing the implementation. (Sun)

declarative security
The security configuration of an application during assembly stage that is defined in the deployment descriptors and enforced by the security run time.

decode
To convert data by reversing the effect of some previous encoding.

decoration
In graphical user interfaces (GUIs), a glyph that annotates a resource with status information, for example to indicate that a file has changed since it was last saved or checked out of a repository.

de-envelope
To extract a document from an EDI envelope.

de-enveloping
The process of removing one or more envelopes from a document or a set of documents.

default portal page
The page that displays to a user at initial portal deployment and before the user completes enrollment. Sometimes used as a synonym for home page.

default public place
A place whose membership automatically includes all users and which appears in the Places selector for every user. A user is always a member of this place.

definition file
A file that defines the content that is displayed within the navigation and work area frames.

delegation
The process of propagating a security identity from a caller to a called object. According to the Java Platform, Enterprise Edition (Java EE) specification, a servlet and an enterprise bean can propagate either the client identity when invoking enterprise beans, or can use another specified identity as indicated in the corresponding deployment descriptor.

delimited format
Data that has data objects that are separated by delimiters.

delimited text
A simple file format that consists of text separated into meaningful chunks by specific characters. The chunks of text are typically individual fields. The specific character is called a delimiter, and can be any character that is not found in the text. Comma and tab are common delimiters. If the delimiter is used as a character in the text, it must be enclosed by a pair of text qualifiers, usually double quotation marks.

delimiter

1. A flag that is formed by a character or a sequence of characters to group or separate items of data by marking the beginning and end of a unit of data. The delimiter is not a part of the flagged unit of data.
2. A character, such as comma or tab, used to group or separate units of text by marking the boundary between them.

delta business object
A business object used in an update operation. Such a business object contains only key values and the values to be changed. See also after-image.

delta deployment
Deployment of only that data that is required to transform a current runtime environment into a target runtime environment. See also full deployment.

demilitarized zone (DMZ)
A configuration that includes multiple firewalls to add layers of protection between a corporate intranet and a public network, such as the Internet.

denial-of-service attack (DoS)
In computer security, an assault on a network that brings down one or more hosts on a network such that the host is unable to perform its functions properly. Network service is interrupted for some period.

dependency

1. A requirement that one managed resource has on another managed resource in order to operate correctly.
2. A relationship that allows a module to use artifacts from a library or that allows a process application to use artifacts from a toolkit. A toolkit can also have a dependency on another toolkit.

dependency relationship
In UML modeling, a relationship in which changes to one model element (the supplier) impact another model element (the client).

deploy
To place files or install software into an operational environment. In Java Platform, Enterprise Edition (Java EE), this involves creating a deployment descriptor suitable to the type of application that is being deployed.

deployment
The process of transferring rules from a local development environment into an operational, or runtime, environment.

deployment code
Additional code that enables bean implementation code written by an application developer to work in a particular EJB runtime environment. Deployment code can be generated by tools that the application server vendor supplies.

deployment data
The resource files, generated during customization, that are used to create the resources for an instance.

deployment data set
A data set containing the resource files generated during customization.

deployment descriptor

An XML file that describes how to deploy a module or application by specifying configuration and container options. For example, an EJB deployment descriptor passes information to an EJB container about how to manage and control an enterprise bean.

deployment directory

1. The directory where the published server configuration and web application are located on the machine where the application server is installed.
2. The directory containing the subdirectories and resource files created during customization.

deployment environment
: A collection of configured clusters, servers, and middleware that collaborate to provide an environment to host software modules. For example, a deployment environment might include a host for message destinations, a processor or sorter of business events, and administrative programs.

deployment instruction
: A set of instructions that describe how to execute the resource files, and deploy, on the runtime systems, the resources required by the instance.

deployment manager
: A server that manages and configures operations for a logical group or cell of other servers. See also subprocess.

deployment phase
: A phase that includes a combination of creating the hosting environment for your applications and the deployment of those applications. This includes resolving the application's resource dependencies, operational conditions, capacity requirements, and integrity and access constraints.

deployment policy

1. An optional way to configure an eXtreme Scale environment based on various items, including: number of systems, servers, partitions, replicas (including type of replica), and heap sizes for each server.
2. A policy that modifies the domain configuration at deployment time to accommodate the environment in which the appliance operates.

deployment topology
: The configuration of servers and clusters in a deployment environment and the physical and logical relationships among them.

deployment vehicle
: A job or other executable file that is used to deploy resources. Each vehicle corresponds to a particular resource file.

deploy phase
: See deployment phase.

deprecated
: Pertaining to an entity, such as a programming element or feature, that is supported but no longer recommended and that might become obsolete.

dequeue
: To remove items from a queue. See also enqueue.

DER
: See Distinguished Encoding Rules.

Derby
: An embeddable, all Java, object-relational database management system (ORDBMS).

derivation
: In object-oriented programming, the refinement or extension of one class from another.

derived event
: See synthetic event.

derived page
: One or more child pages that have a shared layout that is inherited from the properties of the parent page.

DES
: See Data Encryption Standard.

deserialization
: A method for converting a serialized variable into object data. See also serializer.

DESPI
: See Data Exchange SPI architecture.

destination
: An exit point that is used to deliver documents to a back-end system or a trading partner.

developer
: A decision management user role that is responsible for implementing the rule applications.

device
: A component that is used for an event provider to provide location, notification, or telemetry data. Devices always belong to a hub and can be grouped in device groups.

device input format (DIF)
: The Message Format Service (MFS) control block that describes the format of the data that is entered on the device and presented to MFS.

device output format (DOF)
: The Message Format Service (MFS) control block that describes the format of the output data that is presented to the device.

dialog
: The recorded interaction between a user and the 3270 application that the user accesses. Users can record a dialog using the Record Dialog function in the 3270 terminal service recorder. A recorded dialog includes the keystrokes, inputs and outputs that move the user from one screen to another in the 3270 application.

dialog editor
: A 3270 terminal service development tool that enables a developer to modify the dialog that was recorded with the 3270 terminal service recorder.

dialog file
: The result of recording a dialog from the 3270 terminal service recorder. The dialog file is saved to a WSDL file in the workbench.

dictionary
: A grouping of logically related components of a particular syntax type, such as ROD dictionaries, EDI dictionaries, and XML dictionaries.

DIF
: See device input format.

digest code
: A number that is the result of a message digest function or a secure hash algorithm distilling a document.

digital certificate

An electronic document used to identify an individual, a system, a server, a company, or some other entity, and to associate a public key with the entity. A digital certificate is issued by a certification authority and is digitally signed by that authority.

digital signature
Information that is encrypted with a private key and is appended to a message or object to assure the recipient of the authenticity and integrity of the message or object. The digital signature proves that the message or object was signed by the entity that owns, or has access to, the private key or shared-secret symmetric key.

digital signature algorithm (DSA)
A security protocol that uses a pair of keys (one public and one private) and a one-way encryption algorithm to provide a robust way of authenticating users and systems. If a public key can successfully decrypt a digital signature, a user can be sure that the signature was encrypted using the private key.

dimension
A data category that is used to organize and select monitoring context instances for reporting and analysis. Examples of dimensions are time, accounts, products, and markets. See also member.

dimensional model
The part of the monitor model that defines the cubes and cube content that are used for storing, retrieving, and analyzing the data that is gathered over time.

dimension level
An element or subelement of a dimension that is arranged hierarchically. For example, the time dimension can have years, months, and days as its levels.

directive
A first-failure data capture (FFDC) construct that provides information and suggested actions to assist a diagnostic module in customizing the logged data.

dirty read
A read request that does not involve any locking mechanism. This means that data can be read that might later be rolled back resulting in an inconsistency between what was read and what is in the database.

DIS client
See Data Interchange Services client.

discover
In UDDI, to browse the business registry to locate existing web services for integration.

discovered server
A server that runs the middleware agent and is found outside of the administrative environment but has a server representation automatically created within the administrative environment. The representation that is created is an assisted life-cycle server.

dispatcher
A standalone application that acts as an intermediary between one or more devices and large event providers. The dispatcher retrieves all location messages from the event providers it is connected to and distributes them to one or more devices.

distinguishable types
Types that do not have common data objects.

Distinguished Encoding Rules (DER)
A standard, based on the Basic Encoding Rules, that is designed to ensure a unique encoding of each ASN.1 value, defined in ITU-T X.690.

distinguished name (DN)

1. A set of name-value pairs (such as CN=person name and C=country or region) that uniquely identifies an entity in a digital certificate.
2. The name that uniquely identifies an entry in a directory. A distinguished name is made up of attribute:value pairs, separated by commas.

distributed application
An application made up of distinct components that are located on different computer systems, connected by a network. See also client/server.

distributed eXtreme Scale
A usage pattern for interacting with eXtreme Scale when servers and clients exist on multiple processes.

DLL
See dynamic link library.

DLQ
See dead-letter queue.

DMZ
See demilitarized zone.

DN
See distinguished name.

DNS
See Domain Name System.

document
A business document, such as a purchase order or invoice, that can be represented in any supported format. For example, an XML purchase order and an EDI purchase order are both documents, but each uses a different format.

document access definition (DAD)
An XML document format used by DB2 XML Extender to define the mapping between XML and relational data.

document access definition extension (DADX)
An XML document format that specifies how to create a web service using a set of operations that are defined by DAD documents and SQL statements.

document definition
A description of a document layout that is used to identify the format of a document. Examples include record oriented data document definitions, EDI document definitions, XML schema document definitions, and XML DTD document definitions.

document envelope
A structure that is applied to a document to prepare it for exchange between trading partners.

document flow definition
A collection of information specified for each type of document that tells the hub how to process that particular type of document. Each document to be exchanged between the internal partner and a participant must have a document flow definition.

document ID
A unique identifier for a document.

document literal wrapped
A convention or style that is used to structure a web service definition to generate a SOAP message that is WS-I compliant and can be easily validated.

Document Object Model (DOM)

A system in which a structured document, for example an XML file, is viewed as a tree of objects that can be programmatically accessed and updated. See also Simple API for XML.

document type
A classification that helps to organize and classify documents that belong to a specific case. Properties can be assigned to a document type to provide additional information about the documents. An example of a document type is a job application form. See also case property.

document type definition (DTD)
The rules that specify the structure for a particular class of SGML or XML documents. The DTD defines the structure with elements, attributes, and notations, and it establishes constraints for how each element, attribute, and notation can be used within the particular class of documents.

DOF
See device output format.

DOM
See Document Object Model.

domain
1. A logical grouping of resources in a network for the purpose of common management and administration. See also federation domain.
2. An object, icon, or container that contains other objects representing the resources of a domain. The domain object can be used to manage those resources.

Domain Name System (DNS)
The distributed database system that maps domain names to IP addresses.

DOM element
One member of a tree of elements that is created when an XML file is parsed with a DOM parser. DOM elements make it easy to quickly identify all elements in the source XML file.

DoS
See denial-of-service attack.

do-while loop
A loop that repeats the same sequence of activities as long as some condition is satisfied. Unlike a while loop, a do-while loop tests its condition at the end of the loop. This means that its sequence of activities always runs at least once.

downstream
Pertaining to the direction of the flow, which is from the first node in the process (upstream) toward the last node in the process (downstream).

DSA
See digital signature algorithm.

DSO
See dynamic shared object.

DTD
See document type definition.

DTD document definition
A description or layout of an XML document based on an XML DTD.

DTF
See Data Transformation Framework.

dual authorization
A setting requiring that an action carried out by one person be confirmed by a second person. This prevents a single person from being able to carry out actions requiring a high level of security, for example the distribution of funds or the granting of access rights. See also single authorization.

DUNS
See Data Universal Numbering System.

durable subscription
A Java Message Service (JMS) subscription that persists and stores subscribed messages even when the client is not connected.

DVS
See Decision Validation Services.

dynaform
An instance of a DynaActionForm class or subclass that stores HTML form data from a submitted client request or that stores input data from a link that a user clicked.

dynamic analysis
The process of extracting targeted types of information based on the results of process simulations. This differs from static analysis, which extracts information from model elements in their static form.

dynamic assembly
A process that selects specific endpoints to meet the conditions of a service request at run time.

dynamic cache
A consolidation of several caching activities, including servlets, web services, and commands into one service where these activities share configuration parameters and work together to improve performance.

dynamic cluster
A server cluster that uses weights to balance the workloads of its cluster members dynamically, based on performance information collected from cluster members.

dynamic cluster isolation
The ability to specify whether the dynamic cluster runs on the same nodes as other instances of dynamic clusters, or if the dynamic cluster is the only dynamic cluster that runs on a single node.

dynamic domain
A domain in which the set of possible values for a type is defined. With a dynamic domain, the set of values is stored and managed outside the business object model (BOM), and changes to the set of values are automatically reflected in the business object model. When rule authors write business rules using the type, they choose from a list of values that is created dynamically and is always up-to-date.

dynamic link library (DLL)
A file containing executable code and data bound to a program at load time or run time, rather than during linking. The code and data in a DLL can be shared by several applications simultaneously. See also library.

dynamic operations
Operations that monitor the server environment and make recommendations that are based on the observed data.

dynamic policy

A template of permissions for a particular type of resource.

dynamic property
: A property that can be overridden at run time by inserting information into the service message object (SMO).

dynamic reloading
: The ability to change an existing component without restarting the server for the changes to become effective. See also hot deployment.

dynamic routing
: The automatic routing of a service request, a message, or an event that is based on conditions at the time of the routing.

dynamic shared object (DSO)
: A mechanism that provides a way to build a piece of program code in a special format for loading into the address space of an executable program at run time. The DSO gets knowledge of the executable program symbol set as if it had been statically linked with it in the first place.

dynamic web content
: Programming elements such as JavaServer Pages (JSP) files, servlets, and scripts that require client or server-side processing for accurate runtime rendering in a web browser.

dynamic web project
: A project that contains resources for a web application with dynamic content such as servlets or JavaServer Pages (JSP) files. The structure of a dynamic web project reflects the Java EE standard for web content, classes, class paths, the deployment descriptor, and so on.

dynamic workload manager
: A feature of the on demand router that routes workload based on a weight system, which establishes a prioritized routing system. The dynamic workload manager dynamically modifies the weights to stay current with the business goals.

E

EAR
: See enterprise archive.

EAR file
: See enterprise archive.

early bind
: To connect one process to another process so that a specific version of the called process is used. The calling process always uses the specified version of the invoked process even if updated versions are available.

early binding
: The connection between two processes that uses a specified version of the invoked process. As a result, the calling process uses the specified version of the process that it is invoking, even when updated versions are available.

EAR project
: See enterprise application project.

Eclipse
: An open-source initiative that provides independent software vendors (ISVs) and other tool developers with a standard platform for developing plug-compatible application development tools.

ECSA
: See extended common service area.

Edge Side Include (ESI)
: A technology supporting cacheable and noncacheable web page components that can be gathered and assembled at the edge of a network.

EDI
: See electronic data interchange.

EDI administrator
: The person responsible for setting up and maintaining Data Interchange Services.

EDI composite data element
: A group of related EDI data elements, such as the elements that make up a name and address.

EDI data element
: A single item of data in an EDI document, such as a purchase order number, that corresponds to a ROD field in a ROD document definition. An EDI data element is equivalent to a simple element. It is also used to maintain EDI composite data elements.

EDI document definition
: A description or layout of an EDI document, which comprises loops, EDI segments, EDI data elements, and EDI composite data elements. It is equivalent to the layout of an EDI transaction or an EDI message.

EDI envelope
: The EDI segments and EDI data elements that make up the headers and trailers that enclose EDI transaction sets, functional groups, and interchanges.

EDI loop
: A group of consecutive EDI segments that repeat together in an EDI document definition. There is no object type in Data Interchange Services that defines an EDI loop on its own. EDI loops are logically defined within an EDI document definition.

EDI message
: In UN/EDIFACT EDI Standards, a group of logically related data that makes up an electronic business document, such as an invoice. It is equivalent to an EDI transaction. Called an EDI document definition in Data Interchange Services.

EDI message set
: A group of logically related data that make up an electronic business document, such as an invoice or a purchase order. A single EDI document. The layout of an EDI transaction is described by an EDI document definition in Data Interchange Services.

EDI segment
: A group of related EDI data elements. An EDI segment is a single line in an EDI document definition, beginning with a segment identifier and ending with a segment terminator delimiter. The EDI data elements in the EDI segment are separated by data element delimiters.

EDI standard
: The industry-supplied, national or international formats to which information is converted, allowing different computer systems and applications to exchange information.

edit conflict
: The result of a user applying changes and the system detecting that another user has made intervening and potentially conflicting changes.

edition
: A successive deployment generation of a particular set of versioned artifacts.

edit mode

The state in which users can create or modify a document.

editor area
: In Eclipse and Eclipse-based products, the area in the workbench window where files are opened for editing.

EDI transaction
: In X12 EDI Standards, a group of logically related data that makes up an electronic business document, such as an invoice. It is equivalent to an EDI message. The layout of an EDI transaction is described by an EDI Document Definition in Data Interchange Services.

EDI transaction set
: A group of logically related data that make up an electronic business document, such as an invoice or a purchase order. A single EDI document.

EIS
: See enterprise information system.

EJB
: See Enterprise JavaBeans.

EJB container
: A container that implements the EJB component contract of the Java EE architecture. This contract specifies a runtime environment for enterprise beans that includes security, concurrency, life cycle management, transaction, deployment, and other services. (Sun) See also EJB server.

EJB context
: In enterprise beans, an object that allows an enterprise bean to invoke services provided by the container and to obtain information about the caller of a client-invoked method. (Sun)

EJB factory
: An access bean that simplifies the creating or finding of an enterprise bean instance.

EJB home object
: In Enterprise JavaBeans (EJB) programming, an object that provides the life cycle operations (create, remove, find) for an enterprise bean. (Sun)

EJB inheritance
: A form of inheritance in which an enterprise bean inherits properties, methods, and method-level control descriptor attributes from another enterprise bean that resides in the same group.

EJB JAR file
: A Java archive that contains an EJB module. (Sun)

EJB module
: A software unit that consists of one or more enterprise beans and an EJB deployment descriptor. (Sun)

EJB object
: In enterprise beans, an object whose class implements the enterprise bean remote interface (Sun).

EJB project
: A project that contains the resources needed for EJB applications, including enterprise beans; home, local, and remote interfaces; JSP files; servlets; and deployment descriptors.

EJB query
: In EJB query language, a string that contains an optional SELECT clause specifying the EJB objects to return, a FROM clause that names the bean collections, an optional WHERE clause that contains search predicates over the collections, an optional ORDER BY clause that specifies the ordering of the result collection, and input parameters that correspond to the arguments of the finder method.

EJB reference
: A logical name used by an application to locate the home interface of an enterprise bean in the target operational environment.

EJB server
: Software that provides services to an EJB container. An EJB server may host one or more EJB containers. (Sun) See also EJB container.

elasticity mode
: A mode that is used to dynamically grow or shrink a cell by adding or removing nodes. Nodes are added when a particular dynamic cluster is not meeting service policies and all possible servers are started. Nodes are removed if they are unused and service policies can be met without them.

elasticity operation
: An operation that adds or removes the resources of the application placement controller depending on the defined runtime behavior.

electronic data interchange (EDI)
: The exchange of structured electronic data between computer systems according to predefined message standards.

element
: 1. A component of a document, such as an EDI, XML, or ROD record. An element can be a simple element or a compound element.
  2. In markup languages, a basic unit consisting of a start tag, end tag, associated attributes and their values, and any text that is contained between the two.
  3. In Java development tools, a generic term that can refer to packages, classes, types, interfaces, methods, or fields.

element separator
: See data element delimiter.

embedded server
: A catalog service or container server that resides in an existing process and is started and stopped within the process.

emitter factory
: A type of factory that handles the details of event transmission such as the event server location, the filter settings, or the underlying transmission mechanism.

empty activity
: An activity with no defined implementation that can be used as a place holder in the design stage.

emulator
: A facility of the integration test client that enables the emulation of components and references during module testing. Emulators are either manual or programmatic. See also manual emulator, programmatic emulator.

encode
: To convert data by the use of a code in such a manner that reconversion to the original form is possible.

end event
: An event that ends a process flow and, therefore, does not have outgoing sequence flow paths. Types of end events are message, terminate, and error. See also error end event, message end event, terminate end event.

end node

1. A visual marker within a process that identifies where a particular flow ends. Other concurrent flows within the same process will still continue executing.
2. A node that identifies where a rule flow stops. A rule flow has at least one end node.

endpoint

1. The system that is the origin or destination of a session.
2. A JCA application or other client consumer of an event from the enterprise information system.

endpoint listener
: The point or address at which incoming messages for a web service are received by a service integration bus.

enqueue
: To put a message or item in a queue. See also dequeue.

enrollment

1. An entitlement for an organization to subscribe to a business service.
2. The process of entering and saving user or user group information in a portal.

enterprise application
: See Java EE application.

enterprise application project (EAR project)
: A structure and hierarchy of folders and files that contain a deployment descriptor and IBM extension document as well as files that are common to all Java EE modules that are defined in the deployment descriptor.

enterprise archive (EAR)
: A specialized type of JAR file, defined by the Java EE standard, used to deploy Java EE applications to Java EE application servers. An EAR file contains EJB components, a deployment descriptor, and web archive (WAR) files for individual web applications. See also Java archive, web archive.

enterprise bean
: A component that implements a business task or business entity and resides in an EJB container. Entity beans, session beans, and message-driven beans are all enterprise beans. (Sun) See also bean.

enterprise bundle archive
: A compressed file, with a .eba extension, that contains or refers to one or more OSGi bundles that are deployed as one OSGi application. See also bundle.

Enterprise Information Portal
: Software developed by IBM that provides tools for advanced searching, and content customization and summarization.

enterprise information system (EIS)
: The applications that comprise an enterprise's existing system for handling company-wide information. An enterprise information system offers a well-defined set of services that are exposed as local or remote interfaces or both. (Sun) See also resource adapter.

Enterprise JavaBeans (EJB)
: A component architecture defined by Sun Microsystems for the development and deployment of object-oriented, distributed, enterprise-level applications (Java EE).

enterprise service
: A service that typically accesses one or more enterprise information systems.

enterprise service bus (ESB)
: A flexible connectivity infrastructure for integrating applications and services; it offers a flexible and manageable approach to service-oriented architecture implementation.

entity

1. A simple Java class that represents a row in a database table or entry in a map.
2. In markup languages such as XML, a collection of characters that can be referenced as a unit, for example to incorporate often-repeated text or special characters within a document.

entity bean
: In EJB programming, an enterprise bean that represents persistent data maintained in a database. Each entity bean carries its own identity. (Sun) See also session bean.

entry breakpoint
: A breakpoint set on a component element that is hit before the component element is invoked.

envelope

1. A combination of header, trailer, and control segments that define the start and end of an individual EDI message. Each envelope in EDI data begins with a particular segment and ends with a particular segment.
2. A control structure containing documents.

environment
: A named collection of logical and physical resources used to support the performance of a function.

environment variable

1. A variable that provides values for each type of environment in which a process will run (for example, development, test, and production environments). A user can set environment variables for each process application.
2. A variable that defines an aspect of the operating environment for a process. For example, environment variables can define the home directory, the command search path, the terminal in use, or the current time zone.
3. A variable that specifies how an operating system or another program runs, or the devices that the operating system recognizes.

ephemeral port number
: In some TCP/IP implementations, a temporary port number that is assigned to a process for the duration of a call. Ephemeral port numbers are typically assigned to client processes that must provide servers with a client port number so that the server can respond to the correct process.

EPV
: See exposed process value.

error
: A discrepancy between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition.

error end event
: An end event that also throws an error. See also end event.

error event
: An event that indicates that an error has been caught or thrown.

error intermediate event
: An intermediate event that is triggered by a thrown error.

error log stream
: A continuous flow of error information that is transmitted using a predefined format.

error start event
: A start event that is triggered by a thrown error. An error start event is used only for event subprocesses as an error handling mechanism. See also start event.

ESB
: See enterprise service bus.

ESB server
: An application server that provides the execution environment for mediation modules in addition to application programs.

escalation
: A course of action that runs when a task is not completed satisfactorily within a specific period of time.

ESI
: See Edge Side Include.

ESI processor
: A processor that supports fragment caching and fragment assembly into full pages.

ESM
: See external security manager.

ETL
: See extract, transform, and load.

event
: 1. An occurrence of significance to a task or system. Events can include completion or failure of an operation, a user action, or the change in state of a process. See also alert, message, receiver, resource model, situation.
  2. A change to a state, such as the completion or failure of an operation, business process, or human task, that can trigger a subsequent action, such as persisting the event data to a data repository or invoking another business process.
  3. An element that is used to represent a change in state.

event catalog
: A repository of event metadata used by applications to retrieve information about classes of events and their permitted content.

event context
: An activity or group of activities in an expanded subprocess that can be interrupted by an exception (such as by an error intermediate event).

event correlation sphere
: The scope of an ECSEmitter method that allows an event consumer to correlate events. Each event includes the identifier of the correlation sphere to which it belongs and the identifier of its parent correlation sphere from the event hierarchy.

event data
: In an event message, the part of the message data that contains information about the event (such as the queue manager name, and the application that gave rise to the event).

event database
: A database in which events that can be monitored are stored, and which is required to support the persistence of those events.

Event Designer
: An event rule application development tool that is integrated into the Eclipse development environment and dedicated to the creation and management of event rule applications.

event-driven translation
: A translation automatically triggered by the receipt of a document.

event emitter
: A component of the Common Event Infrastructure that receives events from event sources, completes and validates the events, and then sends events to the event server based on filter criteria. See also Common Event Infrastructure, event source.

event flow
: A visual representation of the event processing that will take place when the application is run.

event gateway
: A gateway that represents a branching point in the process where the alternative paths that follow the gateway are based on events that occur rather than the evaluation of expressions using process data (as with an exclusive or inclusive gateway).

event group
: 1. A set of criteria that is applied to events to identify a subset of those events. The criteria include constraints expressions that define the filter conditions.
  2. A container for inbound events that enables the user to group events without having to create a new monitoring context. Event groups are purely a visual construct and are not represented in the monitor model.

event listener
: A type of asynchronous bean that serves as a notification mechanism and through which Java Platform, Enterprise Edition (Java EE) components within a single application can notify each other about various asynchronous events.

event model
: The part of the monitor model that contains references to all of the elements of the event definitions used in the monitor model.

event object
: A subset of the fields in the definition of an event.

event part
: An XML Schema Definition (XSD) type that provides information about the structure of part of an event. A single event definition can have different event parts that are defined by different XML schemas.

event project

A project in which the user can manage and event rules and business objects.

event queue
: An ordered list of events.

event rule
: A piece of business logic that is evaluated by the runtime server when an event is received.

event rule group
: A group of event rules that operate together and typically include an otherwise clause.

event run time
: A shared, secured component that runs event assets such as business objects, events, and actions.

event source
: An object that supports an asynchronous notification server within a single Java virtual machine. Using an event source, the event listener object can be registered and used to implement any interface.

event store
: A persistent cache where event records are saved until a polling adapter can process them.

evictor
: A component that controls the membership of entries in each BackingMap instance. Sparse caches can use evictors to automatically remove data from the cache without affecting the database.

exception

1. An event that occurs during the performance of the process that causes a diversion from the normal flow of the process. Exceptions can be generated by intermediate events, such as time, error, or message.
2. A condition or event that cannot be handled by a normal process.

exception flow
: A set of sequence flow paths that originates from an intermediate event that is attached to the boundary of an activity. The process does not traverse this path unless the activity is interrupted by the triggering of a boundary intermediate event. See also normal flow.

exception handler
: A set of routines that responds to an abnormal condition. An exception handler is able to interrupt and to resume the normal running of processes.

exception queue
: A queue to which messages associated with certain exceptional conditions, such as errors, are routed.

exception report
: A WebSphere MQ report message type that is created by a message channel agent when a message is sent to another queue manager, but that message cannot be delivered to the specified destination queue.

exclusive gateway
: A gateway that creates alternative paths in a process flow. The exclusive gateway indicates the diversion point in the flow of a process.

exclusive lock
: A lock that prevents concurrently executing application processes from accessing database data. See also shared lock.

executable map
: A compiled map.

execution component
: A component that authorizes the execution of a rule set by the execution unit (XU).

execution object model (XOM)
: A model that references implementation objects used in rules.

execution settings
: Settings that influence how a component behaves at execution time. These settings are compiled into the map file or system file. Many of these settings compiled into the map can be overridden (or partially overridden) using execution commands and options.

execution trace
: A chain of events that is recorded and displayed in a hierarchal format on the Events page of the integration test client.

execution unit (XU)
: A Java EE connector or resource adapter that handles the low-level details of rule set execution for a rule execution server.

exit breakpoint
: A breakpoint set on a component element that is hit after the component element is invoked.

exit condition
: A Boolean expression that controls when processing at a process node is completed.

exit zone
: A zone that defines where a tag exits the area. If a tag can no longer be detected within the zone, the item has left the area.

expanded component
: A component that displays the sources and targets that are associated with it in the Integration Flow Designer. See also contracted component.

expanded subprocess
: A subprocess that exposes its flow detail within the context of its parent process. An expanded subprocess is displayed as a rounded rectangle that is enlarged to display the flow objects within.

expected value
: The average value for a given attribute for a population dataset. This value is ultimately used to determine reason code assignment, it is typically used for linear and logistic models where the interaction of variables is controlled.

explicit format
: A format that relies upon syntax to separate data objects. Each data object can be identified by its position or by a delimiter in the data. Delimiters will also appear for missing data objects. See also implicit format.

export
: An exposed interface from a Service Component Architecture (SCA) module that offers a business service to the outside world. An export has a binding that defines how the service can be accessed by service requesters, for example, as a web service.

export file

1. A file created during the development process for inbound operations that contains the configuration settings for inbound processing.
2. The file containing data that has been exported.

exposed process value (EPV)

A variable that enables process participants to set or change a value while an instance of a process is running. Process participants use EPVs to adjust specific variable values as constants, thereby affecting the flow of a process or task assignment.

expression

1. An SQL or XQuery operand or a collection of SQL or XQuery operators and operands that yields a single value.
2. A statement about data objects. Expressions are a combination of literals, object names, operators, functions, and map names. Component rules are expressions that evaluate to either TRUE or FALSE. Map rules are expressions that evaluate to data to produce the desired output.

extended common service area (ECSA)

A major element of z/OS virtual storage above the 16 MB line. This area contains pageable system data areas that are addressable by all active virtual storage address spaces. It duplicates the common system area (CSA) which exists below the 16 MB line.

extended data element

An application-specific element that contains information relevant to an event.

extended messaging

A function of asynchronous messaging where the application server manages the messaging infrastructure and extra standard types of messaging beans are provided to add functionality to that provided by message-driven beans.

Extensible Access Control Markup Language (XACML)

A language used to express policies and rules for controlling access to information.

Extensible Hypertext Markup Language (XHTML)

A reformulation of HTML 4.0 as an application of XML. XHTML is a family of current and future DTDs and modules that reproduce, subset, and extend HTML.

Extensible Markup Language (XML)

A standard metalanguage for defining markup languages that is based on Standard Generalized Markup Language (SGML).

Extensible Stylesheet Language (XSL)

A language for specifying style sheets for XML documents. Extensible Stylesheet Language Transformation (XSLT) is used with XSL to describe how an XML document is transformed into another document.

Extensible Stylesheet Language Transformation (XSLT)

An XML processing language that is used to convert an XML document into another document in XML, PDF, HTML, or other format.

extension

1. In Eclipse, the mechanism that a plug-in uses to extend the platform. See also extension point.
2. A class of objects designated by a specific term or concept; denotation.
3. An element or function not included in the standard language.

extension point

In Eclipse, the specification that defines what attributes and values must be declared by an extension. See also extension.

external command

A command that causes the command-line interface (CLI) to generate a message and send it to a service to be processed.

external link

In the Integration Flow Designer, solid lines displayed in a system definition diagram that visually represent the data flow between two map components.

external partner

A trading community participant that sends business documents to and receives business documents from the internal partner. See also trading partner.

external security manager (ESM)

A security product that performs security checking on users and resources. RACF is an example of an ESM.

extract, transform, and load (ETL)

The process of collecting data from one or more sources, cleansing and transforming it, and then loading it into a database.

eXtreme Scale grid

A pattern that is used to interact with eXtreme Scale when all of the data and clients are in one process.

F

fabric

A complex network of hubs, switches, adapter endpoints, and connecting cables that support a communication protocol between devices. For example, Fibre Channel uses a fabric to connect devices.

Faces component

One of a collection of user interface components (such as input fields) and data components (representing data such as records in a database) that can be dragged to a Faces JSP file and then bound to each other to build a dynamic web project. See also JavaServer Faces.

Faces JSP file

A file that represents a page in a dynamic web project and contains JavaServer Faces UI and data components. See also JavaServer Faces.

factory

In object-oriented programming, a class that is used to create instances of another class. A factory is used to isolate the creation of objects of a particular class into one place so that new functions can be provided without widespread code changes.

failed event

An object that records the source, destination, description, and time of failure between two service connector components.

failover

An automatic operation that switches to a redundant or standby system in the event of a software, hardware, or network interruption.

FastCGI

See Fast Common Gateway Interface Protocol.

Fast Common Gateway Interface Protocol (FastCGI)

An extension of the Common Gateway Interface that improves performance and allows for greater scalability.

Fastpath mode

A rule execution mode that uses an optimized sequential algorithm.

fast response cache accelerator (FRCA)

A cache that resides in the kernel on AIX and Windows platforms that provides support for caching on multiple web servers and on servers with multiple IP addresses.

fast view

In Eclipse, a view that is opened and closed by clicking a button on the shortcut bar.

fault message
: An object that contains status information and details about a problem with a message.

favorite
: A library item that a user has marked for easy access.

feature
: In Eclipse, a JAR file that is packaged in a form that the update manager accepts and uses to update the platform. Features have a manifest that provides basic information about the content of the feature, which can include plug-ins, fragments and other files.

Federal Information Processing Standard (FIPS)
: A standard produced by the National Institute of Standards and Technology when national and international standards are nonexistent or inadequate to satisfy the U.S. government requirements.

federated search
: A search capability that enables searches across multiple search services and returns a consolidated list of search results.

federation
: The process of combining naming systems so that the aggregate system can process composite names that span the naming systems.

federation domain
: A domain that determines the scope over which the federated REST API provides federation support for business processes and human tasks. A federation domain spans one or many BPM environments. See also domain.

feed
: A data format that contains periodically updated content that is available to multiple users, applications, or both. See also Rich Site Summary.

FFDC
: See first-failure data capture.

field
: In object-oriented programming, an attribute or data member of a class.

field constructor
: An element that defines the mapping from an event object to a business object.

FileAct directory
: A directory that is used exclusively to store files that are involved in FileAct transfers.

FileNet P8 domain
: A domain that represents a logical grouping of physical resources and the Content Engine servers that provide access to those resources. Each resource and server belong to only one domain. A server can access any resource in the domain but cannot access any resource that lies outside of the domain.

file serving
: A function that supports the serving of static files by web applications.

file splitting
: The division of an event file, based on a delimiter or based on size, to separate individual business objects within the file and send them as if they are each an event file to reduce memory requirements.

file store
: A type of message store that directly uses files in a file system through the operating system.

File Transfer Protocol (FTP)
: In TCP/IP, an application layer protocol that uses TCP and Telnet services to transfer bulk-data files between machines or hosts.

filter
: 1. A reusable set of conditions that is used in an event rule to evaluate whether an event matches certain criteria.
  2. A device or program that separates data, signals, or material in accordance with specified criteria. See also servlet filtering.

filter expression
: An optional expression, used by a notification receiver to filter the notification instances that it will accept. The receiver is listening for a particular type of notification, and in addition it will only accept notification instances that meet the criteria specified by the filter expression.

FIN
: The SWIFT store-and-forward message-processing service defining message standards and protocols. See also SWIFTNet FIN.

final action
: An action attached to a rule flow task that is performed after the task has been executed.

final score
: An additive function of a continuous collection of weighted attributes.

find
: See discover.

finder method
: In enterprise beans, a method defined in the home interface and invoked by a client to locate an entity bean. (Sun)

fine-grained
: Pertaining to viewing an individual object in detail. See also coarse-grained.

fingerprint
: See digest code.

finite state machine (FSM)
: The theoretical base describing the rules of a service request state and the conditions to state transitions.

FIPS
: See Federal Information Processing Standard.

fire
: In object-oriented programming, to cause a state transition.

firewall
: A network configuration, typically both hardware and software, that prevents unauthorized traffic into and out of a secure network.

first-failure data capture (FFDC)
: A problem diagnosis aid that identifies errors, gathers and logs information about these errors, and returns control to the affected runtime software.

fixed syntax
: A group whose components have a fixed size. Each component is padded to a fixed size or its minimum and maximum content size values are equal.

fix pack

A cumulative collection of fixes that is made available between scheduled refresh packs, manufacturing refreshes, or releases. It is intended to allow customers to move to a specific maintenance level. See also interim fix, program temporary fix, refresh pack.

flat file
A file stored on a local file system, as opposed to a more complex set of files, such as those in a structured database.

floating segment
An EDI segment of an EDI document definition that can exist in many positions relative to other EDI segments.

flow
A directional connector between elements in a process, collaboration, or choreography that represents the overall progression of how a process or process segment is performed. There are two types of flows: sequence flow and message flow.

flow object

1. An object of the business process model that helps connect components in the workflow.
2. A graphical object that can be connected to or from a sequence flow. In a process, flow objects are events, activities, and gateways. In a choreography, flow objects are events, choreography activities, and gateways.

folder

1. A container used to organize objects.
2. A project element that can be used to group rules according to business logic.

foreign bus
A service integration bus with which a particular service integration bus can exchange messages.

foreign key
In a relational database, a key in one table that references the primary key in another table. See also constraint, primary key.

forest
A collection of one or more Windows 2000 Active Directory trees, organized as peers and connected by two-way transitive trust relationships between the root domains of each tree. All trees in a forest share a common schema, configuration, and Global Catalog. When a forest contains multiple trees, the trees do not form a contiguous namespace.

fork

1. A point in the process where one sequence flow path is split into two or more paths that run in parallel within the process, allowing multiple activities to run simultaneously rather than sequentially. BPMN uses multiple outgoing sequence flow paths from activities or events or a parallel gateway to perform a fork.
2. A process element that makes copies of its input and forwards them by several processing paths in parallel.
3. In a rule flow, a node that splits the execution flow into several parallel transitions. The transitions created from a fork do not have conditions.

for loop
A loop that repeats the same sequence of activities a specified number of times.

form
A display screen, printed document, or file with defined spaces for information to be inserted.

form-based login
An authentication process where a user ID and a password are retrieved using an HTML form, and sent to the server over the HTTP or HTTPS protocol.

form bean
In Struts, a class that stores HTML or JSP form data from a submitted client request or that stores input data from a link that a user clicked. The superclass for all form beans is the ActionForm class.

form logout
A mechanism to log out without having to close all web browser sessions.

forward
In Struts, an object that is returned by an action and that has two fields: a name and a path (typically the URL of a JSP). The path indicates where a request is to be sent. A forward can be local (pertaining to a specific action) or global (available to any action).

forwardable credential
A mechanism-specific security credential that is issued to access a resource, which is used to obtain another credential for access to a different resource.

FQDN
See fully qualified domain name.

frame
In hypertext markup language (HTML) coding, a subset of the web browser window.

frameset
An HTML file that defines the layout of a web page that is composed of other, separate HTML files.

FRCA
See fast response cache accelerator.

free float
A period of time in a process flow after a task runs and before the subsequent task can start. Free floats may result from parallel paths in a process that take varying lengths of time to complete.

free-form project
A monitored directory where Java EE artifacts or module files can be created or dropped. As artifacts are introduced or modified in the free-form project, the artifacts are placed in the appropriate Java EE project structures that are dynamically generated in the workspace. The rapid deployment tools generates deployment artifacts required to construct a Java EE-compliant application and deploy that application to a target server. See also monitored directory.

free-form surface
The open area in a visual editor where developers can add and manipulate objects. For example, the Struts application diagram editor provides a free-form surface for representing JSP pages, HTML pages, action mappings, other Struts application diagrams, links from JSP pages, and forwards from action mappings.

FSM
See finite state machine.

FSM instance directory
A directory used by a finite state machine (FSM) to store temporary files, such as shared memory handles and trace files.

FTP

See File Transfer Protocol.

full build
: In Eclipse, a build in which all resources within the scope of the build are considered. See also incremental build.

full deployment
: Deployment of all the data required to set up the resources for an entire instance. See also delta deployment.

fully qualified domain name (FQDN)
: In Internet communications, the name of a host system that includes all of the subnames of the domain name. An example of a fully qualified domain name is rchland.vnet.ibm.com. See also host name.

function
: A named group of statements that can be called and evaluated and can return a value to the calling statement.

functional acknowledgment
: An electronic acknowledgment returned to the sender to indicate acceptance or rejection of EDI documents.

functional acknowledgment map
: A set of mapping instructions that describe how to create an EDI Standard functional acknowledgment. One of three supported map types.

functional group
: One or more documents of a similar type transmitted from the same location and enclosed by functional group header and trailer segments.

function key
: A keyboard key that can be programmed to perform certain actions.

G

garbage collection
: A routine that searches memory to reclaim space from program segments or inactive data.

gate
: An entry to or an exit from an area or zone that is monitored by one device

gate condition
: A condition on a message being processed that must be fulfilled for a mediation policy to apply.

gateway

1. An element that is used to control the divergence and convergence of sequence flow paths in a process and in a choreography.
2. An element that controls the divergence and convergence of sequence lines and determines the branching, forking, merging, and joining of paths that a process can take during execution.
3. See destination.
4. An integration pattern that provides format-independent boundary functions that apply to all incoming messages.
5. A device or program used to connect networks or systems with different network architectures.
6. A middleware component that bridges Internet and intranet environments during web service invocations.

gateway destination
: A type of service destination that receives messages for gateway services. Gateway destinations are divided into those that are used for request processing and those that are used for reply processing.

gateway queue manager
: A cluster queue manager that is used to route messages from an application to other queue managers in the cluster.

gateway service
: A web service that is made available through the web services gateway.

General Inter-ORB Protocol (GIOP)
: A protocol that Common Object Request Broker Architecture (CORBA) uses to define the format of messages.

General System service
: A service that is used to coordinate other services or to manipulate variable data. See also service.

generic object
: An object that is used in API calls and XPATH expressions to refer to concepts, custom entities, or collections. For example, the XPATH expression /WSRR/GenericObject will retrieve all concepts from WebSphere Service Registry and Repository.

generic server
: A server or process, such as a Java server, a C or C++ server or process, a CORBA server, or a Remote Method Invocation (RMI) server, that is managed in the product administrative domain and supports the product environment.

generic server cluster
: A group of remote servers that need routing by the proxy server.

getter method
: A method whose purpose is to get the value of an instance or class variable. This allows another object to find out the value of one of its variables. See also setter method.

GIOP
: See General Inter-ORB Protocol.

global

1. Pertaining to information available to more than one program or subroutine. See also local.
2. Pertaining to an element that is available to any process in the workspace. A global element appears in the project tree and can be used in multiple processes. Tasks, processes, repositories, and services can be either global (referenced by any process in the project) or local (specific to a single process). See also local.

global asset
: A library item that is available to the entire process application in which it is located. For example, environment variables for a process application are global assets and can be called from any implementation.

global attribute
: In XML, an attribute that is declared as a child of the schema element rather than as part of a complex type definition. Global attributes can be referenced in one or more content models using the ref attribute.

global element

In XML, an element that is declared as a child of the schema element rather than as part of a complex type definition. Global elements can be referenced in one or more content models using the ref attribute.

global instance identifier
: A globally unique identifier that is generated either by the application or by the emitter and is used as a primary key for event identification.

global security
: Pertains to all applications running in the environment and determines whether security is used, the type of registry used for authentication, and other values, many of which act as defaults.

global transaction
: A recoverable unit of work performed by one or more resource managers in a distributed transaction environment and coordinated by an external transaction manager.

global transaction management (GTX)
: The monitoring of transactions that can include operations on two or more different data sources. This feature enables databases or servers to be returned to a pre-transaction state if an error occurs. Either all databases and servers are updated or none are. The advantage of this strategy is that databases and servers remain synchronized and data remains consistent.

global variable
: A variable that is used to hold and manipulate values assigned to it during translation and that is shared across maps and across document translations. One of the three types of variables supported by the Data Interchange Services mapping command language.

Globus certificate service
: An online service that issues low-quality GSI certificates for people who want to experiment with Grid (or distributed) computing components that require certificates but have no other means to acquire certificates. The Globus certificate service is not a true CA. Certificates from the Globus certificate service are intended solely for experimentation. Use caution when using these certificates, for they are not intended for use in production systems. See also certificate authority.

glue code
: A segment of code that is used to connect two pre-existing pieces of code and retain full functionality. See also API stub.

GMT
: See Greenwich mean time.

governance
: The decision making processes in the administration of an organization. The rights and responsibilities of these processes are typically shared among the organization's participants, especially the management and stakeholders.

governance lifecycle
: A life cycle that represents the states and transitions that can exist in SOA deployment.

governance policy validator
: A sample validator that enables the user to control the operations that can be performed on specific entities based on the metadata that is attached to those entities.

governance process
: A process that ensures that compliance and operational polices are enforced, and that change occurs in a controlled fashion and with appropriate authority as envisioned by the business design.

governance state
: A state defined within the governance life cycle, for example, "created", "planned", or "specified".

governance web service
: A service that retrieves information and runs actions, relating to the governance of objects, from a web service client.

governed collection
: Group of objects on which an operation may be performed automatically, as a result of an initial operation.

governed entity
: Controls visibility of artifacts as well as controlling who can perform which actions on specific governed entities.

GPM
: See Graphical Process Modeler.

grammar
: A document type definition (DTD) or schema providing a structured format used for successful processing by the trace service.

Graphical Process Modeler (GPM)
: A stand-alone graphical interface tool that is used in Sterling B2B Integrator to create and modify business processes. The GPM converts the graphical representation of business processes to well-formed BPML (source code) and saves the effort of writing code.

Greenwich mean time (GMT)
: The mean solar time at the meridian of Greenwich, England.

group
: 1. A set of elements that is associated with the same category.
2. A complex data object that consists of components.
3. A collection of users who can share access authorities for protected resources.

GTX
: See global transaction management.

H

HA
: See high availability.

HADR
: See high availability disaster recovery.

HA group
: A collection of one or more members used to provide high availability for a process.

handle
: In the Java EE specification, an object that identifies an enterprise bean. A client may serialize the handle, and then later deserialize it to obtain a reference to the enterprise bean. (Sun)

handler

In web services, a mechanism for processing service content and extending the function of a JAX-RPC runtime system.

handshake
: The exchange of messages at the start of a Secure Sockets Layer session that allows the client to authenticate the server using public key techniques (and, optionally, for the server to authenticate the client) and then allows the client and server to cooperate in creating symmetric keys for encryption, decryption, and detection of tampering.

HA policy
: A set of rules that is defined for an HA group that dictate whether zero (0), or more members are activated. The policy is associated with a specific HA group by matching the policy match criteria with the group name.

hash
: In computer security, a number generated from a string of text that is used to ensure that transmitted messages arrived intact.

hashed method authentication code (HMAC)
: A mechanism for message authentication that uses cryptographic hash functions.

header
: The portion of a message that contains control information.

headless
: Pertains to a program or application that can run without a graphical user interface or, in some cases, without any user interface at all. Headless operation is often used for network servers or embedded systems.

health
: The general condition or state of the database environment.

health controller
: An autonomic manager that constantly monitors defined health policies. When a specified health policy condition does not exist in the environment, the health controller verifies that configured actions correct the error.

Health Insurance Portability and Accountability Act
: A legislative act in the U.S. that requires health plans and providers to use a common format when electronically communicating health information.

health policy
: A set of rules that an administrator can define and use to monitor conditions and take actions when the conditions occur.

heap
: In Java programming, a block of memory that the Java virtual machine (JVM) uses at run time to store Java objects. Java heap memory is managed by a garbage collector, which automatically de-allocates Java objects that are no longer in use.

heartbeat
: A signal that one entity sends to another to convey that it is still active.

HFS
: See hierarchical file system.

hidden widget
: A fully functional widget that transforms business data so that another widget can use this data. A hidden widget is not displayed on a page, unless all widgets are displayed. When a hidden widget is made visible, the widget has a dashed frame.

hierarchical
: Pertaining to data that is organized on computer systems using a hierarchy of containers, often called folders (directories) and files. In this scheme, folders can contain other folders and files. The successive containment of folders within folders creates the levels of organization, which is the hierarchy.

hierarchical file system (HFS)
: A system for organizing files in a hierarchy, as in a UNIX system.

hierarchical loop (HL)
: A technique for describing the relationship of data entities which are related in a parent to child manner, like a corporate organization chart.

hierarchical property
: An extended rule property whose values are organized into a hierarchy.

high availability
: 1. The ability of IT services to withstand all outages and continue providing processing capability according to some predefined service level. Covered outages include both planned events, such as maintenance and backups, and unplanned events, such as software failures, hardware failures, power failures, and disasters.
  2. Pertaining to a clustered system that is reconfigured when node or daemon failures occur so that workloads can be redistributed to the remaining nodes in the cluster.

high availability disaster recovery (HADR)
: A disaster recovery solution that uses log shipping and provides data to a standby system if a partial or complete site failure occurs on a primary system.

high availability file system
: A cluster file system that can be used for component redundancy to provide continued operations during failures.

high availability manager
: A framework within which core group membership is determined and status is communicated between core group members.

high-level qualifier (HLQ)
: A qualifier that groups tables together with other tables that have different names, but the same qualifier.

HL
: See hierarchical loop.

HLQ
: See high-level qualifier.

HMAC
: See hashed method authentication code.

home interface
: In enterprise beans, an interface that defines zero or more create and remove methods for a session bean or zero or more create, finder, and remove methods for an entity bean. See also remote interface.

home method
: A method in the home interface that is used by a client to create, locate, and remove instances of enterprise beans.

home page
: The top-level web page of a portal.

homogeneous rule
: A rule for which the conditions are written on the same type and number of objects.

hook
: A location in a compiled program where the compiler has inserted an instruction that allows programmers to interrupt the program (by setting breakpoints) for debugging purposes.

horizontal scaling
: A topology in which more than one application server running on multiple computing nodes is used to run a single application.

host
: 1. A computer that is connected to a network and that provides an access point to that network. The host can be a client, a server, or both a client and server simultaneously. See also client, server.
2. In performance profiling, a machine that owns processes that are being profiled. See also server.

host name
: 1. In Internet communication, the name given to a computer. The host name might be a fully qualified domain name such as mycomputer.city.company.com, or it might be a specific subname such as mycomputer. See also fully qualified domain name, IP address.
2. The network name for a network adapter on a physical machine in which the node is installed.

host system
: An enterprise mainframe computer system that hosts 3270 applications. In the 3270 terminal service development tools, the developer uses the 3270 terminal service recorder to connect to the host system.

hot deployment
: The process of adding new components to a running server without stopping and restarting the application server or application. See also dynamic reloading.

hot directory
: See monitored directory.

hot servant region
: A servant region that had a request dispatched to it previously and now has available threads.

HTTP channel
: A type of channel within a transport chain that provides client applications with persistent HTTP connections to remote hosts that are either blocked by firewalls or require an HTTP proxy server. An HTTP channel is used to exchange application data in the body of an HTTP request and an HTTP response that is sent to and received from a remote server.

HTTP over SSL (HTTPS)
: A web protocol for secure transactions that encrypts and decrypts user page requests and pages returned by the web server.

HTTPS
: 1. See HTTP over SSL.
2. See Hypertext Transfer Protocol Secure.

hub administrator
: The superuser who configures the hub and who has the ability to perform all the tasks associated with setting up and administering the hub.

human task
: An interaction between people and business processes or services. See also inline task, stand-alone task.

human workflow
: A business process flow that includes human interactions.

Hypertext Transfer Protocol Secure (HTTPS)
: An Internet protocol that is used by web servers and web browsers to transfer and display hypermedia documents securely across the Internet.

hypervisor
: A program or a portion of Licensed Internal Code that allows multiple instances of operating systems to run simultaneously on the same hardware.

I

IAMS
: See inbound application message store.

ICAP
: See Internet Content Adaptation Protocol.

ICMP
: See Internet Control Message Protocol.

IDE
: See integrated development environment.

identifier
: 1. In the 3270 terminal services development tool, a field on a screen definition that uniquely identifies the state of the screen. Users can choose which fields will be identifiers when creating recognition profiles.
2. The name of an item in a program written in the Java language.

identifier attribute
: An attribute that can be assigned to one component to identify a collection of components, when creating type trees and defining components of a group. An identifier attribute is used during data validation to determine whether a data object exists.

identity
: The data that represents a person and that is stored in one or more repositories.

identity assertion
: The invocation credential that is asserted to the downstream server. This credential can be set as the originating client identity, the server identity, or another specified identity, depending on the RunAs mode for the enterprise bean.

identity token
: A token that contains the invocation credential identity, which with the client authentication token are required by the receiving server to accept the asserted identity.

IDL

See Interface Definition Language.

if-then rule
: A rule in which the action (then part) is performed only when the condition (if part) is true. See also action rule, rule set.

IIOP
: See Internet Inter-ORB Protocol.

ILOG Rule Language (IRL)
: An executable rule language. Rules in ILOG Rule Language (IRL) can reference any execution object and invoke methods on these objects.

i-mode
: An Internet service for wireless devices.

implicit format
: A format that defines a group type whose data objects are distinguishable by content, not syntax. Implicit format relies on the properties of the component types. Unlike explicit format, if delimiters separate data objects, they do not appear for missing data objects. See also explicit format.

import

1. The point at which an SCA module accesses an external service, (a service outside the SCA module) as if it was local. An import defines interactions between the SCA module and the service provider. An import has a binding and one or more interfaces.
2. A development artifact that imports a service that is external to a module. See also import file.

import file
: A file created during the development process for outbound operations that contains the configuration settings for outbound processing. See also import.

IMS
: See Information Management System.

IMS command
: A request from a terminal or AO (automated operator) to perform a specific IMS service, such as altering system resource status or displaying specific system information.

IMS Connect
: The product that runs on a z/OS platform and through which IMS Connector for Java communicates with IMS. IMS Connect uses OTMA to communicate with IMS. See also Open Transaction Manager Access.

IMS conversation

1. In IMS Connector for Java, the dialog between a Java client program and a message processing program.
2. A dialog between a terminal and a message processing program using IMS conversational processing facilities. See also conversational processing.

IMS transaction
: A specific set of input data that triggers the execution of a specific process or job. A transaction is a message destined for an IMS application program.

IMS transaction code
: A 1- to 8-character alphanumeric code that invokes an IMS message processing program.

inbound
: In communication, pertaining to data that is received from the network. See also outbound.

inbound application message store (IAMS)
: A message store, implemented by means of the database table DNF_IAMS, in which WebSphere BI for FN stores messages that are received from remote destinations (OSN messages).

inbound authentication
: The configuration that determines the type of accepted authentication for inbound requests.

inbound document
: See source document.

inbound event
: A declaration that a monitoring context or KPI context will accept a specific event at run time.

inbound port
: A type of port that takes a message that is received at an endpoint listener and passes it to the service integration bus for forwarding to the appropriate inbound service.

inbound processing
: The process by which changes to business information in an enterprise information system (EIS) are detected, processed, and delivered to a runtime environment by a JCA Adapter. An adapter can detect EIS changes by polling an event table or by using an event listener.

inbound service
: The external interface for a service that is provided by your own organization and hosted in a location that is directly available through the service destination.

inbound transport
: Network ports in which a server listens for incoming requests.

inclusive gateway
: A gateway that creates alternative or parallel paths in a process flow where all outgoing sequence flow condition expressions are evaluated independently.

incremental build
: In Eclipse, a build in which only resources that have changed since the last build are considered. See also full build.

index
: A set of pointers that is logically ordered by the values of a key. Indexes provide quick access to data and can enforce uniqueness of the key values for the rows in the table.

individual resource
: A single resource that can be uniquely identified, such as a person or computer. Individual resources are used when a specific resource must be allocated to a task. For example, the Mary Smith resource must perform the Approve Payment task.

industry capability map
: A logical view of the business competencies that an industry needs to process.

information center
: A collection of information that provides support for users of one or more products, can be launched separately from the product, and includes a list of topics for navigation and a search engine.

Information Management System (IMS)

Any of several system environments that have a database manager and transaction processing that can manage complex databases and terminal networks.

inheritance
: An object-oriented programming technique in which existing classes are used as a basis for creating other classes. Through inheritance, more specific elements incorporate the structure and behavior of more general elements.

initial action
: An action attached to a rule flow task that is performed before the task is executed.

initial CDD
: A customization definition document (CDD) to which placeholders have not yet been added.

initial context
: Starting point in a namespace.

initialization point
: A user-defined constant or variable used to initialize the attributes of an object.

initial option set
: For a scenario that uses an option set group, the first option set that the scenario used. The initial option set is used to determine when all of the option sets of an option set group have been used at least once.

initial reference
: A well-known reference associated with an identifier.

initiator
: A syntax object in a data stream that signals the beginning of a data object. For example, if a record begins with an asterisk (*), the asterisk would be the record's initiator.

inline
: In Content Manager, the property of an object that is online and in a drive, but has no active mounts.

inline schema
: An XML schema in a Web Service Description Language file (.wsdl).

inline task
: A unit of work that is defined within an implementation of a business process. See also human task, stand-alone task.

inout parameter
: A parameter value that is provided as input to a rule set when it is executed. It can be modified by the execution process and is provided as output when the execution is completed.

in parameter
: A parameter value that is provided as input to the rule set at execution time.

input
: An entry point through which an element is notified that it can start, typically because an upstream element, on which it depends, has completed. If the element has all of its required input, then it will start.

input activity
: The origin of the process that is the source of the invocation data of the entire process.

input branch
: The area of a decision, fork, join, or merge that contains the inputs.

input card
: In the Map Designer, a component that contains the complete definition of input for the map, including information such as source identification, retrieval specifics, and the behavior that should occur during processing.

input criteria
: Number and types of inputs required to start a task or process.

input node
: 1. A message flow node that represents a source of messages for a message flow or subflow.
2. The point where a service message from a source enters the request flow.

input response node
: The end point for a mediation response flow from which the service message object is sent to the source.

input terminal node
: A primitive through which a message is received by a subflow. Each input terminal node is represented as an input terminal of the corresponding subflow node.

INS
: See Interoperable Naming Service.

insertion
: The action of adding a new object into the object set provided to the rule engine for execution.

installation image
: A copy of the software, in backup format, that the user is installing, as well as copies of other files the system needs to install the software product.

installation package
: An installable unit of a software product. Software product packages are separately installable units that can operate independently from other packages of that software product.

installation target
: The system on which selected installation packages are installed.

instance
: 1. A specific occurrence of an object that belongs to a class. See also object.
2. A set of servers that share a common runtime database, plus their corresponding brokers and queue managers.
3. An active process element, for example, the performance of a process.

instance document
: An XML document that conforms to a particular schema.

instance metric
: A metric that returns the result, such as the amount of an order, from one run of the process. See also metric.

instantiate

To represent an abstraction with a concrete instance.

integrated development environment (IDE)
A set of software development tools, such as source editors, compilers, and debuggers, that are accessible from a single user interface.

integration broker
A component that integrates data among heterogeneous applications. An integration broker typically provides various services that can route data, as well as a repository of rules that govern the integration process, connectivity to various applications, and administrative capabilities that facilitate integration.

integration service
A service that is used to integrate with an external system, such as a web service. See also Advanced Integration service, service.

intelligent page
A page that is based on platform capabilities that deliver a unified presentation and architecture, rapid assembly of multiple component types including feeds, widgets, and portlets, and rich media that provide access to dynamic web pages, enabling real-time web-page analysis and channel-delivery analysis.

interaction
A definition that explains what the target document should be. An interaction consists of the source document, target document, action, and a transformation map.

interaction endpoint
A service requester or provider.

interaction pattern
A communication method for sending or receiving messages in a service interaction. Examples of interaction patterns include request/reply, one-way interaction, and publish/subscribe.

interactive process design
The development of deployable processes through modeling, testing, and revision by business users.

interactive session
A work session in which there is an exchange of communication between a 3270 application and the 3270 terminal service recorder.

Interactive System Productivity Facility (ISPF)
An IBM licensed program that serves as a full-screen editor and dialog manager. Used for writing application programs, it provides a means of generating standard screen panels and interactive dialogs between the application programmer and the terminal user. See also Time Sharing Option.

interactive view
In 3270 terminal services, real-time access to a host application in the 3270 terminal service recorder editor.

interchange
The exchange of information between trading partners. Also a set of documents grouped together, such as EDI documents enclosed within an EDI envelope.

interface
A collection of operations that are used to specify a service of a class or a component. See also class, port type.

Interface Definition Language (IDL)
In CORBA, a declarative language that is used to describe object interfaces, without regard to object implementation.

interface map
A map that resolves and reconciles the differences between the interfaces of interacting components. There are two levels of interface maps: operation mappings and parameter mappings.

interim fix
A certified fix that is generally available to all customers between regularly scheduled fix packs, refresh packs, or releases. See also fix pack, refresh pack.

intermediate CDD
A customization definition document (CDD) to which placeholders have been added, but for which placeholder values have not yet been specified.

intermediate event
An event that occurs after a process has been started, affecting the flow of the process by showing where messages and delays are expected and distributing the normal flow through exception handling. Intermediate event types are message, timer, tracking, and error. See also catching message intermediate event, message intermediate event, throwing message intermediate event, timer event, tracking intermediate event.

internal command
A command that is processed directly by and that controls the command-line interface (CLI).

internal link
In the Integration Flow Designer, a solid line displayed by an expanded map component that visually represents the source and target of the map.

internal partner
A company that acts as the hub community for its partners. The internal partner has one administrative user and the manager administrator who is responsible for the health and maintenance of the internal partner's portion of the community.

internal rate of return (IRR)
The interest rate received for an investment, based on anticipated expenses and income that will occur at regular periods

Internet Content Adaptation Protocol (ICAP)
A high-level protocol for requesting services from an Internet-based server.

Internet Control Message Protocol (ICMP)
An Internet protocol that is used by a gateway to communicate with a source host, for example, to report an error in a datagram.

Internet Inter-ORB Protocol (IIOP)
A protocol used for communication between Common Object Request Broker Architecture (CORBA) object request brokers. See also Common Object Request Broker Architecture.

Internet Protocol (IP)
A protocol that routes data through a network or interconnected networks. This protocol acts as an intermediary between the higher protocol layers and the physical network. See also Transmission Control Protocol.

interoperability
The ability of a computer or program to work with other computers or programs.

Interoperable Naming Service (INS)
A program that supports the configuration of the Object Request Broker (ORB) administratively to return object references.

interoperable object reference (IOR)
An object reference with which an application can make a remote method call on a CORBA object. This reference contains all the information needed to route a message directly to the appropriate server.

interrupt
A condition that applies to a simulation that causes the simulation execution to be halted if the condition is met.

interval
: The defining range or ranges of an attributes value. Intervals are a one-to-many relationship to the attributes utilized in the scorecard.

introspector
: In Java, a class (java.beans.Introspector) that provides a standard way for tools to learn about the properties, events, and methods supported by a target bean. Introspectors follow the JavaBeans specification.

invocation
: The activation of a program or procedure.

invocation credential
: An identity with which to invoke a downstream method. The receiving server requires this identity with the sending server identity to accept the asserted identity.

invoker attribute
: An assembly property for a web module that is used by the servlet that implements the invocation behavior.

IOR
: See interoperable object reference.

IP
: See Internet Protocol.

IP address
: A unique address for a device or logical unit on a network that uses the Internet Protocol standard. See also host name.

IP group
: A range of IP addresses that can be selected for use with specific hypervisors.

IP sprayer
: A device that is located between inbound requests from the users and the application server nodes that reroutes requests across nodes.

IRL
: See ILOG Rule Language.

IRR
: See internal rate of return.

ISPF
: See Interactive System Productivity Facility.

item
: 1. A simple data object that does not consist of other objects. An item type is represented by a blue dot next to the type name in the type tree.
  2. An entity within a location that can be equipped with tags and whose positions can therefore be tracked, such as an asset or person.

iteration
: See loop.

iterator
: A class or construct that is used to step through a collection of objects one at a time.

iWidget
: A browser-oriented component, potentially extending a server-side component, that provides either a logical service to the page or a visualization for the user (typically related to a server-side component or a configured data source).

iWidget specification
: An open-source specification upon which Business Space widgets are based.

J

J2C
: See J2EE Connector architecture.

J2EE
: 1. See Java 2 Platform, Enterprise Edition.
  2. See Java Platform, Enterprise Edition.

J2EE Connector architecture (J2C)
: See Java EE Connector Architecture.

J2SE
: See Java 2 Platform, Standard Edition.

JAAS
: See Java Authentication and Authorization Service.

JAF
: See JavaBeans Activation Framework.

JAR
: See Java archive.

JAR file
: A Java archive file. See also Java archive.

JASPI
: See Java Authentication for SPI for containers.

Java
: An object-oriented programming language for portable interpretive code that supports interaction among remote objects. Java was developed and specified by Sun Microsystems, Incorporated.

Java 2 Connector security
: See Java Connector security.

Java 2 Platform, Enterprise Edition (J2EE)
: See Java Platform, Enterprise Edition.

Java 2 Platform, Standard Edition (J2SE)
: See Java Platform, Standard Edition.

Java API for XML (JAX)

A set of Java-based APIs for handling various operations involving data defined through Extensible Markup Language (XML).

Java API for XML-based RPC (JAX-RPC, JSR 101)
A specification that describes application programming interfaces (APIs) and conventions for building web services and web service clients that use remote procedure calls (RPC) and XML.

Java API for XML Web Services (JAX-WS)
The next-generation web services programming model that is based on dynamic proxies and Java annotations.

Java Architecture for XML Binding (JAXB)
A Java binding technology that supports transformation between schema and Java objects, as well as between XML instance documents and Java object instances.

Java archive (JAR)
A compressed file format for storing all of the resources that are required to install and run a Java program in a single file. See also enterprise archive, JAR file, web archive.

Java Authentication and Authorization Service (JAAS)
In Java EE technology, a standard API for performing security-based operations. Through JAAS, services can authenticate and authorize users while enabling the applications to remain independent from underlying technologies.

Java Authentication for SPI for containers (JASPI)
A specification that supports third-party security providers handling the Java Platform, Enterprise Edition (Java EE) authentication of HTTP request and response messages that are sent to web applications.

JavaBeans
As defined for Java by Sun Microsystems, a portable, platform-independent, reusable component model. See also bean.

JavaBeans Activation Framework (JAF)
A standard extension to the Java platform that determines arbitrary data types and available operations and can instantiate a bean to run pertinent services.

Java class
A class that is written in the Java language.

Java Command Language
A scripting language for the Java environment that is used to create web content and to control Java applications.

Java Connector security
An architecture designed to extend the end-to-end security model for Java EE-based applications to include enterprise information systems (EIS).

Java Database Connectivity (JDBC)
An industry standard for database-independent connectivity between the Java platform and a wide range of databases. The JDBC interface provides a call level interface for SQL-based and XQuery-based database access. See also Open Database Connectivity.

Java Development Kit (JDK)
See Java SE Development Kit.

Javadoc

1. Pertaining to the tool that parses the declarations and documentation comments in a set of source files and produces a set of HTML pages describing the classes, inner classes, interfaces, constructors, methods, and fields.
2. A tool that parses the declarations and documentation comments in a set of source files and produces a set of HTML pages describing the classes, inner classes, interfaces, constructors, methods, and fields. (Sun)

Java EE
See Java Platform, Enterprise Edition.

Java EE application
Any deployable unit of Java EE functionality. This unit can be a single module or a group of modules packaged into an enterprise archive (EAR) file with a Java EE application deployment descriptor. (Sun)

Java EE Connector Architecture (JCA)
A standard architecture for connecting the Java EE platform to heterogeneous enterprise information systems (EIS).

Java EE server
A runtime environment that provides EJB or web containers.

Java file
An editable source file (with .java extension) that can be compiled into bytecode (a .class file).

JavaMail API
A platform and protocol-independent framework for building Java-based mail client applications.

Java Management Extensions (JMX)
A means of doing management of and through Java technology. JMX is a universal, open extension of the Java programming language for management that can be deployed across all industries, wherever management is needed.

Java Message Service (JMS)
An application programming interface that provides Java language functions for handling messages.

Java Naming and Directory Interface (JNDI)
An extension to the Java platform that provides a standard interface for heterogeneous naming and directory services.

Java Persistence API
An object/relational mapping facility to Java developers for managing relational data in Java applications.

Java platform
A collective term for the Java language for writing programs; a set of APIs, class libraries, and other programs used in developing, compiling, and error-checking programs; and a Java virtual machine which loads and runs the class files. (Sun)

Java Platform, Enterprise Edition (J2EE, Java EE)
An environment for developing and deploying enterprise applications, defined by Sun Microsystems Inc. The Java EE platform consists of a set of services, application programming interfaces (APIs), and protocols that provide the functionality for developing multitiered, web-based applications. (Sun)

Java Platform, Standard Edition (Java SE)
The core Java technology platform. (Sun)

Java project
In Eclipse, a project that contains compilable Java source code and is a container for source folders or packages.

Java project for Rules
A predefined Java project for Eclipse that contains a single, runnable main class to execute rules contained in a rule project.

Java runtime environment (JRE)

A subset of a Java developer kit that contains the core executable programs and files that constitute the standard Java platform. The JRE includes the Java virtual machine (JVM), core classes, and supporting files.

JavaScript
: A web scripting language that is used in both browsers and web servers. (Sun)

JavaScript Object Notation (JSON)
: A lightweight data-interchange format that is based on the object-literal notation of JavaScript. JSON is programming-language neutral but uses conventions from languages that include C, C++, C#, Java, JavaScript, Perl, Python.

Java SE
: See Java Platform, Standard Edition.

Java Secure Socket Extension (JSSE)
: A Java package that enables secure Internet communications. It implements a Java version of the Secure Sockets Layer (SSL) and Transport Layer Security (TSL) protocols and supports data encryption, server authentication, message integrity, and optionally client authentication.

Java SE Development Kit
: The name of the software development kit that Sun Microsystems provides for the Java platform.

JavaServer Faces (JSF)
: A framework for building web-based user interfaces in Java. Web developers can build applications by placing reusable UI components on a page, connecting the components to an application data source, and wiring client events to server event handlers. See also Faces component, Faces JSP file, JavaServer Pages.

JavaServer Pages (JSP)
: A server-side scripting technology that enables Java code to be dynamically embedded within web pages (HTML files) and run when the page is served, in order to return dynamic content to a client. See also JavaServer Faces, JSP file, JSP page.

Java Specification Request (JSR)
: A formally proposed specification for the Java platform.

Java virtual machine (JVM)
: A software implementation of a processor that runs compiled Java code (applets and applications).

Java virtual machine Profiler Interface (JVMPI)
: A profiling tool that supports the collection of information, such as data about garbage collection and the Java virtual machine (JVM) API that runs the application server.

JAX
: See Java API for XML.

JAXB
: See Java Architecture for XML Binding.

JAX-RPC
: See Java API for XML-based RPC.

JAX-WS
: See Java API for XML Web Services.

JCA
: See Java EE Connector Architecture.

JCA contract
: A collaborative agreement between an application server and an EIS system-level. A JCA contract indicates how to keep all mechanisms (for example, transactions, security, and connection management) transparent from the application components.

JCL
: See job control language.

JDBC
: See Java Database Connectivity.

JDBC connection filter
: A control that limits the amount of data that is transferred during the JDBC metadata load. The filter enhances performance.

JDK
: See Java Development Kit.

Jetspeed
: The open-source portal that is part of the Jakarta project by Apache.

JMS
: See Java Message Service.

JMS data binding
: A data binding that provides a mapping between the format used by an external JMS message and the Service Data Object (SDO) representation used by a Service Component Architecture (SCA) module.

JMS destination
: An object in which message queuing applications use the Java Message Service specification to put messages, and from which they can get messages.

JMS provider
: A messaging engine that implements the JMS messaging specification, for example WebSphere MQ or SIBus.

JMX
: See Java Management Extensions.

JNDI
: See Java Naming and Directory Interface.

job class
: Any one of a number of job categories that can be defined.

job control language (JCL)
: A command language that identifies a job to an operating system and describes the job requirements. See also xJCL.

job group security
: A security model in which groups of users can access and control a common set of jobs owned by that group.

job log
: A record of requests submitted to the system by a job, the messages related to the requests, and the actions performed by the system on the job. The job log is maintained by the system program.

job management console
: A stand-alone web interface that is used to submit, monitor, view, and manage jobs.

job manager
: An administrative process that manages multiple base application servers or network deployment cells.

job scheduler
: A component that provides all job-management functions. A job scheduler maintains a history of all jobs and usage data for jobs that have run.

job step
: The execution of a computer program explicitly identified by a job control statement. A job may specify that several job steps be executed. [A]

join
: 1. In a rule flow, a node that combines all the transitions created from a fork.
  2. A point in the process where two or more parallel sequence flow paths are combined into one sequence flow path. BPMN uses a parallel gateway to perform a join.
  3. An SQL relational operation in which data can be retrieved from two tables, typically based on a join condition specifying join columns.
  4. A process element that recombines and synchronizes parallel processing paths after a decision or fork. A join waits for input to arrive at each of its incoming branches before permitting the process to continue.

join condition
: A condition that determines whether to run the next activity.

join failure
: A fault that is thrown if a join condition cannot be evaluated.

JRas
: A toolkit that consists of a set of Java packages that enable developers to incorporate message logging and trace facilities into Java applications.

JRE
: See Java runtime environment.

JSF
: See JavaServer Faces.

JSON
: See JavaScript Object Notation.

JSP
: See JavaServer Pages.

JSP file
: A scripted HTML file that has a .jsp extension and allows for the inclusion of dynamic content in web pages. A JSP file can be directly requested as a URL, called by a servlet, or called from within an HTML page. See also JavaServer Pages, JSP page.

JSP page
: A text-based document using fixed template data and JSP elements that describes how to process a request to create a response. (Sun) See also JavaServer Pages, JSP file.

JSR
: See Java Specification Request.

JSR 101
: See Java API for XML-based RPC.

JSSE
: See Java Secure Socket Extension.

junction
: A logical connection created to establish a path from one server to another.

JUnit
: An open-source regression testing framework for unit-testing Java programs.

JVM
: See Java virtual machine.

JVMPI
: See Java virtual machine Profiler Interface.

Jython
: An implementation of the Python programming language that is integrated with the Java platform.

K

kernel
: The part of an operating system that contains programs for such tasks as input/output, management and control of hardware, and the scheduling of user tasks.

key
: 1. Information that characterizes and uniquely identifies the real-world entity that is being tracked by a monitoring context.
  2. A cryptographic mathematical value that is used to digitally sign, verify, encrypt, or decrypt a message. See also private key, public key.

key class
: In EJB query language, a class that is used to create or find an entity bean. It represents the identity of the entity bean, corresponding to the primary-key columns of a row in a relational database.

key database
: In security, a storage object, either a file or a hardware cryptographic card, where identities and private keys are stored for authentication and encryption purposes. Some key databases also contain public keys. See also stash file.

key database file
: See key ring.

Keyed-Hashing Message Authentication Code
: A mechanism for message authentication that uses cryptographic hash functions.

key field
: In EJB query language, a container-managed field in an entity bean that corresponds to one of the primary-key columns of a row in a relational database. Each key field is a member of the entity bean key class.

key file

See key ring.

key locator
: A mechanism that retrieves the key for XML signing, XML digital signature verification, XML encryption, and XML decryption.

key pair
: In computer security, a public key and a private key. When the key pair is used for encryption, the sender uses the public key to encrypt the message, and the recipient uses the private key to decrypt the message. When the key pair is used for signing, the signer uses the private key to encrypt a representation of the message, and the recipient uses the public key to decrypt the representation of the message for signature verification.

key performance indicator (KPI)
: A quantifiable measure that is designed to track one of the critical success factors of a business process.

key ring
: In computer security, a file that contains public keys, private keys, trusted roots, and certificates.

keystore
: In security, a file or a hardware cryptographic card where identities and private keys are stored, for authentication and encryption purposes. Some keystores also contain trusted, or public, keys. See also certificate signing request, truststore.

keystring
: Additional specification of the entry within a naming service.

key value pair
: Information that is expressed as a paired set of parameters. For example, if you want to express that the specific sport is football, this data can be expressed as key=sport and value=football.

keyword
: One of the predefined words of a programming language, artificial language, application, or command. See also parameter.

keyword parameter
: A parameter that consists of a keyword followed by one or more values.

knowledge asset
: A document external to the scope of the product that contains information associated to existing metadata.

KPI
: See key performance indicator.

KPI context
: A container for key performance indicators (KPIs) and their associated triggers and events.

KPI model
: The part of the monitor model that contains the KPI contexts, which in turn contain key performance indicators and their associated triggers and events.

L

label
: A node in a portal that cannot contain any content, but can contain other nodes. Labels are used primarily to group nodes in the navigation tree.

lane
: A container in a pool for the activities and events that take place during process execution. A lane is designated by a user and typically represents departments in a business organization. For example, a Call Center lane would include all activities to be handled by Call Center personnel during process execution. See also pool.

language code
: A two character (ISO 639-1) or three letter (ISO 639-2) abbreviation for a language. For example: en or eng for English. Country codes and language codes together form the basis for locale names.

large object (LOB)
: A data object whose data type supports the storage and manipulation of more data than most other data types.

late bind
: To connect one process to another process so that the connection is resolved dynamically in the runtime environment and the calling process uses the currently valid version of the process that it is invoking.

late binding
: The connection between two processes that is resolved dynamically in the runtime environment. As a result, the calling process uses up the currently valid version of the process that it is invoking.

LAU
: See local authentication.

launch configuration
: A mechanism for defining and saving different workbench configurations that can be launched separately. Configurable options include run and debug settings.

launchpad
: A graphical interface for launching the product installation wizard.

layout box
: In Page Designer, a control that web designers can use to move text and images within the page. Layout boxes can be stacked or aligned by using a grid.

layout manager
: In programming graphical user interfaces, an object that controls the size and position of Java components within a container. The Java platform supplies several commonly used layout managers for AWT and Swing containers.

lazy authentication
: The process whereby the security run time environment obtains the required authentication data when the Java client accesses a protected enterprise bean for the first time.

LDAP
: See Lightweight Directory Access Protocol.

LDAP directory
: A type of repository that stores information on people, organizations, and other resources and that is accessed using the LDAP protocol. The entries in the repository are organized into a hierarchical structure, and in some cases the hierarchical structure reflects the structure or geography of an organization.

leaf
: In a tree, an entry or node that has no children.

library

1. In Business Process Management, a project that is used for the development, version management, and organization of shared resources. Only a subset of the artifact types can be created and stored in a library, such as business objects and interfaces. See also project.
2. A collection of model elements, including business items, processes, tasks, resources, and organizations.

lifecycle
: One complete pass through the four phases of software development: inception, elaboration, construction and transition.

Lightweight Directory Access Protocol (LDAP)
: An open protocol that uses TCP/IP to provide access to directories that support an X.500 model and that does not incur the resource requirements of the more complex X.500 Directory Access Protocol (DAP). For example, LDAP can be used to locate people, organizations, and other resources in an Internet or intranet directory.

Lightweight Third Party Authentication (LTPA)

1. A protocol that uses cryptography to support security in a distributed environment.
2. An authentication framework that allows single sign-on across a set of web servers that fall within an Internet domain.

link
: A line or arrow that connects activities in a process. A link passes information between activities and determines the order in which they run.

link aggregation
: The grouping of physical network interface cards, such as cables or ports, into a single logical network interface. Link aggregation is used to increase bandwidth and network availability. See also aggregate interface.

link name
: A name defined in the deployment descriptor of the encompassing application.

link pack area (LPA)
: The portion of virtual storage below 16 MB that contains frequently used modules.

listener
: A program that detects incoming requests and starts the associated channel.

listener port
: An object that defines the association between a connection factory, a destination, and a deployed message-driven bean. Listener ports simplify the administration of the associations between these resources.

literal
: A symbol or a quantity in a source program that is itself data, rather than a reference to data.

Literal XML
: An encoding style for serializing data over SOAP protocol. Literal XML is based on an XML schema instance.

little endian
: A format for storage or transmission of binary data in which the least significant value is placed first. See also big endian.

load balancing
: The monitoring of application servers and management of the workload on servers. If one server exceeds its workload, requests are forwarded to another server with more capacity.

loader
: A component that reads data from and writes data to a persistent store.

LOB
: See large object.

local

1. Pertaining to an element that is available only in its own process. See also global.
2. Pertaining to a device, file, or system that is accessed directly from a user system, without the use of a communication line. See also remote.

local authentication (LAU)
: The process of validating a user identity to the system according to the local operating system account to which the user logged in. If the user is authenticated, the user is mapped to a principal.

local database
: A database that is located on the workstation in use. See also remote database.

locale
: A setting that identifies language or geography and determines formatting conventions such as collation, case conversion, character classification, the language of messages, date and time representation, and numeric representation.

local history
: Copies of files that are saved in the workbench in order to compare the current version with previous versions. Subject to configurable preferences, the workbench updates the local history each time an editable file is saved.

local home interface
: In EJB programming, an interface that specifies the methods used by local clients for locating, creating, and removing instances of enterprise bean classes. See also remote home interface.

local queue
: A queue that belongs to the local queue manager. A local queue can contain a list of messages waiting to be processed. See also remote queue.

local queue manager
: The queue manager to which the program is connected and that provides message queuing services to the program. See also remote queue manager.

local server
: A predefined server that designates the current computer to run the Integration Flow Designer.

local transaction
: A recoverable unit of work managed by a resource manager and not coordinated by an external transaction manager.

local transaction containment (LTC)
: A bounded scope that is managed by the container to define the application server behavior in an unspecified transaction context.

location

1. A particular occurrence or example of a location definition. If there is a location definition called USA Call Center, an example of a location would be Toledo Call Center.
2. A physical space that is being monitored. A location can contain many areas. See also area.

location service daemon
: A component of the Remote Method Invocation and Internet Inter-ORB Protocol (RMI/IIOP) communication function that works with workload management to distribute RMI requests among application servers in a cell.

lock
: A means of preventing uncommitted changes made by one application process from being perceived by another application process and for preventing one application process from updating data that is being accessed by another process. A lock ensures the integrity of data by preventing concurrent users from accessing inconsistent data.

logger
: A named and stateful object with which the user code interacts and that logs messages for a specific system or application component.

logging
: The recording of data about specific events on the system, such as errors.

logging level
: A value that controls which events are processed by Java logging.

log handler
: A class that uses loggers, levels, and filters to direct whether events are processed or suppressed.

logical derivation
: A derivation from a physical document that can have additional service description metadata allocated to the derivation. See also logical model.

logical model
: A set of logical derivations. See also logical derivation.

logical terminal (LT)
: In SWIFT, the logical entity through which users send and receive SWIFT messages. A logical terminal is identified by its LT name.

logical terminal table (LTT)
: A MERVA table used to define logical terminals, their synonyms, and other attributes.

logical unit of work (LUW)
: The work that occurs between the start of a transaction and commit or rollback and between subsequent commit and rollback actions. This work defines the set of operations that must be considered part of an integral set.

login binding
: A definition of the implementation to provide login information per authentication methods.

login mapping
: A Java Authentication and Authorization Service (JAAS) login configuration that is used to authenticate a security token in a web service security header.

long name
: The property that specifies the logical name for the server on the z/OS platform.

long-running process
: A process that can come to a complete stop while waiting for input or instructions. The most common form of this interruption is a human interaction or decision.

loop
: A sequence of instructions performed repeatedly.

loop ID
: A unique code that identifies an EDI loop.

loop repeat
: A number indicating the maximum number of times a loop can be used in succession.

loose coupling
: A coupling that supports an extensible software architecture.

LPA
: See link pack area.

LT
: See logical terminal.

LTC
: See local transaction containment.

LT code
: The ninth character of an LT name. For example, the LT code of the LT name XXXXUSNYA is A.

LT name
: A nine-character name of the form BBBBCCLLX, where BBBBCCLL represents the eight-character bank identifier code (BIC8), and X represents the logical terminal (LT) code.

LTPA
: See Lightweight Third Party Authentication.

LTT
: See logical terminal table.

LUW
: See logical unit of work.

M

MAC
: See Media Access Control.

macroflow
: See long-running process.

mail session
: A resource collection of protocol providers that authenticate users and control user access to messaging systems.

maintenance mode
: A state of a node or server that an administrator can use to diagnose, maintain, or tune the node or server without disrupting incoming traffic in a production environment.

manageability
: The ability to manage a resource, or the ability of a resource to be managed. (OASIS)

manageability capability

A capability associated with one or more management domains. (OASIS)

manageability capability interface
: A web service interface representing one manageability capability. (OASIS)

manageability consumer
: A user of manageability capabilities associated with one or more manageable resources. (OASIS)

manageability endpoint
: A web service endpoint associated with and providing access to a manageable resource. (OASIS)

manageability interface
: The composition of one or more manageability capability interfaces. (OASIS)

manageable resource
: A resource capable of supporting one or more standard manageability capabilities. (OASIS)

Managed Bean (MBean)
: In the Java Management Extensions (JMX) specification, the Java objects that implement resources and their instrumentation.

managed deployment environment
: A set of server components that are used to test and deploy applications in a controlled environment.

managed environment
: An environment where services, such as transaction demarcation, security, and connections to Enterprise Information Systems (EISs), are managed on behalf of the running application. Examples of managed environments are the web and Enterprise JavaBeans (EJB) containers.

managed file
: A library item that is created outside of IBM Process Designer and that is part of a process application, such as an image or Cascading Style Sheet (CSS). Creating managed files ensures that all required files are available and installed when a project is ready for testing or production.

managed mode
: An environment in which connections are obtained from connection factories that the Java EE server has set up. Such connections are owned by the Java EE server.

managed node
: A node that is federated to a deployment manager and contains a node agent and can contain managed servers. See also node.

managed resource
: An entity that exists in the runtime environment of an IT system and that can be managed. See also sensor.

managed server
: A server within a managed node, to which SCA modules and applications can be deployed.

management domain
: An area of knowledge relative to providing control over, and information about the behavior, health and life cycle of manageable resources.

Management Information Base (MIB)
: In the Simple Network Management Protocol (SNMP), a database of objects that can be queried or set by a network management system. See also Simple Network Management Protocol.

manifest
: A special file that can contain information about the files packaged in a JAR file. (Sun)

manual emulator
: An emulator that requires users to specify response values for an emulated component or reference at run time. See also emulator, programmatic emulator.

map
: 1. A file that defines the transformation between sources and targets.
  2. A data structure that maps keys to values.
  3. In the EJB development environment, the specification of how the container-managed persistent fields of an enterprise bean correspond to columns in a relational database table or other persistent storage.

map chaining
: The process of producing multiple documents from a single document by executing several maps to translate the single document.

map component
: An Integration Flow Designer object that encapsulates a reference to an executable map, along with its execution settings. There are three types of map components: source, compiled, and pseudo.

map control string
: An object compiled from a map, which contains the instructions used by the translator to translate a document from one format to another.

map object
: An object used in the TX Programming Interface that represents an instance of a map in the program memory.

mapped expression
: Part of an SQL statement that is used to retrieve data from a data connection for a field in a business object.

mapping
: 1. The process of transforming data from one format to another.
  2. The relationship between fields in different abstractions of event and action objects.
  3. The act of developing and maintaining a map.

mapping specialist
: The person responsible for creating data transformation maps, validation maps, and functional acknowledgment maps using the Data Interchange Services client.

map rule
: An expression that evaluates to data and produces the required output. A map rule is entered on an output card in the Map Designer and cannot be longer than 32KB.

marker bar
: The gray border at the left of the editor area of the workbench, where bookmarks and breakpoints are shown.

marshal
: To convert an object into a data stream for transmission over a network.

mashup

A graphical interface that features two or more reusable web applications (widgets) presenting seemingly disparate data in an understandable combination for a specific purpose.

master configuration
: The configuration data held in a set of files that form the master repository for either a deployment manager profile or a stand-alone profile. For a deployment manager profile, the master configuration stores the configuration data for all the nodes in the network deployment cell.

matching rule
: The portion of a policy rule in a processing policy that defines the criteria to determine whether the message is processed by its processing rule.

maximum possible score
: A score that describes the maximum of the maximum scores of all individual scorecards. The maximum possible score is a complex scorecard property, its value should be the same in all the scorecards used in a complex scorecard.

maximum score
: The upper limit in a given interval for an attribute that is used in determining reason code assignment. Typically used for linear and logistic models where variable interaction is controlled.

maximum transmission unit (MTU)
: The largest possible unit of data that can be sent on a given physical medium in a single frame. For example, the maximum transmission unit for Ethernet is 1500 bytes.

maximum use
: A number indicating the maximum number of times a compound or simple element can repeat.

MBean
: See Managed Bean.

MBean provider
: A library containing an implementation of a Java Management Extensions (JMX) MBean and its MBean Extensible Markup Language (XML) descriptor file.

MD5
: A type of message algorithm that converts a message of arbitrary length into a 128-bit message digest. This algorithm is used for digital signature applications where a large message must be compressed in a secure manner.

MDB
: See message-driven bean.

measure
: A metric combined with an aggregation type such as average, count, maximum, minimum, sum, or average. See also aggregate metric.

Media Access Control (MAC)
: In networking, the lower of two sublayers of the Open Systems Interconnection model data link layer. The MAC sublayer handles access to shared media, such as whether token passing or contention will be used.

mediation
: An application of service interaction logic to messages flowing between service requesters and providers.

mediation flow
: A sequence of processing steps, or mediation primitives, that run to produce the mediation when a message is received. See also message flow.

mediation flow component
: A component that contains one or more mediation primitives arranged into request and response flows. Rather than performing business functions, mediation flow components are concerned with the flow of messages.

mediation framework
: A mechanism that supports creation of mediation flows through the composition of mediation primitives.

mediation module
: An SCA module that includes a mediation flow component and primarily enables communication between applications by changing the format, content, or target of service requests.

mediation policy
: A policy that is held in a registry and is applied to a Service Component Architecture (SCA) module. The mediation policy enables mediation flows, which are in the module, to be configured at run time by using dynamic properties.

mediation policy attachment
: An attachment that is a prerequisite for using the mediation policy and gate conditions on the mediation policy.

mediation primitive
: The building blocks of mediation flow components.

mediation service
: A service that intercepts and modifies messages that are passed between client services (requesters) and provider services.

mediation subflow
: A preconfigured set of mediation primitives that are wired together to create a common pattern or use case. Mediation subflows run in the context of a parent flow, and can be reused in mediation flows or in subflows.

meet-in-the-middle mapping
: An approach for mapping enterprise beans to database tables in which enterprise beans and database schema are created simultaneously but independently.

member
: In the Type Designer, a single occurrence of a component in a group in a type tree. If a component has a range, each occurrence of that component might be referred to as a member of a series.

membership
: The state of being a portal user and a place member. Membership in the portal is controlled by the administrator during the installation and set up of portal servers. Membership in places is controlled by a place manager, who determines the level of access for each place member: participant, place designer, or place manager.

membership policy
: A subexpression that is evaluated against the nodes in a cell to determine which nodes host dynamic cluster instances.

memory leak
: The effect of a program that maintains references to objects that are no longer required and therefore need to be reclaimed.

merge
: 1. A point in the process where two or more alternative sequence flow paths are combined into one sequence flow path. No synchronization is required because no parallel activity runs at the join point. BPMN uses multiple incoming sequence flow paths for an activity or an exclusive gateway to perform a merge.
  2. A process element that recombines multiple processing paths, typically after a decision. A merge brings several alternative paths together.

message

1. An object that depicts the contents of a communication between two participants. A message is transmitted through a message flow and has an identity that can be used for alternative branching of a process through the event-based exclusive gateway.
2. A set of data that is passed from one application to another. Messages must have a structure and format that is agreed by the sending and receiving applications. See also category.
3. A communication sent from a person or program to another person or program.

message body
: The part of the message that contains the message payload. See also message header.

message category
: A group of messages that are logically related, such as message that are all used by one application.

message channel
: In distributed message queuing, a mechanism for moving messages from one queue manager to another. A message channel comprises two message channel agents (a sender at one end and a receiver at the other end) and a communication link. See also channel.

message definition
: Information that describes the structure of the messages of a particular type, the elements that each message of that type can or must contain, how a message of that type is represented in various network formats, and the validation rules that apply to a message of that type.

message digest
: A hash value or a string of bits resulting from the conversion of processing data to a number.

message domain
: A group of all the message definitions that are required to satisfy a particular business need (for example, transferring SWIFTNet FIN messages, transferring SWIFTNet Funds messages, or transferring SWIFTNet system messages).

message-driven bean (MDB)
: An enterprise bean that provides asynchronous message support and clearly separates message and business processing.

message-driven rule bean
: An enterprise bean that allows Java EE applications to process messages asynchronously. The bean invokes the execution unit (XU) when a JMS message arrives and posts the results of the rule engine processing to a JMS destination.

message end event
: An end event that also sends a message. See also end event.

message event
: An event that arrives from a participant and triggers another event. If the message event is attached to the boundary of the activity, it changes the normal flow into an exception flow upon being triggered.

message file
: A file containing messages sent in bulk through a message bulking service.

message flow

1. A sequence of processing steps that execute in the broker when an input message is received. Message flows are defined in the workbench by including a number of message flow nodes, each of which represents a set of actions that define a processing step. The connections in the flow determine which processing steps are carried out, in which order, and under which conditions. See also mediation flow, subflow.
2. A connecting object that shows the flow of messages between two collaborating participants. A message flow is represented by a dashed line.

Message Format Service (MFS)
: An IMS editing facility that allows application programs to deal with simple logical messages instead of device-dependent data, thus simplifying the application development process.

Message Format Service control block (MFS control block)
: In MFS, the representation of a message or format that is stored in the IMS.FORMAT library and called into the MFS buffer pool as needed for online execution.

message header
: The part of a message that contains control information such as a unique message ID, the sender and receiver of the message, the message priority, and the type of message. See also message body.

message input descriptor (MID)
: The Message Format Service (MFS) control block that describes the format of the data presented to the application program. See also message output descriptor.

message intermediate event
: An intermediate event that can be used to either receive or send a message. See also intermediate event.

message log
: A file in which an application logs messages about errors that occur or metadata about the message.

message output descriptor (MOD)
: The Message Format Service (MFS) control block that describes the format of the output data produced by the application program. See also message input descriptor.

message processing node
: A node in a message flow that represents a processing step. A message processing node can be either a primitive or a subflow node.

message processing unit (MPU)
: A message processing unit is used to correlate information within a message, for example reason or completion information, and a message text.

message queue
: A named destination to which messages can be sent until they are retrieved by programs that service the queue.

message reception registry (MRR)
: The registry where SWIFT stores the central routing rules. Each receiver defines its own rules and submits them to SWIFT. SWIFT uses these rules to determine the destination of message traffic, that is, to which store and forward queue or to which SWIFTNet Link it is to route each message.

message reference number (MRN)
: A unique 16-digit number assigned to each message for identification purposes. The message reference number consists of an 8-digit domain identifier that is followed by an 8-digit sequence number.

message sequence number (MSN)
: A sequence number for messages.

message standard

A standard that describes a family of message definitions.

message start event
: A start event that is triggered when a specific message is received. See also start event.

message type
: The logical structure of the data within a message. For example, the number and location of character strings.

message warehouse table
: A database table in which the message warehouse service stores index and status information about each message processed by services.

messaging API
: A programming interface that enables an application to send and receive messages and attached files over a messaging system.

messaging engine
: The messaging and connection point to which applications connect to the bus.

messaging middleware
: Software that provides an interface between applications, allowing them to send data back and forth to each other asynchronously. Data sent by one program can be stored and then forwarded to the receiving program when it becomes available to process it.

messaging system
: Software used to deliver electronic messages.

metadata
: Data that describes the characteristics of data; descriptive data. See also application-specific information.

metadata tree
: A list in a tree structure, which is prepared and displayed by the external service wizard, that presents all of the objects discovered from the enterprise information system (EIS).

method

1. A way to implement a function on a class.
2. In object-oriented programming, an operation that an object can perform. An object can have many methods. See also operation.

method extension
: An IBM extension to the standard deployment descriptors for enterprise beans that define transaction isolation methods and control the delegation of credentials.

method permission
: A mapping between one or more security roles and one or more methods that a member of a role can call.

metric
: A holder for information, typically a business performance measurement, in a monitoring context. See also aggregate metric, instance metric.

MFS
: See Message Format Service.

MFS control block
: See Message Format Service control block.

MIB
: See Management Information Base.

microflow
: A short-running process that runs in one transaction. A microflow, which is an IBM extension to the BPEL programming language, runs automatically from start to finish and cannot be interrupted.

micropattern
: A pattern that creates a reusable subprocess from a main process. See also pattern.

MID
: See message input descriptor.

middleware agent
: An agent that enables the administrative domain to manage servers that run middleware software.

middleware descriptor
: An XML file that contains information about different middleware platform types, including discovery sensor intervals and installation information.

middleware node
: A node that is federated to the deployment manager. These nodes must include nodes that run the node agent or middleware agent.

MIME
: See Multipurpose Internet Mail Extensions.

MOD
: See message output descriptor.

model
: A representation of a process, system, or subject area, typically developed for understanding, analyzing, improving, and replacing the item being represented. A model can include a representation of information, activities, relationships, and constraints.

modeled fault
: A fault message that is returned from a service that has been modeled on the Web Services Description Language (WSDL) port type.

model element
: An element that is an abstraction drawn from the system being modeled. In the MOF specification, model elements are considered to be meta-objects.

model view controller (MVC)
: A software architecture that separates the components of the application: the model represents the business logic or data; the view represents the user interface; and the controller manages user input or, in some cases, the application flow. See also view.

module

1. In Java EE programming, a software unit that consists of one or more components of the same container type and one deployment descriptor of that type. Examples include EJB, web, and application client modules. (Sun) See also project.
2. A software artifact that is used for developing, managing versions, organizing resources, and deploying to the runtime environment.
3. A program unit that is discrete and identifiable with respect to compiling, combining with other units, and loading.

monitor

1. In performance profiling, to collect data about an application from the running agents that are associated with that application.

2. A facility of the integration test client that listens for requests and responses that flow over the component wires or exports in the modules of a test configuration.

monitor configuration server
: The application server installation that owns the overall application server configuration for a cell.

monitor details model
: A container for monitoring contexts and their associated metrics, keys, counters, stopwatches, triggers, and inbound and outbound events. The monitor details model holds most of the monitor model information.

monitored directory
: The directory where the rapid deployment tools detect added or changed parts and produce an application that can run on the application server. See also automatic application installation project, free-form project.

monitoring context
: A definition that corresponds to an object to be monitored, such as a process execution, an ATM, a purchase order, or the stock level in a warehouse. At run time, monitoring contexts process the events for a particular object.

monitor model
: A model that describes the business performance management aspects of a business model, including events, business metrics, and key performance indicators (KPIs) that are required for real-time business monitoring.

monitor model CEI configuration owner
: The server installation that owns the overall server configuration that contains the monitor model Common Event Infrastructure (CEI) server target.

mount point
: A logical drive through which volumes are accessed in a sequential access device class. For removable media device types, such as cartridges, a mount point is a logical drive associated with a physical drive. For the file device type, a mount point is a logical drive associated with an I/O stream.

MPMT
: See multiprocess multithread.

MPU
: See message processing unit.

MRN
: See message reference number.

MRR
: See message reception registry.

MSN
: See message sequence number.

MTU
: See maximum transmission unit.

multidimensional analysis
: The process of assessing and evaluating an enterprise on more than one level.

multiple configuration instances
: More than one instance of a product running in the same machine at the same time.

multiple-occurrence mapping
: A form of mapping in which all occurrences of a repeating compound or simple element are mapped to the same repeating compound or simple element in another document.

multiprocess multithread (MPMT)
: A process architecture of the IBM HTTP Server that supports multiple processes as well as multiple threads per process.

Multipurpose Internet Mail Extensions (MIME)
: An Internet standard that allows different forms of data, including video, audio, or binary data, to be attached to email without requiring translation into ASCII text.

MVC
: See model view controller.

N

named constant
: A descriptive name that is given to a value and can be used in a filter in place of a value.

namespace
: A logical container in which all the names are unique. The unique identifier for an artifact is composed of the namespace and the local name of the artifact.

namespace object
: A Data Interchange Services object that contains information about an XML namespace and assists the translator in being namespace aware when translating a source document to an XML document.

naming
: An operation that is used to obtain references to objects that are related to applications.

naming context
: A logical namespace containing name and object bindings.

naming federation
: The process of binding naming systems so that the aggregate system can process composite names that span the naming systems.

naming service
: An implementation of the Java Naming and Directory Interface (JNDI) standard.

NAS
: See network access server.

NAT
: See network address translation.

native
: Pertaining to the relationship between a transport user and a transport provider that are both based on the same transport protocol.

navigation bar
: A set of links to other web pages in a website. For example, a navigation bar is typically located across the top or down the side of a page and contains direct links to the major sections within the website.

navigation phrase
: In the vocabulary, a phrase that associates two business elements. A navigation phrase corresponds to a method that has a return value or an attribute in the business object model (BOM).

net change
: The cumulative effect of multiple changes to an object. For example, an add action followed by a remove action cancels out other changes, consequently yielding no net change.

netmask
: See network mask.

net present value (NPV)
: The estimated monetary value of an investment based on expected returns and expected costs, where these expected returns and expenses are discounted by a rate that reflects inflation and opportunity costs.

network
: A system of resources, such as appliances, computers, and storage devices, that are connected virtually or physically.

network access server (NAS)
: A device that functions as an access control point for users in remote locations who connect to an internal network or to an ISP. A NAS might include its own authentication services or rely on a separate authentication server. A NAS can be a dedicated server or a software service within a regular server.

network acknowledgment
: A response from the network indicating the status of an interchange envelope, such as sent or received.

network address translation (NAT)
: The conversion of a network address that is assigned to a logical unit in one network into an address in an adjacent network.

network delivery notification
: A delivery notification that conforms to the network protocol. See also application delivery notification.

network deployment cell
: A logical group of servers, on one or more machines, managed by a single deployment manager.

Network File System (NFS)
: A protocol, developed by Sun Microsystems, Incorporated, that allows a computer to access files over a network as if they were on its local disks.

network identifier
: A single character that is placed before a message type to indicate which network is to be used to send the message; for example, S for SWIFT.

Network Installation Management (NIM)
: An environment that provides installation and configuration of software within a network interface.

network mask (netmask)
: A number that is the same as an Internet Protocol (IP) address. A network mask identifies which part of an address is to be used for an operation, such as making a TCP/IP connection.

network protocol stack
: A set of network protocol layers and software that work together to process the protocols.

Network Time Protocol (NTP)
: A protocol that synchronizes the clocks of computers in a network.

NFS
: See Network File System.

NIM
: See Network Installation Management.

node
: 1. An endpoint or junction used in a message flow.
  2. Any element in a tree.
  3. In XML, the smallest unit of a valid, complete structure in a document.
  4. A logical group of managed servers. See also managed node.

node agent
: An administrative agent that manages all application servers on a node and represents the node in the management cell.

node federation
: The process of combining the managed resources of one node into a distributed network such that the central manager application can access and administer the resources on the node.

node group
: A collection of application server nodes that defines a boundary for server cluster formation.

node name
: The machine name or host name that must be unique.

nonce
: A unique cryptographic number that is embedded in a message to help detect a replay attack.

none start event
: A start event that does not have a defined trigger. A none start event can be used in a descriptive process that does not require technical information or in a subprocess where the control of the process flow is passed from its parent process. See also start event.

nonrepudiation
: In business-to-business communication the ability of the recipient to prove who sent a message based on the contents of the message. This can derive from the use of a digital signature on the message, which links the sender to the message.

nonrepudiation data repository
: The repository in which copies of documents (and authentication information for signed documents) are stored in case disputes arise regarding the authenticity of document exchanges.

normal flow
: All sequence flow paths in a process except those paths that originate from an intermediate event that is attached to the boundary of an activity. See also exception flow.

notation
: An XML construct that contains a note, a comment or an explanation about information in an XML file. A notation can be used to associate a binary description with an entity or attribute.

notification

1. A message that contains the event descriptions that are sent to managed resources, web services and other resources.
2. An occurrence within a process that can trigger an action. Notifications can be used to model conditions of interest to be transmitted from a sender to a (typically unknown) set of interested parties (the receivers).

notification broadcaster
: An element that is responsible for publishing notifications. Notification receivers listen for these notifications.

notification channel
: A mode by which a subscriber uses a business service.

notification program
: A program or web service that can be triggered when an event occurs.

notification receiver
: An element that listens for and receives notifications. By default, this element starts listening when its owning process starts.

NPV
: See net present value.

NTP
: See Network Time Protocol.

numeric constant
: The actual numeric value to be used in processing, instead of the name of a field containing the data. A numeric constant can contain any of the numeric digits 0 through 9, a sign (plus or minus), and a decimal point.

O

OAEP
: See optimal asymmetric encryption padding.

OAMS
: See outbound application message store.

object
: In object-oriented design or programming, a concrete realization (instance) of a class that consists of data and the operations associated with that data. An object contains the instance data that is defined by the class, but the class owns the operations that are associated with the data.

object adapter
: In Common Object Request Broker Architecture (CORBA), the primary interface that a server implementation uses to access Object Request Broker (ORB) functions.

object-oriented programming
: A programming approach based on the concepts of data abstraction and inheritance. Unlike procedural programming techniques, object-oriented programming concentrates not on how something is accomplished but instead on what data objects comprise the problem and how they are manipulated.

object reference
: In Common Object Request Broker Architecture (CORBA), the information needed to reliably identify a particular object.

Object Request Broker (ORB)
: In object-oriented programming, software that serves as an intermediary by transparently enabling objects to exchange requests and responses.

observer
: A task that watches a process and its associated repositories, and produces output when a certain condition becomes true (for example, a threshold value has been reached).

ODBC
: See Open Database Connectivity.

OLAP
: See online analytical processing.

on-demand configuration
: A component that detects and dynamically configures routing rules, which tell the on demand router (ODR) how to route requests.

on demand router
: A proxy server that is the point of entry into the product environment and is a gateway through which prioritized HTTP requests and Session Initiation Protocol (SIP) messages flow to the middleware servers in the environment.

one-way hash
: An algorithm that converts processing data into a string of bits; known as a hash value or a message digest.

one-way interaction
: A type of messaging interaction in which a request message is used to request function without a reply.

online analytical processing (OLAP)
: The process of collecting data from one or many sources; transforming and analyzing the consolidated data quickly and interactively; and examining the results across different dimensions of the data by looking for patterns, trends, and exceptions within complex relationships of that data.

ontology
: An explicit formal specification of the representation of the objects, concepts, and other entities that can exist in some area of interest and the relationships among them. See also Web Ontology Language.

Open Database Connectivity (ODBC)
: A standard application programming interface (API) for accessing data in both relational and nonrelational database management systems. Using this API, database applications can access data stored in database management systems on a variety of computers even if each database management system uses a different data storage format and programming interface. See also Java Database Connectivity.

Open Mobile Alliance
: An industry forum for developing interoperable mobile service enablers.

open relationship
: A relationship on an object that no longer points to a second object because the second object has been deleted.

Open Servlet Engine (OSE)
: A lightweight communications protocol developed by IBM for interprocess communication.

open source
: Pertaining to software whose source code is publicly available for use or modification. Open source software is typically developed as a public collaboration and made freely available, although its use and redistribution might be subject to licensing restrictions. Linux is a well known example of open source software.

Open Transaction Manager Access (OTMA)
: A component of IMS that implements a transaction-based, connectionless client/server protocol in an MVS sysplex environment. The domain of the protocol is restricted to the domain of the z/OS Cross-System Coupling Facility (XCF). OTMA connects clients to servers so that the client can support a large network (or a large number of sessions) while maintaining high performance. See also IMS Connect.

operation
: An implementation of functions or queries that an object might be called to perform. See also method.

operation mapping
: An interface map in which operations of the source interface are mapped to operations of the target interface.

operator
: A building block that lets the user compare or establish relationships between the different parts of business rule statements.

optimal asymmetric encryption padding (OAEP)
: In cryptography, a padding scheme that is often used with RSA encryption.

optimistic locking
: A locking strategy whereby no lock is held between the time that a row is selected and the time that an update or a delete operation is attempted on that row. See also pessimistic locking.

option
: A parameter that determines how a message is to be processed.

optional component
: Within a group type, a component that can be defined to represent a data object that is not required to be present in the data. The component range maximum specifies how many occurrences of the data object might optionally exist.

option set
: A named group of options and their settings that can be specified in a request or in another option set, thereby eliminating the need to specify each option individually.

ORB
: See Object Request Broker.

organization
: An entity where people cooperate to accomplish specified objectives, such as an enterprise, a company, or a factory.

organizational unit (OU)
: A body whose data is to be kept separate from that of other, similar bodies. WebSphere BI for FN uses OUs to control access to resources, and to ensure data segregation. Typically, OUs are used to represent different financial institutions, or different departments within a financial institution.

organization unit
: A particular occurrence or example of an organization definition. For an organization definition called Department, an example of an organization unit would be Sales and Marketing.

orphaned token
: A token that is associated with an activity that was removed from a business process definition (BPD).

OSE
: See Open Servlet Engine.

OSGi framework
: A general-purpose, secure, and managed Java framework that supports the deployment of extensible and downloadable applications known as bundles.

OSGi service
: An interface registered in the OSGi Service Platform and made available for receiving remote or local invocations.

OTMA
: See Open Transaction Manager Access.

OU
: See organizational unit.

outbound
: In communication, pertaining to data that is sent to the network. See also inbound.

outbound application message store (OAMS)
: A message store in which messages sent by local applications (ISN messages) and their acknowledgement messages (ISN ACKs) are stored.

outbound authentication
: The configuration that determines the type of accepted authentication for outbound requests.

outbound document
: See target document.

outbound event
: An event emitted from a monitoring context or from a KPI context.

outbound port
: The mechanism through which an outbound service communicates with the externally hosted web service. Messages pass between the outbound service and the external service through the appropriate port.

outbound processing
: The process by which a calling client application uses the adapter to update or retrieve data in an enterprise information system (EIS). The adapter uses operations such as create, update, delete, and retrieve to process the request.

outbound service
: The service that provides access through one or more outbound ports to a web service that is hosted externally.

out parameter
: A parameter value that is set by the execution process and provided as output from the rule set after the execution is completed.

output
: An exit point through which an element can notify downstream elements that they can now start.

output activity
: The end point of the business process.

output branch
: The area of a decision, fork, join, or merge that contains the outputs.

output card
: In the Map Designer, a card that contains the complete definition of an output for the map including information such as target identification, destination specifics and the behavior that should occur during processing.

output criteria

Number and types of outputs required to be produced by a task or process.

output screen
: A screen that a user navigates to based on data entry and keystrokes in a 3270 application. In the 3270 terminal service recorder, the access route from one screen to another can be recorded and saved in a dialog file.

output terminal node
: A primitive through which a message is propagated by a subflow. Each output terminal node is represented as an output terminal of the corresponding subflow node.

override
: An execution setting that overrides default source and target settings of a map.

OWL
: See Web Ontology Language.

P

package

1. To assemble components into modules and modules into enterprise applications.
2. In Java programming, a group of types. Packages are declared with the package keyword. (Sun)
3. The wrapper around the document content that defines the format used to transmit a document over the Internet, for example, RNIF, AS1, and AS2.

pack type
: A container, such as a case or pallet. Each pack type is associated with various pieces of information that are required for converting customer-specific product codes to EPC format.

pad character
: A character used to fill empty space. For example, in a database application, a field that is ten characters in length that has the word "file" in it contains four text characters and six pad characters

page
: A node in a portal that can contain content in addition to labels and other pages. Pages can contain child nodes, column containers, row containers, and portlets.

page list
: An assembly property that specifies the location to forward a request, but automatically tailors that location, depending on the Multipurpose Internet Mail Extensions(MIME) type of the servlet.

page template
: In Page Designer, a page that is used as a starting point to define consistent styles and layout for any new HTML or JavaServer Pages (JSP) page within a website.

palette

1. The location for building customized components and parenting them with other components. Subsequently, components can be reused by copying and moving them to other interfaces.
2. A range of graphically displayed choices, such as colors or collections of tools, that can be selected in an application.

pallet
: An industry standard sized wooden, plastic, or metal platform to facilitate the movement of materials. Cartons are stacked on the pallet allowing movement via pallet jacks or forklift trucks.

PAP
: See policy administration point.

parallel garbage collection
: A type of garbage collection that uses several threads simultaneously.

parallel gateway
: A gateway that creates parallel paths without checking conditions.

parallel job
: A job that is run as multiple concurrent steps. A top-level job is submitted to the job scheduler and after submission is divided into subordinate jobs that run at the same time.

parameter (parm)
: A value or reference passed to a function, command, or program that serves as input or controls actions. The value is supplied by a user or by another program or process. See also keyword.

parameter mapping
: An interface map that is one level deeper than operation mappings because it maps the parameters in the source operation to the parameters in the target operation. There are five types of parameter mappings: move, map, extract, Java, and assign.

parent document
: A document whose values are inherited by another document (the child document).

parent process
: A process that contains a subprocess.

parm
: See parameter.

parse
: To break down a string of information, such as a command or file, into its constituent parts.

parser
: A module used to break down a document into its component parts and to construct a document from its component parts.

participant
: A business entity (such as a company, company division, or a customer) or a business role (such as a buyer or a seller) that controls or is responsible for a business process.

partition

1. A group of cells in a decision table that are in the same condition column and have a common cell immediately to the left.

2. To divide a type into subtypes that are mutually exclusive.

partitioned data set (PDS)
: A data set on direct access storage that is divided into partitions, called members, each of which can contain a program, part of a program, or data.

partitioned type
: A type whose subtypes are distinguishable or mutually exclusive.

partner connection
: An interaction that has been associated with specific sending and receiving partners, and also specifies the destinations and other routing information necessary for an exchange.

partner profile
: A profile that includes information about the partner such as its name, its business identifier, such as a DUNS number, and a list of user IDs authorized to access the Community Console. See also Data Universal Numbering System.

part reference
: An object that is used by a configuration to reference other related configuration objects.

passivation
: In enterprise beans, the process of transferring an enterprise bean from memory to auxiliary storage. (Sun) See also activation.

PassTicket
: In RACF secured sign-on, a dynamically generated, random, one-time-use, password substitute that a workstation or other client can use to sign on to the host rather than sending a RACF password across the network.

password
: In computer and network security, a specific string of characters used by a program, computer operator, or user to access the system and the information stored within it. See also authentication.

password stashing
: Saving a password that is encrypted in a file or on a hard disk drive. The keydb password must reside in a file to use secure sockets layer (SSL).

path

1. A route that the flow can take through the activities in a process. There may be several alternative paths.
2. The route through a file system to a specific file.

path qualified mapping
: A form of mapping in which all occurrences of a repeating compound or simple element are mapped to the same repeating compound or simple element in another document.

pattern
: A reusable solution that encapsulates a tested approach to solving a common architecture, design, or deployment task in a particular context. See also micropattern.

payload
: The body of a message that holds content.

PCRE
: See Perl-compatible regular expression.

PDP
: See policy decision point.

PDS
: See partitioned data set.

peer access point
: A means by which core groups can communicate with other cells.

PEM
: See privacy enhanced mail.

people assignment criterion
: A property that defines the members of each of the role groups.

people awareness
: The collaboration feature that provides access to people from various contexts. People awareness lets you see references to people and contact people by name through the Sametime online status indicator. Throughout the portal, wherever you see the name of a person, you can view the online status of the person, send email, initiate a chat, or share an application via an electronic meeting. See also person link.

PEP
: See policy enforcement point.

Performance Monitoring Infrastructure (PMI)
: A set of packages and libraries assigned to gather, deliver, process, and display performance data.

Perl-compatible regular expression (PCRE)
: A regular expression C library that is much richer than classic regular expression libraries. See also regular expression.

permission
: Authorization to perform activities, such as reading and writing local files, creating network connections, and loading native code.

persist
: To be maintained across session boundaries, typically in nonvolatile storage such as a database system or a directory.

persistence

1. In Java EE, the protocol for transferring the state of an entity bean between its instance variables and an underlying database. (Sun)
2. A characteristic of data that is maintained across session boundaries, or of an object that continues to exist after the execution of the program or process that created it, typically in nonvolatile storage such as a database system.

persistence level
: A level that determines the degree of detail written to the database as a business process runs. Decreasing the persistence level increases the business process performance at the cost of full tracking for each step of the business process.

persistence service
: A service that provides private application programming interface (API) support to store and accesses executable resources.

persistent data store
: A nonvolatile storage for event data, such as a database system, that is maintained across session boundaries and that continues to exist after the execution of the program or process that created it.

person
: An individual authenticated by the portal and having a person record in one or more corporate directories. Persons can be members of places, public groups within the organization corporate directory, or personal groups that a user defines.

personal group
: In Sametime Connect, a group of people designated by the user as a group. A user can choose individuals from the public Directory (public group) and create personal groups, which are then stored locally. Users can add and remove people from a personal group, whereas the membership of the public group is defined by the owner of the public Directory.

personalization
: The process of enabling information to be targeted to specific users based on business rules and user profile information.

person link
: A reference to a person name or a group name that appears with the Sametime online status indicator. The reference lets you view the online status the person, send an email, start a chat, or share an application using an electronic meeting, among other actions shown on the person link menu. See also people awareness.

perspective
: A group of views that show various aspects of the resources in the workbench.

pessimistic locking
: A locking strategy whereby a lock is held between the time that a row is selected and the time that a searched update or delete operation is attempted on that row. See also optimistic locking.

phantom read
: A read request in which two identical queries run, and the collection of rows returned by the second query is different from the first query.

PHP Hypertext Preprocessor
: A widely-used general-purpose scripting language that is especially suited for web development and can be embedded into HTML.

phrase template
: A pattern for the verbalization of a business element.

PIP
: See RosettaNet Partner Interface Process.

pivot table
: A table characterized by having one metric as a column dimension and all the rest of the metrics represented as row dimensions.

PKA
: See public key algorithm.

PKCS
: See Public Key Cryptography Standards.

PKI
: See public key infrastructure.

place
: A virtual location that is visible in the portal where individuals and groups meet to collaborate. In a portal, each user has a personal place for private work, and individuals and groups have access to a variety of shared places, which can be either public places or restricted places.

place designer
: A member of a place who can edit place layout and bookmarks. See also participant, place manager.

placeholder
: A variable that is replaced with a value.

place manager
: A member of a place who can edit place membership, layout, and bookmarks. See also participant, place designer.

place member
: A individual or group who has joined or been granted access to a place. Place members have three levels of access to a place: manager, designer, and participant.

plug-in
: A separately installable software module that adds function to an existing program, application, or interface.

PMI
: See Performance Monitoring Infrastructure.

point
: The numeric value that is assigned to an attribute based on the value of the attribute and the interval in which the value is included.

point difference
: The upper limit in a given interval for an attribute that is ultimately used to determine reason code assignment. Typically, this limit is used for linear and logistic models where the interaction of variables is controlled.

point-to-point
: Pertaining to a style of messaging application in which the sending application knows the destination of the message.

poison message
: In a queue, an incorrectly formatted message that the receiving application cannot process. The message can be repeatedly delivered to the input queue and repeatedly backed out by the application.

policy
: A set of considerations that influence the behavior of a managed resource or a user. See also policy expression.

policy administration point (PAP)
: A capability that provides enterprise service-oriented architecture (SOA) policy administration capabilities, such as policy creation, modification, storage, and distribution.

policy-controlled mediation
: A mediation that has dynamic properties that are controlled by mediation policies.

policy decision point (PDP)
: A capability that decides, based on environmental conditions, which predefined policies in the environment should be enforced. For example, a policy decision point might use a requester identity to determine whether to limit access to a resource.

policy enforcement point (PEP)
: A capability that enforces policy decisions maybe by a policy decision point. For example, a policy enforcement point would permit or deny a requester access to a resource depending on what the policy decision point determined is the correct action.

policy expression
: A representation of a policy. See also policy.

policy manager
: A decision management user role who is responsible for enforcing decisions through the creation and maintenance of rules.

policy rule
: A rule in a processing policy that consists of a matching rule and a processing rule.

policy set
: A collection of assertions about how services are defined, which can be used to simplify security configurations.

pool
: The graphical representation of a participant in a collaboration.

port
: 1. In the Internet suite of protocols, a specific logical connector between the Transmission Control Protocol (TCP) or the User Datagram Protocol (UDP) and a higher level protocol or application.
  2. As defined in a Web Services Description Language (WSDL) document, a single endpoint that is defined as a combination of a binding and a network address.

portal
: A single, secure point of access to diverse information, applications, and people that can be customized and personalized.

Portal Administration
: The place where portal administrators set and maintain basic collaboration permissions, place records, place membership records, and server settings for companion products for advanced collaboration.

portal farm
: A series of identically configured, stand-alone portal server instances that offer a way to maintain a highly scalable and highly available server environment.

port destination
: The specialization of a service integration bus destination. Each port destination represents a particular message format and transport protocol that you can use to pass messages to an externally-hosted service.

portlet
: A reusable component that is part of a web application that provides specific information or services to be presented in the context of a portal.

portlet API
: The set of interfaces and methods that are used by Java programs running within the portal server environment to obtain services.

portlet application
: A collection of related portlets that can share resources with one another.

portlet container
: A column or row that is used to arrange the layout of a portlet or other container on a page.

portlet framework
: The set of classes and interfaces that support Java programs running within the portal server environment.

portlet mode
: A form assumed by a portlet to provide a distinctive interface for users to perform different tasks. Portlet modes can include view, edit, and help.

port number
: In Internet communications, the identifier for a logical connector between an application entity and the transport service.

port type
: An element in a Web Services Description Language (WSDL) document that comprises a set of abstract operations, each of which refers to input and output messages that are supported by the web service. See also interface.

POST
: In HTTP, a parameter on the METHOD attribute of the FORM tag that specifies that a browser will send form data to a server in an HTTP transaction separate from that of the associated URL.

postcondition
: A constraint that must be true at the completion of an operation.

precondition
: 1. A group of rule statements in which the user defines global variables for a decision table or decision tree and conditions that must be met before any rows or branches in the decision table or tree can be executed.
  2. A definition of what must be true when a task or process starts.

predefined business process
: A business process that is ready to use upon installation of Sterling B2B Integrator.

predicate
: A Boolean logic term denoting a logical expression that determines the state of a variable.

presumed trust
: A type of identity assertion where trust is presumed and additional trust validation is not performed. Use this mode only in an environment where trust is established with some other mechanism.

primary document
: A document that the services in a business process act on or in relation to. A primary document is usually the document passed to a business process by the initiating adapter.

primary key
: 1. An object that uniquely identifies an entity bean of a particular type.
  2. In a relational database, a key that uniquely identifies one row of a database table. See also constraint, foreign key.

primary server
: The server on which all resources that are to be deployed exactly once per instance or once per organization unit (OU) are deployed.

primitive
: A message processing node that cannot be further subdivided. See also subflow node.

primitive type
: In Java, a category of data type that describes a variable that contains a single value of the appropriate size and format for its type: a number, a character, or a Boolean value. Examples of primitive types include byte, short, int, long, float, double, char, boolean.

principal

An entity that can communicate securely with another entity. A principal is identified by its associated security context, which defines its access rights.

priority
A property that determines the order in which business rules are executed in an application.

privacy enhanced mail (PEM)
A standard for secure email on the Internet.

private business object
1. In XSD, a business object attribute that defines an anonymous complex type instead of referencing a named complex type.
2. A business object that is contained within other business objects. Private business objects are visible only to the containing business object, thereby making them private. See also business object.

private key
In secure communication, an algorithmic pattern used to encrypt messages that only the corresponding public key can decrypt. The private key is also used to decrypt messages that were encrypted by the corresponding public key. The private key is kept on the user system and is protected by a password. See also key, public key.

private process
A process that is strictly internal to a specific organization.

private service bundle
A service bundle that is not explicitly mentioned in the customization definition document (CDD), but that is included in a service bundle set and provides resources required by another service bundle. In a customization definition report, private service bundles are listed, and their names are followed by the string [private].

probe
A reusable set of Java code fragments and supporting attributes for collecting detailed runtime information about objects, arguments, and exceptions. See also Probekit.

Probekit
A scriptable framework for doing byte-code insertion to probe the workings of a target program. See also probe.

process
1. A progressively continuing procedure consisting of a series of controlled activities that are systematically directed toward a particular result or end.
2. The sequence of documents or messages to be exchanged between the Community Managers and participants to run a business transaction.
3. A sequence or flow of activities in an organization with the objective of carrying out work. In BPMN, a process is depicted as a graph of flow elements, which are a set of activities, events, gateways, and sequence flow paths that adhere to BPMN execution semantics.

process application
A container in the Process Center repository for process models and supporting implementations. A process application typically includes business process definitions (BPDs), the services to handle implementation of activities and integration with other systems, and any other items that are required to run the processes. Each process application can include one or more tracks.

process case
A possible path through a process, identified by a unique set of process decision outcomes and possibly determined by attributes and values of incoming data.

Process Center Console
An interface to the Process Center repository where administrators can create and manage process applications, manage user access to library items, install snapshots on test or production servers, and perform other tasks.

process control information
Map component settings that can be changed at run time by specifying overrides at the command line, in a command file, or by configuring the Launcher.

process data
Data that is accumulated in an XML document about a business process during the life of the process. Activities in the process add elements to the process data and use components of the process data to complete configured processing tasks.

process definition
A specification of the runtime characteristics of an application server process.

process diagram
A diagram that represents the flow of work for a process. The objects within a process diagram include tasks, processes, connections, business items, resources, and decisions.

process flow
The representation of interdependencies between activities in a structured format.

processing action
A defined activity in a processing rule that is performed against messages. See also action.

processing policy
A collection of policy rules that define message processing through a service.

processing rule
The portion of a policy rule in a processing policy that identifies the processing actions to perform against messages.

process instance
A manifestation of a modeled process that is created in a simulated or real environment.

process model
A representation of a real-time business process. A business process model is composed of the individual steps or activities that make up the process, contains the conditions that dictate when the steps or activities occur, and identifies the resources that are required to run the business process.

process module
A program unit that contains a set of process templates that support administrative tasks.

producer definition
A set of interfaces that are defined for the producer portal. The producer definition can include the producer service description, the producer portal URL, and the security setup. See also consumer portal, producer portal.

producer portal
A portal that provides portlets as a service so that other portals, called consumer portals, can use the portlets and make the portlets available to their users. See also consumer portal, producer definition.

profile
Data that describes the characteristics of a user, group, resource, program, device, or remote location.

programmatic emulator
: An emulator that uses a Java or visual snippet to automatically specify response values for an emulated component or reference at run time. See also emulator, manual emulator.

programmatic login
: A type of form login that supports application presentation site-specific login forms for the purpose of authentication.

programmatic security
: A collection of methods used by applications when declarative security is not sufficient to express the security model of the application.

program temporary fix (PTF)
: For System i, System p, and System z products, a package containing individual or multiple fixes that is made available to all licensed customers. A PTF resolves defects and might provide enhancements. See also fix pack.

project
: 1. A specific organization of rules and other elements that facilitates the authoring and management of a logical grouping of rules.
  2. An organized collection used to group folders or packages. Projects are used for building, version management, sharing, and organizing resources related to a single work effort. See also library, module.

project versioning
: The component that interacts with a CVS or Rational ClearCase server to share and create version projects and project data.

promoted property
: A property of a mediation module made visible by the solution integrator to the runtime administrator, so that its value can be changed at run time.

prompt
: A component of an action that indicates that user input is required for a field before making a transition to an output screen.

propagation
: The point at which the properties of a type are inherited by its subtypes.

property
: A characteristic of an object that describes the object. A property can be changed or modified. Properties can describe an object name, type, value, or behavior, among other things.

protocol binding
: A binding that enables the enterprise service bus to process messages independently of the communication protocol.

protocol-level RAS granularity
: The level of RAS granularity at which RAS attribute values are assigned on a protocol-wide basis. RAS attribute values defined at the protocol-level are assigned to all requests for a particular protocol, such as the HTTP protocol or IIOP protocol. See also RAS granularity.

proxy
: An application gateway from one network to another for a specific network application such as Telnet or FTP, for example, where a firewall proxy Telnet server performs authentication of the user and then lets the traffic flow through the proxy as if it were not there. Function is performed in the firewall and not in the client workstation, causing more load in the firewall.

proxy cluster
: A group of proxy servers that distributes HTTP requests across the cluster.

proxy peer access point
: A means of identifying the communication settings for a peer access point that cannot be accessed directly.

proxy server
: 1. A server that receives requests intended for another server and that acts on behalf of the client (as the client's proxy) to obtain the requested service. A proxy server is often used when the client and the server are incompatible for direct connection. For example, the client is unable to meet the security authentication requirements of the server but should be permitted some services.
  2. A server that acts as an intermediary for HTTP Web requests that are hosted by an application or a web server. A proxy server acts as a surrogate for the content servers in the enterprise.

pseudoattribute
: An attribute that cannot have a value, and is used to indicate a binary state, such as yes/no or on/off. For example, the attribute local might be present for some resources and absent for others, indicating whether the resource is local. Pseudo attributes are especially useful for implementing access rights, such as read, update, or delete. See also real attribute.

pseudolink
: In the Integration Flow Designer, dotted lines manually drawn in a system definition diagram that visually represent a data flow relationship between two map components that has not yet been determined precisely.

pseudomap component
: An Integration Flow Designer object that is a placeholder for an executable map that has not yet been implemented.

PTF
: See program temporary fix.

public
: 1. In object-oriented programming, pertaining to a class member that is accessible to all classes.
  2. In the Java programming language, pertains to a method or variable that can be accessed by elements residing in other classes. (Sun)

public key
: In secure communication, an algorithmic pattern used to decrypt messages that were encrypted by the corresponding private key. A public key is also used to encrypt messages that can be decrypted only by the corresponding private key. Users broadcast their public keys to everyone with whom they must exchange encrypted messages. See also key, private key.

public key algorithm (PKA)
: An algorithm designed so that the key used for encryption is different from the key used for decryption. The decryption key cannot be derived, at least not in any reasonable amount of time, from the encryption key.

public key cryptography
: A cryptography system that uses two keys: a public key known to everyone and a private or secret key known only to the recipient of the message. The public and private keys are related in such a way that only the public key can be used to encrypt messages and only the corresponding private key can be used to decrypt them.

Public Key Cryptography Standards (PKCS)

A set of industry-standard protocols used for secure information exchange on the Internet. Domino Certificate Authority and Server Certificate Administration applications can accept certificates in PKCS format.

public key infrastructure (PKI)
A system of digital certificates, certification authorities, and other registration authorities that verify and authenticate the validity of each party involved in a network transaction. See also public key, SWIFTNet public key infrastructure.

public place
A shared place that is open to all portal users. The person who creates the place (and who automatically becomes the place manager) designates it as a public place during place creation.

public process
The interactions between a private business process and another process or participant.

publish

1. In UDDI, to advertise a web service so that other businesses can find it and bind with it. Service providers publish the availability of their services through a registry.
2. To make a web site public, for example by putting files in a path known to the HTTP server.
3. To send a rule to a server for storage and management.

publish/subscribe
A type of messaging interaction in which information, provided by publishing applications, is delivered by an infrastructure to all subscribing applications that registered to receive that type of information.

Q

QoS
See quality of service.

qualifier
A simple element that gives another generic compound or simple element a specific meaning. Qualifiers are used in mapping single or multiple occurrences. A qualifier can also be used to denote the namespace used to interpret the second part of the name, typically referred to as the ID.

quality of service (QoS)
A set of communication characteristics that an application requires. Quality of Service (QoS) defines a specific transmission priority, level of route reliability, and security level.

quartile analysis
A type of analysis that displays the value of the business measures boundaries at the 25th, 50th, or 75th percentiles of a frequency distribution divided into four parts, each containing a quarter of the population.

query

1. A reusable request for information about one or more model elements
2. A statement, or combination of statements, that is used to search a rule project (or other scope) and to select rule project elements that meet certain criteria.
3. A request for information from a database that is based on specific conditions: for example, a request for a list of all customers in a customer table whose balances are greater than USD1000.

queue
An object that holds messages for message-queueing applications. A queue is owned and maintained by a queue manager.

queue destination
A service integration bus destination that is used for point-to-point messaging.

queue manager
A component of a message queuing system that provides queuing services to applications.

queuing network
A group of interconnected components.

quiesce

1. To end a process or shut down a system after allowing normal completion of active operations.
2. To pause or alter the state of running processes on a computer, particularly those processes that might modify information stored on a disk during backing up, in order to guarantee a consistent and usable backup.

R

RACF
See Resource Access Control Facility.

RADIUS
See remote authentication dial-in user service.

range

1. The number of consecutive occurrences of the component in the data stream. The range is composed of two numbers separated by a colon.
2. The categorization of an attribute into different segments.

range operator
The logic used when comparing two attributes in a range.

rank
A property that determines the order of a reason.

rapid deployment tool
One of a set of tools to rapidly develop and deploy Java EE artifacts on the server and package the Java EE artifacts into the deployed EAR file.

RAR
See resource adapter archive.

RAS

See reliability, availability, and serviceability.

RAS attribute
: An attribute that the server applies to a request to control how the server processes that request. RAS attribute values can be defined with server-level, protocol-level, or request-level granularity. See also reliability, availability, and serviceability.

RAS granularity
: The extent to which a user can assign different RAS attribute values to different sets of requests within the same application server. The user can define RAS attribute values on a per-server, per-protocol, or per-request basis. See also protocol-level RAS granularity, reliability, availability, and serviceability, request-level RAS granularity, server-level RAS granularity.

Rational Unified Process (RUP)
: A configurable software development process platform that is used to assign and manage tasks and responsibilities within a development organization.

RC
: See return code.

RDMA
: See Remote Direct Memory Access.

read-through cache
: A sparse cache that loads data entries by key as they are requested. When data cannot be found in the cache, the missing data is retrieved with the loader, which loads the data from the back-end data repository and inserts the data into the cache.

real attribute
: An attribute that must have a value. See also pseudoattribute.

realize
: In the web diagram editor, to associate a node with an actual resource by creating that resource or by editing the node path so that it points to an existing resource. See also unrealized.

realm
: A collection of resource managers that honor a common set of user credentials and authorizations.

realm name
: The machine name of a user registry.

reason code
: A code assigned to identify a reason. Each scorecard can have multiple reasons.

reasoning strategy
: The strategy used to sort and compute the reasons returned from a scorecard table.

receiver
: A component that accepts documents from external partners and from back end applications and stores them in a file system for the Document Manager to process. Specifically, it receives a document over a supported transport protocol, writes the document and metadata relating to the document to the shared file system, records any transport-specific data to the metadata file, and completes any transport-specific technical acknowledgment.

receiver bean
: In extended messaging, a message-driven bean or a session bean. A message-driven bean is invoked when a message arrives at a JMS destination for which a listener is active. A session bean polls a JMS destination until a message arrives, gets the parsed message as an object, and can use methods to retrieve the message data.

recognition profile
: In the 3270 Terminal Services tool, a list of the identifiers that uniquely identify the state of a screen, that is, the set of conditions that apply to the screen at the time the screen was imported from the host. Each screen state needs to be uniquely defined in its own recognition profile.

recognition table
: In the 3270 terminal services development tool, the table that appears in the screen editor and provides a screen definition view and a recognition profile view of the screen that was imported.

record ID information object
: A Data Interchange Services object that contains control information for ROD document definitions. It identifies the type of ROD document definition being used and where the record ID, if any, is located in the records associated with the document definition.

record oriented data (ROD)
: The type of document definition used to describe proprietary document formats. One of the supported document syntax types.

record oriented data dictionary (ROD dictionary)
: A logical grouping of related ROD document definition components.

record oriented data document definition (ROD document definition)
: A description or layout of a proprietary document, comprising loops, records, structures, and fields.

record oriented data field (ROD field)
: A single item of data, such as a purchase order number, in a record oriented data (ROD) document definition. A ROD field corresponds to an EDI data element in an EDI document definition.

record oriented data loop (ROD loop)
: A group of consecutive records and loops that repeat together in a ROD document definition.

record oriented data record (ROD record)
: A group of logically related fields set up as a record in a ROD document definition.

record oriented data structure (ROD structure)
: A group of related fields in a ROD document definition, such as the fields making up the line item of an invoice. The record oriented data (ROD) structure corresponds to an EDI composite data element in an EDI document definition.

record processing pattern
: A job step pattern that reads and applies business logic to one record at a time from an input data source. The job step writes the results to an output data source and repeats the steps until all input records are processed.

recurring wait time trigger
: A trigger that is evaluated based on a period of time. For example, a recurring wait time trigger can be evaluated every 30 minutes and fire if it detects that a specific business situation has occurred.

recursion
: A programming technique in which a program or routine calls itself to perform successive steps in an operation, with each step using the output of the preceding step.

reentrance
: A situation where a thread of control attempts to enter a bean instance again.

refactor

To make changes across a set of artifacts without changing the behavior of the application or its relationships to other elements.

reference
: Logical names defined in the application deployment descriptor that are used to locate external resources for enterprise applications. At deployment, the references are bound to the physical location of the resource in the target operational environment.

reference binding
: A binding that maps a logical name (a reference) to a JNDI name.

reference delete conflict
: An edit conflict that occurs when one user has deleted an object that another user has referred to or vice versa.

referenced type
: An object that is referred to by a source object. See also associated type.

referential integrity

1. The condition that exists when all intended references from data in one column of a table to data in another column of the same or a different table are valid.
2. In Extensible Markup Language (XML) tools, the condition that exists when all references to items in the XML schema editor or DTD editor are automatically cleaned up when the schema is detected or renamed.

refresh pack
: A cumulative collection of fixes and new functions that moves the product up one modification level and a particular service level. For example, a refresh pack might move a product from Version 1 Release 1 Modification level 1 Fix Pack 5 to Version 1 Release 1 Modification level 2 Fix Pack 3. See also fix pack, interim fix.

region
: A contiguous area of virtual storage that has common characteristics and that can be shared between processes.

registered user (RU)
: A portal user who has a user ID and password for logging in to a portal. See also anonymous user, authenticated user.

registry
: A repository that contains access and configuration information for users, systems, and software.

regular expression
: A set of characters, meta characters, and operators that define a string or group of strings in a search pattern. See also Perl-compatible regular expression.

rejection code
: A code assigned when a score cannot be derived. Typically, one reject code per scorecard only is assigned.

relationship instance
: The runtime instantiation of the relationship. The relationship definition is a template for the relationship instance.

relationship management application (RMA)
: An application used to manage authorizations. Among other things, it converts bootstrap authorizations created by WebSphere BI for FN into the RMA authorizations required to satisfy FIN PV03.

relationship management data store (RMDS)
: A set of database tables in which WebSphere BI for FN stores data about bootstrap and relationship management application (RMA) authorisations.

relationship manager
: A tool for creating and manipulating relationship and role data at run time.

relationship role
: In EJB programming, a traversal of the relationship between two entity beans in one direction or the other. Each relationship that is coded in the deployment descriptor defines two roles.

relationship service
: A service used to model and maintain relationships across business objects and other data

relative type name
: The name of a type relative to another type. Relative type names are used when defining components, syntax items, and comment types.

release
: To send changed files from the workbench to the team server so that other developers on the team can catch up (synchronize) with the updated version.

release character
: The character that indicates that a separator or delimiter is to be used as text data instead of as a separator or delimiter. The release character must immediately precede the delimiter.

reliability, availability, and serviceability (RAS)
: A combination of design methodologies, system policies, and intrinsic capabilities that, taken together, balance improved hardware availability with the costs required to achieve it. Reliability is the degree to which the hardware remains free of faults. Availability is the ability of the system to continue operating despite predicted or experienced faults. Serviceability is how efficiently and nondisruptively broken hardware can be fixed. See also RAS attribute, RAS granularity.

remote
: Pertaining to a system, program, or device that is accessed through a communication line.

remote authentication dial-in user service (RADIUS)
: An authentication and accounting system that uses access servers to provide centralized management of access to large networks.

remote database
: A database to which a connection is made by using a database link, while connected to a local database. See also local database.

Remote Direct Memory Access (RDMA)
: A communication technique in which data is transmitted from the memory of one computer to that of another without passing through a processor. RDMA accommodates increased network speeds.

remote file system
: A file system residing on a separate server or operating system.

remote file transfer instance
: A file that contains information about the method used for remotely transferring a file.

remote home interface
: In enterprise beans, an interface that specifies the methods used by remote clients for locating, creating, and removing instances of enterprise bean classes. See also local home interface.

remote interface
: In EJB programming, an interface that defines the business methods that can be called by a client. See also home interface.

remote messaging, remote support, and web applications pattern
: A reusable deployment environment architecture for IBM Business Process Management products and solutions in which the functional components of the environment (messaging, support, web-based components, and application deployment) are split across four clusters.

remote messaging and remote support pattern
: A reusable deployment environment architecture for IBM Business Process Management products and solutions in which the functional components of the environment (messaging, support, web-based components, and application deployment) are split across three clusters. Web-based components reside on the support or the application-deployment cluster.

remote method
: A business method in the remote interface that is callable by a client. See also Remote Method Invocation.

Remote Method Invocation (RMI)
: A protocol that is used to communicate method invocations over a network. Java Remote Method Invocation is a distributed object model in which the methods of remote objects written in the Java programming language can be invoked from other Java virtual machines, possibly on different hosts. See also remote method.

Remote Method Invocation over Internet InterORB Protocol (RMI/IIOP)
: Part of the Java Platform, Standard Edition (Java SE) model that developers can use to program in the Java language to work with RMI interfaces, but use IIOP as the underlying transport.

Remote OSE
: A transport mechanism that is based on the Open Servlet Engine (OSE) protocol and is used to communicate between two separate machines in the application server environment.

Remote Procedure Call (RPC)
: A protocol that allows a program on a client computer to run a program on a server.

remote product installation
: A product installation onto a remote workstation that has a pre-installed operating system.

remote queue
: A queue that belongs to a remote queue manager. Programs can put messages on remote queues, but they cannot get messages from remote queues. See also local queue.

remote queue manager
: A queue manager to which a program is not connected, even if it is running on the same system as the program. See also local queue manager.

remove method
: In enterprise beans, a method defined in the home interface and invoked by a client to destroy an enterprise bean.

repeating data element
: An EDI data element or EDI composite data element that occurs more than once consecutively in an EDI segment.

repertoire
: Configuration information that contains the details necessary for building a Secure Sockets Layer (SSL) connection.

replica
: A server that contains a copy of the directory or directories of another server. Replicas back up servers in order to enhance performance or response times and to ensure data integrity.

replication
: 1. The process of copying objects from one node in a cluster to one or more other nodes in the cluster, which makes the objects on all the systems identical.
2. The process of maintaining a defined set of data in more than one location. Replication involves copying designated changes for one location (a source) to another (a target) and synchronizing the data in both locations.

replication domain
: A collection of application server components that share data. These components might include HTTP sessions, dynamic cache, stateful session beans, or the session initiation protocol (SIP) component.

replication entry
: A runtime component that handles the transfer of internal data.

reply message
: A type of message used for replies to request messages. See also report message, request message.

reply-to queue
: The name of a queue to which the program that issued an MQPUT call wants a reply message or report message sent.

report
: A formatted presentation of information relating to a model or to process simulation results. Reports can be viewed online, printed, or exported to a variety of file formats.

report container
: A group of settings that define the overall presentation of a report, including page dimensions and orientation, margin sizes, and options for displaying title, author, and summary information.

report message
: A type of message that gives information about another message. A report message can indicate that a message has been delivered, has arrived at its destination, has expired, or could not be processed for some reason. See also reply message, request message.

repository
: A persistent storage area for data and other application resources.

repository checkpoint
: A function that backs up copies of files from the master configuration repository. The backup files can be used to restore the configuration to a previous state if future configuration changes cause operational problems.

Representational State Transfer (REST)
: A software architectural style for distributed hypermedia systems like the World Wide Web. The term is also often used to describe any simple interface that uses XML (or YAML, JSON, plain text) over HTTP without an additional messaging layer such as SOAP. See also RESTful.

request
: In a request/response interaction, the role performed by a business object that instructs a connector to interact with an application or other programmatic entity.

request consumer binding
: A definition of the security requests for the request message that is received by a web service.

request flow

The flow of the message from the service requester.

Request for Comments (RFC)
In Internet communication, one of a series of numbered documents that describe Internet communication protocols.

request generator binding
A definition of the security requests for the request message that is sent to a web service.

request-level RAS granularity
The level of RAS granularity at which RAS attributes are assigned on a request-by-request basis to all requests for a particular request classification, such as HTTP requests that end in .jpg, a specific HTTP request for a URI such as /PlantsByWebSphere/index.html, or all IIOP requests for a particular EJB. See also RAS granularity.

request message
A type of message used to request a reply from another program. See also reply message, report message.

request metrics
A mechanism to monitor and troubleshoot performance bottlenecks in the system at an individual request level.

request receiver binding
A definition of the security requirements for the request message that is received from a request to a web service.

request/reply
A type of messaging application in which a request message is used to request a reply from another application. See also datagram.

request sender binding
A definition of the security requirements for the request message that is sent to a web service.

required component
A component that can be defined within a group type to represent a data object that must be present in the data. The component range minimum specifies how many occurrences of the data object are required.

resource
A person, piece of equipment, or material that is used to perform an activity.

Resource Access Control Facility (RACF)
An IBM licensed program that provides access control by identifying users to the system; verifying users of the system; authorizing access to protected resources; logging unauthorized attempts to enter the system; and logging accesses to protected resources.

resource adapter

1. Map input and output data sources that are used to retrieve and route data. Resource adapters provide access to databases, files, messaging systems, and other data sources and targets. Each adapter includes a set of adapter commands that can be used to customize its operation.
2. A system-level software driver that is used by an EJB container or an application client to connect to an enterprise information system (EIS). A resource adapter plugs in to a container; the application components deployed on the container then use the client API (exposed by adapter) or tool-generated, high-level abstractions to access the underlying EIS. (Sun) See also container, enterprise information system.

resource adapter archive (RAR)
A Java archive (JAR) file that is used to package a resource adapter for the Java 2 Connector (J2C) architecture.

resource class
An attribute of a resource that is used to group resources according to the subsystem to which they belong and the purpose for which they are used.

resource distribution report
A report, generated by the Customization Definition Program (CDP), that describes the resources required by an instance.

resource environment reference
A reference that maps a logical name used by the client application to the physical name of an object.

resource file
A file that is used to create, in a runtime environment, one or more resources of a particular class.

resource manager

1. An application, program, or transaction that manages and controls access to shared resources such as memory buffers and data sets. WebSphere MQ, CICS, and IMS are resource managers.
2. A participant, in the execution of a one-phase or two-phase commit, that has recoverable resources that could have been modified. The resource manager has access to a recovery log so that it can commit or roll back the effects of the logical unit of work to the recoverable resources.

resource manager local transaction (RMLT)
A resource manager view of a local transaction that represents a unit of recovery on a single connection that is managed by the resource manager.

resource model
A model that defines the resources used in business operations, including their roles, availability, and cost characteristics.

resource property
A property for a JDBC data source in a server configuration, for example the server name, user ID, or password.

Resource Recovery Services (RRS)
A component of z/OS that uses a sync point manager to coordinate changes among participating resource managers.

resource set
A collection of resources that are members of the same class and that share a common scope. A resource set also determines which other resource sets are its prerequisites and which place holders are used within the corresponding resource file templates.

response file
A file containing predefined values that is used instead of someone having to enter those values one at a time. See also silent installation.

response flow
The flow of the message from the service provider to the service requester.

response generator binding
A definition of the security requests for the response message that is sent to a web service.

response receiver binding
A definition of the security requirements for the response message that is received from a request to a web service.

response sender binding
A definition of the security requirements for the response message that is sent to a web service.

REST
See Representational State Transfer.

restart attribute

An attribute that specifies that processing of the input data should continue even though a data object of the component is invalid. The restart attribute provides instructions for handling errors encountered in a data stream and can be assigned to a component within a group type.

RESTful
: Pertaining to applications and services that conform to Representational State Transfer (REST) constraints. See also Representational State Transfer.

result
: The consequence of reaching an end event. Types of results include message, error, compensation, and signal. There can be multiple results, such as a result that produces a message and another result that sends a signal.

result event
: An action that is generated by the technology connectors and sent back to the runtime server to be processed as a new event.

result set
: A set of row values as returned by, for example, a cursor or procedure.

result tree
: The output document that is created when an XSL file is used to transform an XML file.

resume
: To continue execution of an application after an activity has been suspended.

RetePlus mode
: A rule execution mode for matching patterns with objects. The RetePlus mode is used by the rule engine to minimize the number of rules and conditions that need to be evaluated, compute which rules should be executed, and identify in which order these rules should be fired.

retraction
: The action of removing an object bound to a rule variable from the working memory.

return code (RC)
: A value returned by a program to indicate the result of its processing. Completion codes and reason codes are examples of return codes.

reverse proxy
: An IP-forwarding topology where the proxy is on behalf of the back-end HTTP server. It is an application proxy for servers using HTTP.

RFC
: See Request for Comments.

rich media
: In a web page, content that is aural, visual, or interactive, such as audio or video files.

Rich Site Summary (RSS)
: An XML-based format for syndicated web content that is based on the RSS 0.91 specification. The RSS XML file formats are used by Internet users to subscribe to websites that have provided RSS feeds. See also feed.

rich text
: A field that can contain objects, file attachments, or pictures as well as text with formatting options such as italics or boldface.

ripplestart
: An action where the system waits for a member in a cluster to start before starting the next member of the cluster.

RMA
: See relationship management application.

RMA authorisation
: An authorisation that has been processed by a relationship management application (RMA).

RM distribution file
: A file used to exchange relationship data with an relationship management application (RMA). It is the file that is created when you export bootstrap authorizations, and it is the file from which you import authorizations from an RMA.

RMDS
: See relationship management data store.

RMI
: See Remote Method Invocation.

RMI/IIOP
: See Remote Method Invocation over Internet InterORB Protocol.

RMLT
: See resource manager local transaction.

RM report
: A report used to determine whether all the relationships that are required when using PV03 exclusively have already been recorded, and whether corresponding authorisations already exist.

ROD
: See record oriented data.

ROD dictionary
: See record oriented data dictionary.

ROD document definition
: See record oriented data document definition.

ROD field
: See record oriented data field.

ROD loop
: See record oriented data loop.

ROD record
: See record oriented data record.

ROD structure
: See record oriented data structure.

role

1. A collection of access rights that can be assigned to a user, group of users, system, service, or application that enable it to carry out certain tasks.
2. A logical group of principals that provides a set of permissions. Access to operations is controlled by granting access to a role.
3. A job function that identifies the tasks that a user can perform and the resources to which a user has access. A user can be assigned one or more roles.
4. In a relationship, a role determines the function and participation of entities. Roles capture structure and constraint requirements on participating entities and their manner of participation. For example, in an employment relationship, the roles are employer and employee.

5. A description of a function to be carried out by an individual or bulk resource, and the qualifications required to fulfill the function. In simulation and analysis, the term role is also used to refer to the qualified resources.

role-based authorization
: The use of authorization information to determine whether a caller has the necessary privilege to request a service.

role-based security
: Security that provides access rights to certain files, business processes, web templates, and features, according to the permissions associated with the user account.

role mapping
: The process of associating groups and principals recognized by the container to security roles specified in the deployment descriptor.

rollback
: The process of restoring data that was changed by an application program or user.

root
: The user name for the system user with the most authority.

root element
: The implicit highest-level node of a parsed XML document. You may not always be able to predict which element will be the document element of a parsed instance, but it will always have a root node that you can count on being able to use for preliminary or setup processing.

root type
: The type from which all other types stem. The root type represents the data objects of all the types in the tree.

RosettaNet Partner Interface Process (PIP)
: A specialized system-to-system XML-based dialog that depicts the activities, decisions, and partner role interactions that fulfill a business transaction between two partners in a given supply chain.

routing policy
: A set of rules that determine how the server routes incoming requests.

row
: The horizontal component of a table, consisting of a sequence of values, one for each column of the table.

RPC
: See Remote Procedure Call.

RRS
: See Resource Recovery Services.

RSA encryption
: A system for public-key cryptography used for encryption and authentication. It was invented in 1977 by Ron Rivest, Adi Shamir, and Leonard Adleman. The security of the system depends on the difficulty of factoring the product of two large prime numbers.

RSS
: See Rich Site Summary.

RU
: See registered user.

rule

1. A condition that must be satisfied when a business activity is being performed.
2. A statement that defines or constrains some aspect of the business. See also business rule, event rule.
3. The criteria or circumstances that are defined to trigger an event. For example, rules can be triggered during entry to or exit from a zone and can be specified for a tag ID, class, or group.

rule analysis
: A mechanism for checking whether business rules are semantically consistent. Inconsistencies can be found either in the rule itself, or with respect to other rules.

RuleApp
: A deployment and management unit for Rule Execution Server that can contain one or more rule sets.

RuleApp archive
: An archive that allows RuleApps to be stored to a file system. RuleApp archives are saved in a strict directory structure.

RuleApp project
: An Eclipse project that performs the deployment of a RuleApp to a running Rule Execution Server.

rule artifact
: An item used to express a business policy in a business rule application. Action rules and decision tables are examples of rule artifacts.

Rule Designer
: A business rule application development tool integrated into the Eclipse development environment and dedicated to the creation and management of business rule applications.

RuleDoc
: A document containing business rules and rule metadata that can be edited.

rule editor
: A graphical tool used to create rules.

rule engine
: Accessible through a class, an engine that takes a rule set and executes it against a set of objects. The rule engine evaluates the conditions of rules in the rule set against the object in an object set to determine which rules are eligible to be executed.

Rule Execution Server
: A module that integrates into the Java EE environment, and as such offers the standard services of an application to execute, control, and monitor rule sets contained in RuleApps.

Rule Execution Server configuration project
: A project in which a server configuration persists. The configuration provides the information necessary to make a connection to the rule execution server when a RuleApp is applied.

Rule Execution Server console
: A web user interface that provides support for deploying RuleApps and the management of executable resources on Rule Execution Server.

rule execution set
: A collection of rules that are intended to be executed together.

rule flow

A method of controlling and ordering the execution of rule artifacts. A rule flow is defined in terms of tasks.

rule instance
: An occurrence of a rule that includes the combination of objects in the working memory that match the patterns specified in the rule. More than one instance of the same rule may exist in the agenda at any time because the rule patterns may be satisfied by more than one object or set of objects.

rule logic
: The business logic, which is expressed by a business rule, that consists of decisions that affect how a business responds to specific business conditions. For example, a decision that determines how much of a discount to give to a preferred customer is rule logic.

rule model
: A model that defines the set of items that are managed in a rule and event projects, and their associated properties.

rule package
: A container for organizing rule artifacts according to business logic. Rule packages become folders after they have been published to Decision Center.

rule perspective
: An Eclipse perspective that defines the initial set and layout of views in the workbench window that will be used in the development of a rule project.

rule project
: A type of project in which the user can manage and organize rule artifacts and business object models.

rule project template
: A partly completed rule project that can be used to create a series of rule projects with the same structure.

rules-based personalization
: Personalization technology that enables you to customize web content based on user needs and preferences, and business requirements.

rule schedule
: An interface for modifying the values of a business rule in the rule logic selection record.

rule session
: A runtime connection between a client and a rule engine. A rule session may consume initialized rule engine resources.

rule set
: 1. A set of rules that can be executed by the rule engine and includes rule artifacts and non-rule artifacts.
  2. An if-then statement that is composed of a set of textual statements, or rules, that are evaluated sequentially. if is the condition and then is the action. Each condition that evaluates to true is acted upon. See also action rule, decision table, if-then rule.

rule set extractor
: A mechanism for selecting the rules of the rule set to be deployed. Selection is typically based on the value of rule properties.

rule set interceptor
: A mechanism that allows services to be added to an execution component transparently and to be triggered automatically when certain events occur.

rule set parameter
: A parameter that can be defined to set and retrieve values on a rule set. Rule set parameters are accessible from outside of the rule set, and therefore are a bridge between the business logic and the application.

rule set signature
: The list of in, out, and inout parameters of a rule set.

rule set variable
: A variable that can be defined to be used in all the rule artifacts of a rule set.

rule task
: In a rule flow, a task that refers to rule artifacts and orders them.

rule template
: A partly completed business rule that can be used to create a series of rules with the same structure.

RunAs role
: A role used by a servlet or an enterprise bean component to invoke and delegate a role to another enterprise bean.

run map
: An executable map that is called using the RUN function.

runtime
: Pertaining to the time period during which a computer program is running.

run time
: The time period during which a computer program is running.

runtime environment
: A set of resources that are used to run a program or process.

runtime object
: An object used by the translator, such as a control string, code list, translation table, or user exit profile.

runtime rule selection
: In a rule task, a way to filter rule artifacts at run time. Runtime rule selection is expressed in rule statements.

runtime task
: A generated administrative action plan that contains recommendations to improve the health and performance of a runtime environment.

runtime topology
: A depiction of the momentary state of the environment.

RUP
: See Rational Unified Process.

S

SAAJ
: See SOAP with attachments API for Java.

SACL
: See State Adaptive Choreography Language.

SAF
: See System Authorization Facility.

SAG
: See SWIFTAlliance Gateway.

SAG MQ connection
: An entity within an SAG that encapsulates a WebSphere MQ connection.

SAML
: See Security Assertion Markup Language.

SAS
: See Secure Association Service.

SAX
: See Simple API for XML.

SCA
: See Service Component Architecture.

SCA component
: A building block of the Service Component Architecture, used to build SCA modules such as mediation modules.

SCA export binding
: A concrete definition that specifies the physical mechanism used by a service requester to access an SCA module; for example, using SOAP/HTTP.

SCA export interface
: An abstract definition that describes how service requesters access an SCA module.

SCA import binding
: A concrete definition that specifies the physical mechanism used by an SCA module to access an external service; for example, using SOAP/HTTP.

SCA import interface
: An abstract definition that describes how an SCA module accesses a service.

scalability
: The ability of a system to expand as resources, such as processors, memory, or storage, are added.

SCA module
: A module with interfaces that conforms to the Service Component Architecture (SCA).

SCA request
: A service request that conforms to the Service Component Architecture (SCA). An SCA module routes the request to a service provider, after having done any additional processing specified by the module.

SCA run time
: The server functions that provide support for the Service Component Architecture.

scenario
: 1. A set of actions representing a business process within the context of a collaboration. Scenarios can be used to partition collaboration logic. For example, if a collaboration handles one type of business object with various possible verbs, the user might develop Create, Update, and Delete scenarios. See also activity.
  2. A real or fictitious use case that can be used to validate the behavior of rules with test suites or simulations. Each scenario contains all the necessary information required for rules to execute properly.

scenario provider
: An object that defines how scenarios are loaded for test suites and simulations.

scheduler
: A service that provides time-dependent services.

schema
: A collection of database objects such as tables, views, indexes, or triggers that define a database. A schema provides a logical classification of database objects.

schema document definition
: A description or layout of an XML document based on an XML schema.

SCM
: See software configuration management.

scope
: 1. In web services, a property that identifies the lifetime of the object serving the invocation request.
  2. A specification of the boundary within which system resources can be used.

scorecard
: A set of measurements on a subject to help to make a business decision. See also scoring strategy.

scorecard property
: A property that defines the reasoning strategy and the scoring strategy. Scorecard properties are used together to determine the final score and the reason codes that are displayed.

scoring strategy
: A strategy to calculate the final score from each of the attribute scores for the overall scorecard. See also scorecard.

scratchpad area (SPA)
: A work area used in conversational processing to retain information from an application program across executions of the program.

screen
: The display that the user sees when connected to a 3270 application on the host system. A single 3270 application can include many screens, each of which has a purpose within the context of the application.

screen editor
: A 3270 terminal service development tool that enables a developer to create and modify recognition profiles for an imported screen and to assign names to the fields on the screen definition.

screen file
: The result of importing a screen definition from a 3270 application into the 3270 terminal service development workbench. A screen file represents a screen definition. The screen definition contains identifiers such as the number of fields on the screen and the row and column position of fields on the screen. There are multiple screen files per 3270 terminal service project. Each screen file can have multiple recognition profiles assigned to it.

screen import
: The process of importing a screen definition (in its current state) and saving it to a screen file within the 3270 terminal service tools workbench, for the purpose of generating recognition profiles and custom screen records. Use the 3270 terminal service recorder to import screens.

screen recognition

A runtime function that determines the state of a screen and processes the screen in accordance with the identifiers in the recognition profiles. Screen recognition compares the screen as presented by the 3270 application to the defined recognition profiles to determine which screen state applies.

screen state
: The set of conditions (at the time the screen was imported from the host) that determine the allowed and required processing on the screen. A screen state operates on input to change the status, cause an action, or result in a particular output screen. A single screen can have multiple states and the allowed user actions for the screen vary depending on which state the screen is in.

script
: A series of commands, combined in a file, that carry out a particular function when the file is run. Scripts are interpreted as they are run.

scripting
: A style of programming that reuses existing components as a base for building applications.

scriptlet
: A mechanism for adding scripting language fragments to a source file.

script package
: A compressed file consisting of an executable file and supporting files that are added to pattern topologies to customize the behavior of a cell.

SDK
: See software development kit.

SDO
: See Service Data Objects.

SDO repository
: A database that is used for storing and serving the Web Services Description Language (WSDL) definitions of web services. For example, the WSDL definitions for service integration bus-enabled web services are stored as service data objects in an SDO repository.

search center
: A portlet that enables site users to search for keywords. See also search collection, search service.

search collection
: A searchable collection of documents that can span multiple content sources. See also search center, search service.

search service
: A service that is used to define the configuration parameters for a search collection. A search service can be local, remote, inside the product, or outside the product. See also search center, search collection.

secret key
: A key that both encrypts and decrypts information. In symmetric cryptography, both communicating parties use a secret key. In asymmetric or public key cryptography, a public key and a private key are used to encrypt and decrypt information.

Secure Association Service (SAS)
: An authentication protocol used to communicate securely for the client principal by establishing a secure association between the client and server.

Secure FTP
: An FTP protocol that uses Secure Sockets Layer (SSL) protocol.

Secure Hash Algorithm (SHA)
: An encryption method in which data is encrypted in a way that is mathematically impossible to reverse. Different data can possibly produce the same hash value, but there is no way to use the hash value to determine the original data.

Secure Internet Protocol Network
: A SWIFT network based on the Internet Protocol (IP) and related technologies.

Secure Shell (SSH)
: A UNIX-based command interface and protocol for securely getting access to a remote computer.

Secure Sockets Layer (SSL)
: A security protocol that provides communication privacy. With SSL, client/server applications can communicate in a way that is designed to prevent eavesdropping, tampering, and message forgery. See also certificate authority.

security administrator
: The person who controls access to business data and program functions.

Security Assertion Markup Language (SAML)
: An XML framework for exchanging authentication and authorization information.

security attribute propagation
: The transportation of security attributes from one server to another server in an application server configuration.

security constraint
: A declaration of how to protect web content, and how to protect data that is communicated between the client and the server.

security domain
: The set of all the servers that are configured with the same user registry realm name.

security entity
: Entities used to specify what a user is authorized to do. Security entities include roles and users.

security permission
: Authorization granted to access a system resource.

security policy
: A written document that defines the security controls that you institute for your computer systems. A security policy describes the risks that you intend these controls to minimize and the actions that should be taken if someone breaches your security controls.

security role
: In Java EE, an abstract logical grouping of users that is defined by the application assembler. When an application is deployed, the roles are mapped to security identities, such as principals or groups, in the operational environment. (Sun)

security role reference
: A role that defines the access levels that users have and the specific resources that they can modify at those levels.

security token
: A representation of a set of claims that are made by a client that can include a name, password, identity, key, certificate, group, privilege, and so on.

segment
: An EDI logical unit of information. EDI segments are made up of data elements and composites. Segments are delimited; their components are separated by a delimiter.

segmentation
: A strategy that is used for building complex scorecards. This strategy defines segments or subgroups where a scorecard split might be necessary.

segmentation score

The maximum score of the scorecard when complex reasoning is not used. It is used to tune the contribution factor of a scorecard within the complex scorecard. The segmentation score is a property in a complex scorecard.

segment directory
: A file containing the format of all EDI segments in an EDI standard.

segment identifier
: A unique three-character identifier at the beginning of each EDI segment.

segment ID separator
: The character that separates the segment identifier from the EDI data elements in the EDI segment. See also data element delimiter.

segment terminator
: The character that marks the end of an EDI segment.

selector component
: A component that provides a means of interposing a dynamic selection mechanism between the client application and a set of target implementations.

sender bean
: In extended messaging, an enterprise bean (stateless session bean) that can be built to send asynchronous messages. A sender bean translates its method invocation into a JMS message, then passes that message to JMS. It can also retrieve a response message, translate that message into a result value, and return it to the caller.

sensor
: A program that reads information from a managed software system to create configuration information.

sequence flow
: A connecting object, represented by a solid graphical line, that shows the order of flow objects in a process or choreography. A sequence flow can cross the boundaries between swimlanes of a pool, but cannot cross the boundaries of a pool. There are two types of sequence flows: exception flow and normal flow.

sequence grouping
: The specification of the order in which entity beans update relational database tables.

sequence line
: An element that controls the sequence of activities and events during process execution.

sequence number
: A number assigned to each message exchanged between two nodes. The number is increased by one for each successive message. It starts from zero each time a new session is established.

sequential mode
: A rule execution mode for stateless pattern-matching. With this mode, rules can be processed sequentially, which can improve the speed of rule processing in specific cases. The sequential mode can be selected for individual tasks in a rule flow.

serialization
: In object-oriented programming, the writing of data in sequential fashion to a communications medium from program memory.

serializer
: A method for converting object data to another form such as binary or XML. See also deserialization.

series
: The consecutive occurrences of a component. In map rules, the [ ] characters denote an indexed member of a series.

servant region
: A contiguous area of virtual storage that is dynamically started as load increases and automatically stopped as load eases.

server
: A software program or a computer that provides services to other software programs or other computers. See also client, host.

server and bus environment
: The environment in which servers, service integration buses, and their resources are configured and managed.

server cluster
: A group of servers that are typically on different physical machines and have the same applications configured within them, but operate as a single logical server.

server configuration
: A resource that contains information required to set up and deploy to an application server.

server definition
: A definition for a computer that hosts a command server, to which systems under development in the Integration Flow Designer can be assigned as the intended execution server.

server group
: A group of Rule Execution Server for z/OS instances that are configured to be transferred to another if a server fails or if there is a planned outage. A server group can include one to 32 server instances.

server implementation object
: Enterprise beans that client applications require to access and implement the services that support those objects.

server-level RAS granularity
: The level of RAS granularity at which RAS attribute values are assigned on a server-wide basis. RAS attribute values defined at the server-level are assigned to all requests that the server processes. See also RAS granularity.

server message
: A message that is routed to a server application for processing, or a delivery notification that is routed to a client application to acknowledge the receipt of a client message by its destination.

server operation
: A collection of Java or non-Java process definitions that you can define to run on middleware servers. You can create server operations to enable or disable tracing, start or stop applications, query the running state of a server, and so on.

server project
: A project that contains information about test and deployment servers and their configurations.

server-side
: Pertaining to an application or component of an application that runs on a server rather than on the client. JSP and servlets are two examples of technologies that enable server-side programming.

server-side include (SSI)
: A facility for including dynamic information in documents sent to clients, such as current date, the last modification date of a file, and the size or last modification of other files.

service

1. In service-oriented architecture, a unit of work accomplished by an interaction between computing devices.
2. An offering that provides skilled assistance to customers. A service may include consulting, education and training, offering enabling services, managed operations, integration and application development. Services are distinguished from products by their intangibility, inseparability, perishability and variability. See also Advanced Integration service, General System service, integration service.
3. A component that accepts as input a message, and processes the message. For example, a service translates its payload into a different format, or routes it to one of several output queues. Most services are implemented as message flows or primitives.
4. A program that is used to implement activities or to perform one-time or recurring system tasks.

service application
: An application used to deploy mediation modules.

service bundle (SVB)
: A set of services that logically belong together, for example, because they share resources such as a status table or error processing queue. A service bundle contains the definition files for all resources required to provide the services, for example definition files for message flows, queues, and database tables. A service bundle has a unique name in the scope of an instance. A service bundle must be assigned to an organizational unit and loaded into a server before it is operational.

service bundle set
: A group of service bundles that are packaged together to simplify ordering. A definition file that defines the resource classes, resource file types, place holders, and server types that can be used by the service bundles in the set is associated with each service bundle set.

service class
: A group of work that has the same service goals or performance objectives, resource requirements, or availability requirements. For workload management, a service goal and, optionally, a resource group is assigned to a service class.

service client
: A requester that invokes functions in a service provider.

service component
: A collection of processes that represents a business service that publishes or operates on business data.

Service Component Architecture (SCA)
: An architecture in which all elements of a business transaction, such as access to web services, Enterprise Information System (EIS) service assets, business rules, workflows, databases and so on, are represented in a service-oriented way.

service context
: Part of a General InterORB Protocol (GIOP) message that is identified with an ID and contains data used in specific interactions, such as security actions, character code set conversion, and Object Request Broker (ORB) version information.

Service Data Objects (SDO)
: An open standard for enabling applications to handle data from heterogeneous data sources in a uniform way, based on the concept of a disconnected data graph. See also business object.

service definition
: One or more WSDL files that describe a service. Service definitions are produced by the Definition, Deployment, Adapter, Skeleton, and Proxy wizards.

service description
: The description of a web service, which can be defined in any format such as WSDL, UDDI, or HTML.

service destination
: A specialization of a service integration bus destination. Each service destination can directly represent the web service implementation or can indirectly represent the service through a Web Services Description Language (WSDL) document.

service document
: A document that describes a web service, for example a Web Services Description Language (WSDL) document.

service endpoint
: The physical address of a service which implements one or more interfaces.

service input queue
: The queue from which a service retrieves the messages it is to process. In WebSphere BI for FN, this queue is implemented as a WebSphere MQ local queue.

service integration bus (SIBus)
: A managed communication mechanism that supports service integration through synchronous and asynchronous messaging. A bus consists of interconnecting messaging engines that manage bus resources.

service integration bus link
: A link between messaging engines on different service integration buses. This enables requests and messages to pass between the buses.

service integration bus web services enablement
: A software component that enables web services to use IBM service integration technologies. This capability provides a quality of service choice and message distribution options for web services, with mediations that support message rerouting or modification.

service integration logic
: Integration logic on an enterprise service bus to mediate between requesters and providers. The logic performs a number of functions such as to transform and augment requests, convert transport protocols, and route requests and replies automatically

service integration technology
: Technology that provides a highly-flexible messaging system for a service-oriented architecture (SOA). This supports a wide spectrum of quality of service options, protocols, and messaging patterns. The technology supports both message-oriented and service-oriented applications.

service interface queue
: The queue into which applications place messages that are to be processed by a service. In WebSphere BI for FN, each OU that uses a particular service has its own service interface queue, and this queue is implemented as a WebSphere MQ alias queue.

service level
: A class of service that can be used in business policies to aggregate a set of implied service qualities.

service level agreement (SLA)

1. In IBM Business Process Management, a rule that a user creates to analyze the performance of business processes over time. An SLA establishes a condition that triggers a consequence and creates a report for one or more activities. Conditions in SLAs are based on a standard or custom key performance indictator (KPI).
2. A contract between a customer and a service provider that specifies the expectations for the level of service with respect to availability, performance, and other measurable objectives.

service message object (SMO)

A service data object that can exist only in a mediation flow component. The service message object is composed of a body and headers. The body contains the parameters of the invoked interface operation, and the headers may contain information such as service invocation, transport protocol, mediation exception, JMS properties, or correlation information.

service-oriented architecture (SOA)
A conceptual description of the structure of a software system in terms of its components and the services they provide, without regard for the underlying implementation of these components, services and connections between components.

service policy
A performance goal that is assigned to a specific application URI to help designate the business importance of different request types.

service portfolio
The collection of business services that a subscriber is entitled to use.

service project
A collection of related items used to build a service.

service provider
A company or program that provides a business function as a service.

service registry
A repository that contains all of the information that is required to access a web service.

service requester
The application that initiates an interaction with a web service. The service requester binds to the service by using the published information and calls the service.

services
Collections of network endpoints or ports that are used to aggregate a set of related ports.

service segment
The EDI segment used when an EDI document is enveloped (such as ISA, GS, ST, UNB, UNH, UNT, and so on).

service task
A task that uses a service implementation, such as a web service, that a BPM execution engine runs. This task does not require user interaction and does not appear on a task list.

service type definition
In Universal Discovery Description and Integration (UDDI), a description of specifications for services or taxonomies.

service virtualization
A virtualization that compensates for the differences in the syntactic details of the service interactions so that the service requester and provider do not have to use the same interaction protocol and pattern or the same interface, nor do they have to know the identities of the other participants.

servlet
A Java program that runs on a web server and extends the server functions by generating dynamic content in response to web client requests. Servlets are commonly used to connect databases to the web.

servlet archive
A file that contains the same components as a servlet application. Unlike web archives, servlet archives can have only a sip.xml deployment descriptor and not a web.xml deployment descriptor.

servlet container
A web application server component that invokes the action servlet and that interacts with the action servlet to process requests.

servlet filtering
The process of transforming a request or modifying a response without exposing the resource used by the servlet engine. See also filter.

servlet mapping
A correspondence between a client request and a servlet that defines their association.

session

1. A logical or virtual connection between two stations, software programs, or devices on a network that allows the two elements to communicate and exchange data for the duration of the session. See also transaction.
2. A series of requests to a servlet originating from the same user at the same browser.
3. In Java EE, an object used by a servlet to track user interaction with a web application across multiple HTTP requests.

session affinity
A method of configuring applications in which a client is always connected to the same server. These configurations disable workload management after an initial connection by forcing a client request to always go to the same server.

session bean
An enterprise bean that is created by a client and that typically exists only for the duration of a single client/server session. (Sun) See also entity bean, stateful session bean, stateless session bean.

session facade
A mechanism for separating the business and client tiers of an enterprise application by abstracting the data and business methods so that clients are not tightly coupled with the business logic and not responsible for data integrity. Implemented as session enterprise beans, session facades also decouple lower-level business components from one another.

Session Initiation Protocol (SIP)
A protocol for initiating interactive multi-media sessions. See also siplet.

session sequence number
A sequentially incremented 10 byte identifier that is assigned to each request unit in an LT session. It is formed by concatenating the 4 byte session number with a 6 byte sequence number.

setter method
A method whose purpose is to set the value of an instance or class variable. This capability allows another object to set the value of one of its variables. See also getter method.

severity code
A number that indicates the seriousness of an error condition.

SHA
See Secure Hash Algorithm.

shadow zone
A zone where the tags might not be visible temporarily because they are out of reach of the tag reader infrastructure or the signals are shielded. WebSphere Sensor Events assumes that a tag continues to be in the shadow zone at the last reported position after it has been seen. No alert is generated if the tag is no longer visible.

shard
An instance of a partition. A shard can be a primary or replica.
shared library file
A file that consists of a symbolic name, a Java class path and a native path for loading Java Native Interface (JNI) libraries. Applications that are deployed on the same node as this file can access this information.
shared lock
A lock that limits concurrently running application processes to read-only operations on database data. See also exclusive lock.
shared place
A place created for a community of people with a common purpose. Shared places can be public or restricted. The place creator (who automatically becomes the place manager) specifies whether a place is public or restricted during place creation.
shared service instance
An application capability that is made available in the cloud as an always-on, multitenant, elastic service for multiple users or applications.
shell script
A program, or script, that is interpreted by the shell of an operating system.
shortcut bar
In Eclipse, the vertical toolbar at the left side of the workbench window that contains buttons for open perspectives and for fast views.
shortest path
The processing path that takes the shortest time to complete of all parallel paths in a process instance, where each path considered begins at a start node or an input to the process and ends at a terminate node.
Short Message Service (SMS)
A message service that is used to send alphanumeric messages that are 160 characters or less between mobile phones.
short name
In personal communications, the one-letter name (A through Z) of the presentation space or emulation session.
short-running process
See microflow.
showcase
A business space that displays example widgets and data that describes a particular scenario to the user.
shredding
The process of breaking up an XML document for storage in database tables.
SIBus
See service integration bus.
side effect
An undesirable result caused by altering the values of nonlocal variables by a procedure or function.
signer certificate
The trusted certificate entry that is typically in a truststore file.
silent installation
An installation that does not send messages to the console but instead stores messages and errors in log files. A silent installation can use response files for data input. See also response file.
silent mode
A method for installing or uninstalling a product component from the command line with no GUI display. When using silent mode, you specify the data required by the installation or uninstallation program directly on the command line or in a file (called an option file or response file).
Simple API for XML (SAX)
An event-driven, serial-access protocol for accessing XML documents, used. A Java-only API, SAX is used by most servlets and network programs to transmit and receive XML documents. See also Document Object Model.
simple element
An item in the source or target document that does not contain child items, only data. For example: EDI data elements, ROD fields, XML attributes, and XML PCData values. See also element.
Simple Mail Transfer Protocol (SMTP)
An Internet application protocol for transferring mail among users of the Internet.
Simple Network Management Protocol (SNMP)
A set of protocols for monitoring systems and devices in complex networks. Information about managed devices is defined and stored in a Management Information Base (MIB). See also Management Information Base, SNMP trap.
simple type
A characteristic of a simple element that defines the type of data in a message (for example, string, integer, or float). In XML, a simple type cannot have element content and cannot carry attributes. See also complex type.
simple type name
The type name that appears next to the type icon in the type tree.
simulation
A faster-than-real-time performance of a process. Simulation enables organizations to observe how a process will perform in response to variations of inputs to the process, just as in a real-life work environment.
simulation profile
A copy of a process model and the elements on which it depends, augmented with simulation attributes, that you use to run a simulation. Each simulation profile in a snapshot is based on the process as it existed at the time that the snapshot was taken.
simulation snapshot
A record of the complete process model in a state that you want to preserve for simulation purposes. This record contains a copy of all the project elements the process uses, as well as any additional project elements.
single authorization
A setting allowing an action to be carried out by a single person. See also dual authorization.
single-cluster pattern
A reusable deployment environment architecture for IBM Business Process Management products and solutions in which the functional components of the environment (messaging, support, web-based components, and application deployment) are on one cluster.
single-occurrence mapping
A form of mapping in which a specific occurrence of a repeating compound or simple element is mapped to a compound or simple element.
single sign-on (SSO)
An authentication process in which a user can access more than one system or application by entering a single user ID and password.

singleton
: A class that can be instantiated only once. A singleton class cannot be an interface.

SIP
: See Session Initiation Protocol.

siplet
: A Session Initiation Protocol (SIP) servlet that performs SIP signaling to back-end applications of the SIP server, such as the presence server or instant messaging server. See also Session Initiation Protocol.

situation
: A significant occurrence that is detected when a set of conditions are met. For example, exceeding the limits of a Key Performance Indicator (KPI).

situation event
: A Common Base Event that is emitted when a defined situation occurs.

sized attribute
: An attribute that can be assigned to one or more components within a group type, whose value specifies the size, in bytes, of the component immediately following it.

skeleton
: Scaffolding for an implementation class.

skin
: An element of a graphical user interface that can be changed to alter the appearance of the interface without affecting its functionality.

SLA
: See service level agreement.

smart card
: An intelligent token that is embedded with an integrated circuit chip that provides memory capacity and computational capabilities.

smart folder
: A view on a rule project that allows the user to display project elements grouped by property.

SMO
: See service message object.

SMP/E
: See SMP/E for z/OS.

SMP/E for z/OS (SMP/E)
: An IBM licensed program that is used to install software and software changes on z/OS systems.

SMS
: See Short Message Service.

SMTP
: See Simple Mail Transfer Protocol.

snapshot
: 1. The state of a project or branch as captured at a specific time.
  2. A capture of data at a point time for performance analysis.
  3. In Business Process Manager, a capture of a process application or toolkit at a point in time. With a snapshot, a user can revert to a different version of a process or artifact.

snippet
: An excerpt of source code.

SNL
: See SWIFTNet Link.

SNMP
: See Simple Network Management Protocol.

SNMP trap
: An SNMP message sent from the SNMP agent to the SNMP manager. The message is initiated by the SNMP agent and is not a response to a message sent from the SNMP manager. See also Simple Network Management Protocol.

SOA
: See service-oriented architecture.

SOAP
: A lightweight, XML-based protocol for exchanging information in a decentralized, distributed environment. SOAP can be used to query and return information and invoke services across the Internet. See also web service.

SOAP encoding
: Rules for serializing data over the SOAP protocol. SOAP encoding is based on a simple type system that is a generalization of the common features found in type systems in programming languages, databases, and semi-structured data.

SOAP with attachments API for Java (SAAJ)
: An application programming interface (API) that is used to send XML documents over the Internet from a Java base.

Society for Worldwide Interbank Financial Telecommunication (SWIFT)
: An industry-owned cooperative that supplies standardized messaging services and software to financial institutions.

socket
: An identifier that an application uses to uniquely identify an end point of communication. The user associates a protocol address with the socket by associating a socket address with the socket.

Sockets Secure
: A client/server architecture that transports TCP/IP traffic through a secure gateway. A SOCKS server performs many of the same services that a proxy server does.

softcopy
: One or more files that can be electronically distributed, manipulated, and printed by a user.

software configuration management (SCM)
: The tracking and control of software development. SCM systems typically offer version control and team programming features.

software development kit (SDK)
: A set of tools, APIs, and documentation to assist with the development of software in a specific computer language or for a particular operating environment.

solution

A set of one or more related case types, tasks, steps, and other components that provide documents, data, business processing, and routing to case workers. For example, a solution for a human resources department might include a case type for new hires, a case type for retirement, and a case type for employee termination.

source based map
: A map based on the order elements that are defined in the source document definition.

source code
: A computer program in a format that is readable by people. Source code is converted into binary code that can be used by a computer.

source document
: A document that is going to be translated.

source document definition
: A description of a document layout that is used to identify the format of the source document for a translation.

source interface
: In a mediation flow component, the interface that allows the service requester to access the mediation flow through an export.

source map component
: An object that references an executable map within a source map file.

source tree
: The XML input document that is transformed by an XSL stylesheet.

SPA
: See scratchpad area.

spec
: See specification.

special-subject
: Generalization of a particular class of users; a product-defined entity independent of the user registry.

special variable
: A variable that is similar to a local or global variable, except that it is predefined in Data Interchange Services. Special variables are created during translation at the start of a document and cannot be created or maintained by the user.

specification (spec)
: A declarative description of what something is or does.

SPUFI
: See SQL Processor Using File Input.

SQL
: See Structured Query Language.

SQLJ
: See Structured Query Language for Java.

SQL Processor Using File Input (SPUFI)
: A facility of the TSO attachment subcomponent that enables the DB2I user to run SQL statements without embedding them in an application program.

SQL query
: A component of certain SQL statements that specifies a result table.

SSH
: See Secure Shell.

SSH File Transfer Protocol
: A network protocol that provides the ability to transfer files securely over any reliable data stream.

SSI
: See server-side include.

SSL
: See Secure Sockets Layer.

SSL channel
: A type of channel within a transport chain that associates a Secure Sockets Layer (SSL) configuration repertoire with the transport chain.

SSO
: See single sign-on.

stack
: An area in memory that typically stores information such as temporary register information, values of parameters, and return addresses of subroutines and is based on the principle of last in, first out (LIFO).

stack frame
: A section of the stack that contains the local variables, arguments, and register contents for an individual routine, as well as a pointer to the previous stack frame.

stacking number
: The number of application servers that are required for a dynamic cluster to use all the power of a node.

staff activity
: An activity in a process that queries human interaction for decisions on how to proceed. A staff activity is used in a long-running process where the process will halt to await the outcome of the human interaction.

staging
: The process of returning return data or an object from an offline or low-priority device to an online or higher priority device, typically on demand of the system or on request of the user.

stand-alone
: Independent of any other device, program, or system. In a network environment, a stand-alone machine accesses all required resources locally.

stand-alone server
: 1. A fully operational server that is managed independently of all other servers, using its own administrative console.
  2. A catalog service or container server that is managed from the operating system that starts and stops the server process.

stand-alone task
: A unit of work that exists independently of a business process, and implements human interaction as a service. See also human task, inline task.

standard envelope
: See EDI envelope.

standard portlet

A portlet that complies with one of the OASIS portlet standards: JSR168 or JSR286.

Standard Widget Toolkit (SWT)
: An Eclipse toolkit for Java developers that defines a common, portable, user interface API that uses the native widgets of the underlying operating system. See also Abstract Window Toolkit, Swing Set.

star schema
: A type of relational database schema that is composed of a set of tables comprising a single, central fact table surrounded by dimension tables.

start event
: An event that indicates where a process starts. The start event starts the flow of the process and does not have any incoming sequence flow but can have a trigger. Start event types are none, message, timer, ad hoc, and error. See also ad hoc start event, error start event, message start event, none start event, timer start event.

start node

1. A node that identifies where a rule flow begins. A rule flow has one and only one start node.
2. A node that identifies where a process begins.

star topology
: In network architecture, a network topology in which every node on the network is connected to a central node or "hub," through which they communicate with each other.

stash file
: A file that hides other data files within it.

state
: In a business state machine, one of several discrete individual stages that are organized in sequence to compose a business transaction.

State Adaptive Choreography Language (SACL)
: An XML notation that is used to define state machines.

stateful session bean
: A session bean that acts on behalf of a single client and maintains client-specific session information (called conversational state) across multiple method calls and transactions. See also session bean, stateless session bean.

stateless session bean

1. A session bean that is a collection of operations. The server can optimize resources by reusing bean instances on every method call.
2. A session bean with no conversational state. All instances of a stateless bean are identical. (Sun) See also session bean, stateful session bean.

state machine
: A behavior that specifies the sequences of states that an object or an interaction goes through during its life in response to events, together with its responses and actions.

static
: A Java programming language keyword that is used to define a variable as a class variable.

static analysis
: The process of extracting targeted types of information on the models in their static form. This differs from dynamic analysis, which extracts information based on the results of process simulations.

static cluster
: A group of application servers that participates in workload management. Membership for the static cluster is manually managed.

static web page
: A web page that can be displayed without the additional client- or server-side processing that would be required for JavaServer Pages, servlets, or scripts.

static web project
: A project that contains resources for a web application with no dynamic content such as servlets or JavaServer Pages (JSP) files, or Java code. A static web project can be deployed to a static HTTP server and does not require additional application server support.

step
: A stage in a workflow where a distinct, well-defined action is performed. Each step on a workflow map represents a specific activity or task in the business process described by the map. For example, in insurance claims processing, verify account number and calculate deductible could be individual steps. A workflow consists of two or more steps. See also case property.

stored procedure
: A block of procedural constructs and embedded SQL statements that is stored in a database and that can be called by name. Stored procedures allow an application program to be run in two parts, one on the client and the other on the server, so that one call can produce several accesses to the database.

STP
: See straight through processing.

straight through processing (STP)
: A series of uninterrupted electronic processes across and throughout an enterprise which (1) secures an initial transaction as an electronic message, (2) transforms and transports it to its initial execution/processing location and (3) passes it through the processing cycle with little, if any, human intervention.

stream
: In the CVS team programming environment, a shared copy of application resources that is updated by development team members as they make changes. The stream represents the current state of the project.

stream decryption
: A symmetric algorithm that decrypts data one bit or byte of data at a time.

stream encryption
: A symmetric algorithm that encrypts data one bit or byte of data at a time.

stream object
: An object used in the TX Programming Interface that permits overrides to the loaded map input and output specifications.

string
: In programming languages, the form of data used for storing and manipulating text.

structure
: A series of elements that have been graded or ranked in some useful manner. In WebSphere Business Modeler, a graphical representation of the relationships between different real entities in an organization.

Structured Query Language (SQL)
: A standardized language for defining and manipulating data in a relational database.

Structured Query Language for Java (SQLJ)

A standard for embedding SQL in Java programs, defining and calling Java procedures and user-defined functions, and using database structured types in Java.

structured viewing
: The tabular aspect of the Design view of the XML editor that separates the structural constituents of an XML document, such as elements and attribute types, from values, such as attribute values and textual content.

Struts
: An open source framework designed to help developers create web applications that keep database code, page design code, and control flow code separated from each other.

Struts action
: A class that implements a portion of a web application and returns a forward. The superclass for a Struts action is called the Action class.

Struts module
: A Struts configuration file and a set of corresponding actions, form beans, and web pages. A Struts application comprises at least one Struts module.

Struts project
: A dynamic web project with Struts support added.

stub
: A small program routine that substitutes for a longer, possibly remote, program. For example, a stub might be a program module that transfers procedure calls (RPCs) and responses between a client and a server. In web services, a stub is an implementation of a Java interface generated from a Web Services Description Language (WSDL) document.

style sheet
: A specification of formatting instructions that, when applied to structured information, provides a particular rendering of that information (for example, online or printed). Different style sheets can be applied to the same piece of structured information to produce different presentations of the information.

subarea
: An area that is nested within another area.

subclass
: In Java, a class that is derived from a particular class, through inheritance.

subelement
: In UN/EDIFACT EDI standards, an EDI data element that is part of an EDI composite data element. For example, an EDI data element and its qualifier are subelements of an EDI composite data element.

subelement separator
: A character that separates the subelements in an EDI composite data element.

subflow
: A sequence of processing steps, implemented using message flow nodes, that is designed to be embedded in a message flow or in another subflow. A subflow must include at least one Input or Output node. A subflow can be executed by a broker only as part of the message flow in which it is embedded, and therefore it cannot be deployed. See also message flow.

subflow node
: A message flow node that represents a subflow. See also primitive.

subflow task
: In a rule flow, a task that refers to another rule flow. A subflow task can reference a rule flow in the current project, or in a parent project.

subnet
: See subnetwork.

subnet mask
: For internet subnetworking, a 32-bit mask used to identify the subnetwork address bits in the host portion of an IP address.

subnetwork (subnet)
: A network that is divided into smaller independent subgroups, which still are interconnected.

subprocess
: A local process that is also a part of another process. See also deployment manager.

subquery
: In SQL, a subselect used within a predicate, for example, a select-statement within the WHERE or HAVING clause of another SQL statement.

subscriber
: The consumer of a business service.

subscription
: A record that contains the information that a subscriber passes to its local broker to describe the publications that it wants to receive.

substate
: A state that is part of a composite state.

subsystem component
: An Integration Flow Designer object that references another system which a user has defined.

subtree
: A branch of a type tree that includes a type and all of the subtypes that stem underneath it.

subtype
: A type that extends or implements another type; the supertype.

superclass
: In Java, a class from which a particular class is inherited, perhaps with one or more classes in between.

superset
: Given two sets A and B, A is a superset of B if and only if all elements of B are also elements of A. That is, A is a superset of B if B is a subset of A.

supertype
: In a type hierarchy, a type that subtypes inherit attributes from.

suspend
: To pause a process instance.

SVB
: See service bundle.

SWIFT
: See Society for Worldwide Interbank Financial Telecommunication.

SWIFT address
: See bank identifier code.

SWIFTAlliance Gateway (SAG)

A SWIFT interface product extending SWIFTNet Link by additional services such as profile-based processing, and offering a WebSphere MQ interface.

SWIFTNet FileAct
: A SWIFT interactive communication service supporting exchange of files between two applications.

SWIFTNet FIN
: A SWIFT service providing FIN access using the Secure IP Network (SIPN) instead of the SWIFT Transport Network (STN). See also FIN.

SWIFTNet FIN batching
: The transporting of more than one FIN message within a single InterAct message.

SWIFTNet InterAct
: A SWIFT interactive communication service supporting exchange of request and response messages between two applications.

SWIFTNet Link (SNL)
: A SWIFT mandatory software product to access all SWIFTNet services.

SWIFTNet PKI
: See SWIFTNet public key infrastructure.

SWIFTNet public key infrastructure (SWIFTNet PKI)
: SWIFT's mandatory security software and hardware installed with SWIFTNet Link. See also public key infrastructure.

SWIFTNet service
: A SWIFT IP-based communication service that runs on the SIPN.

SWIFTNet service application
: An application that uses SWIFTNet services. Financial organizations such as Continuous Linked Settlement (CLS) or the Global Straight Through Processing Association (GSTPA) offer such applications to financial institutions.

SWIFT transport network
: A SWIFT network providing FIN and IFT service based on X.25 technology.

swimlane
: 1. See pool.
  2. A visually separated row within a process flow diagram that groups all the activities in the process that are performed by a particular combination of roles, resources, organization units, or locations.

Swing Set
: A collection of GUI components that runs consistently on any operating system that supports the Java virtual machine (JVM). Because they are written entirely in the Java programming language, these components provide functionality above and beyond that provided by native-platform equivalents. See also Abstract Window Toolkit, Standard Widget Toolkit.

SWT
: See Standard Widget Toolkit.

symbolic link
: A type of file that contains a pointer to another file or directory.

symmetric algorithm
: An algorithm where the encryption key can be calculated from the decryption key and vice versa. In most symmetric algorithms, the encryption key and the decryption key are the same.

synchronization
: The process of publishing and updating changes in a rule to a server.

synchronize
: To add, subtract, or change one feature or artifact to match another.

synchronous process
: A process that starts by invoking a request-response operation. The result of the process is returned by the same operation.

synchronous replica
: A shard that receives updates as part of the transaction on the primary shard to guarantee data consistency, which can increase the response time compared with an asynchronous replica. See also asynchronous replica.

sync point
: A point during the processing of a transaction at which protected resources are consistent.

sync point manager
: A function that coordinates the two-phase commit process for protected resources, so that all changes to data are either committed or backed out.

syntax
: The rules for the construction of a command or statement.

syntax highlighting
: In source editors, the ability to differentiate text and structural elements, such as tags, attributes, and attribute values, using text highlighting differences, such as font face, emphasis, and color.

syntax object
: One or more characters used as separators between portions of data. A syntax object can be a number separator, a delimiter, a terminator, an initiator, or a release character.

syntax type
: A category used to classify different formats of documents. Data Interchange Services supports three syntax types: XML, EDI, and record oriented data. The user can map and translate between any of these syntax types.

synthesized event
: See synthetic event.

synthetic event
: An event that is triggered in response to a condition that was detected while processing the current event. Unlike an action, which is also triggered in response to a condition that was detected during the processing of the current event, a synthetic event is not sent to an external system.

sysplex
: A set of z/OS systems that communicate with each other through certain multisystem hardware components and software services.

system
: A collection of referenced executable maps that are organized into a unit.

System Authorization Facility (SAF)
: A z/OS interface with which programs can communicate with an external security manager, such as RACF.

system configuration administration
: The administration of configuration object types, organizational units, and roles. This is carried out after the product has been installed and is running.

system definition diagram
: The graphical representation of a system viewed within a system window in the Integration Flow Designer. A user can interact with system definition diagrams to design systems.

system logger
: An integrated logging facility that is provided by MVS and can be used by system and subsystem components. For example, it is used by the CICS log manager.

system menu
: A drop-down menu that is activated by clicking the icon at the left of a window title bar and that allows users to restore, move, size, minimize, or maximize the window.

systems analyst
: A specialist who is responsible for translating business requirements into system definitions and solutions.

system task
: See service task.

system window
: A window in the Integration Flow Designer in which system definition diagrams are created, maintained, and displayed.

T

tag

1. An item that contains identifying information about a person or device. Tags enable tracking and monitoring of assets within locations, areas, and zones.
2. A word or phrase that users create and assign to an asset. Users create tags to develop search criteria that is meaningful to themselves.
3. In UN/EDIFACT EDI Standards, the segment identifier. In export and import, a code that is assigned to each field in the database and used to identify the field in the export file. Such export files are known as tagged files.

taglib directive
: In a JSP page, a declaration stating that the page uses custom tags, defines the tag library, and specifies its tag prefixes. (Sun)

tag library
: In JSP technology, a collection of tags identifying custom actions described using a taglib descriptor and Java classes. A JSP tag library can be imported into any JSP file and used with various scripting languages. (Sun)

TAI
: See trust association interceptor.

target

1. See receiver.
2. The destination for an action or operation.
3. A value that a Key Performance Indicator (KPI) should achieve, such as "300" or "5 days."

target based map
: A map based on the order elements that are defined in the target document definition.

target CDD
: A customization definition document (CDD) to which placeholders have been added, and for which placeholder values have been specified. A target CDD describes a particular target customization definition.

target component
: A component that is the final target of a client service request.

target customization definition
: A customization definition that describes a changed version of a current customization definition. Each target customization definition has a target CDD that describes it.

target document
: A translated version of a document.

target document definition
: A description of the document layout used to create an output document from a translation.

target document definition window
: One of the pages on the Details tab of the Data Transformation Map Editor and the Functional Acknowledgement Map Editor. It displays the target document definition.

target namespace
: A unique logical location for information about the service that associates a namespace with a WSDL location.

target service
: A service that exists outside of the gateway.

task

1. The basic unit of organization in a rule flow.
2. An atomic activity that is included within a process. A task is used when the work in the process is not broken down to a finer level of process model detail. Generally, an end-user, an application, or both perform the task. A task object is the same shape as the subprocess, which is a rectangle that has rounded corners. See also case property.
3. A unit of work to be accomplished by a device or process.
4. One or more actions associated with a case. A task has one or more steps that must be completed to finalize the task. For example, a task might be to review new hire applications. A case is not complete until all required tasks are completed or manually disabled. Each task has roles that are associated with it. See also activity, case.

task-related user exit (TRUE)
: A user exit program that is associated with specified events in a particular task, rather than with every occurrence of a particular event in CICS processing (as is the case with global user exits).

taxonomy
: The hierarchical classification of information according to a known system that is used to easily discuss, analyze, or retrieve that information.

TC
: See test case.

TCP
: See Transmission Control Protocol.

TCP channel
: A type of channel within a transport chain that provides client applications with persistent connections within a local area network (LAN).

TCP/IP
: See Transmission Control Protocol/Internet Protocol.

TCP/IP monitoring server
: A runtime environment that monitors all requests and responses between a web browser and an application server, as well as TCP/IP activity.

TDCC
: See Transportation Data Coordinating Committee.

team development
: The practice of several members of a team contributing to a single project, with the potential for multiple team members to work in parallel on the same files.

team support
: The component that interacts with a repository to share and version projects and project data. See also version control.

technical rule
: A rule written in a technical rule language, such as ILOG Rule Language (IRL).

technology adapter
: An adapter that is designed for interactions that conform to a specific technology. For example, the WebSphere Adapter for FTP, is an intermediary through which an integration broker sends data to a file system that resides on a local or remote FTP server.

technology connector
: An API that passes data between the event processing server (runtime server) and external systems using a standard protocol such as SMTP, HTTP, FTP, or SOAP.

template
: 1. A grouping of elements that share common properties. These properties might be defined only once and are inherited by all elements using the template. In Java terms, this is an abstract class.

template library
: The database, known as the Portal Template Catalog, that stores place template specifications and portlets forms, subforms, and profiles.

temporary file system (TFS)
: A temporary, in-memory physical file system that supports in-storage mountable file systems. Normally, a TFS runs in the kernel address space, but it can be run in a logical file system (LFS) colony address space.

temporary page
: A page that closes and cannot be reopened after a user navigates away from it.

terminal file
: The resource in a 3270 service project that contains the information necessary for connecting to the host system during build time. Terminal files are automatically generated when the 3270 terminal service project is created. In the Navigator view, if a terminal file is selected, the 3270 terminal service recorder opens in the editor area.

terminate end event
: An end event that will stop all parallel activities within its process level and all lower process levels. See also end event.

terminate node
: A node that marks the end of a process. When a flow reaches a terminate node while the process is running, the process immediately terminates, even if there are other currently executing flows within the process.

terminator
: A syntax object that signifies the end of a data object. For example, a carriage return/linefeed at the end of a record might be the record's terminator.

test case (TC)
: A set of tasks, scripts, or routines that automate the task of testing software.

test configuration
: A property of the integration test client that is used to specify modules for testing and to control the tests.

test harness
: A series of script files used to enable a DB2 database for use by the DB2 XML Extender. A test harness is optionally created when a DAD file is generated from a relational database to XML mapping. Once enabled, it tests composing XML from data as well as decomposing XML files into relational data.

test pattern
: A template used for the automatic generation of component tests. There are several test patterns available for testing both Java and EJB components. See also component test.

test suite
: 1. A collection of test cases that define test behavior and control test execution and deployment.
2. A set of usage scenarios with which the user can verify that business rules are correctly designed and written. Running test suites produces a report comparing the expected results and the actual results obtained when applying rules to the scenarios.

text annotation
: An artifact that provides additional textual information about a BPMN diagram.

TFS
: See temporary file system.

theme
: The style element that gives a place a particular look. The portal provides several themes, similar to virtual wallpaper, which can be chosen when creating a place.

thin application client
: A lightweight, downloadable Java application run time capable of interacting with enterprise beans.

thin client
: A client that has little or no installed software but has access to software that is managed and delivered by network servers that are attached to it. A thin client is an alternative to a full-function client such as a workstation.

thread
: A stream of computer instructions that is in control of a process. In some operating systems, a thread is the smallest unit of operation in a process. Several threads can run concurrently, performing different jobs.

thread contention
: A condition in which a thread is waiting for a lock or object that another thread holds.

threshold
: A setting that applies to an interrupt in a simulation that defines when a process simulation should be halted based on a condition existing for a specified proportion of occurrences of some event.

throughput
: The measure of the amount of work performed by a device, such as a computer or printer, over a period of time, for example, number of jobs per day.

throwing message intermediate event
: An intermediate event that sends a message. See also intermediate event.

thumbnail
: An icon-sized rendering of a larger graphic image that permits a user to preview the image without opening a view or graphical editor.

TID
: See transaction identifier.

time-delayed rule
: The delay before an event rule or event rule group is evaluated.

timeout
: A time interval that is allotted for an event to occur or complete before operation is interrupted.

timer
: An event that is triggered by an occurrence at a specific time.

timer event
: An event that is triggered when a time condition is satisfied. See also intermediate event.

timer intermediate event
: An intermediate event that is triggered when a time condition is satisfied. A timer intermediate event can delay the flow of the process or can generate a timeout for activities that exceed the time condition.

timer start event
: A start event that is triggered when a time condition is satisfied. A timer start event is used only for event subprocesses. See also start event.

Time Sharing Option (TSO)
: A base element of the z/OS operating system with which users can interactively work with the system. See also Interactive System Productivity Facility.

timetable
: A schedule of times. In business process modeling, timetables are typically associated with resources or costs. For resources, timetables indicate availability (such as Monday to Friday). For costs, timetables are useful if the cost varies with time of day (such as electricity) or time of year (such as seasonal foods).

time to live (TTL)
: The time interval in seconds that an entry can exist in the cache before that entry is discarded.

timing constraint
: A specialized validation action used to measure the duration of a method call or a sequence of method calls. See also validation action.

tip
: The current working version of a process application or toolkit.

Tivoli Performance Viewer
: A Java client that retrieves the Performance Monitoring Infrastructure (PMI) data from an application server and displays it in various formats.

TLS
: See Transport Layer Security.

token
: 1. A particular message or bit pattern that signifies permission or temporary control to transmit over a network.
  2. A marker that progresses through a process instance and indicates which element is currently running. A process instance can generate several tokens. A token can take only one path.

token-bucket
: A mechanism that controls data flow. As an application requests permission into a network, the token bucket adds characters (or tokens) into a buffer (or bucket). If enough room is available for all the tokens in the bucket, the application is allowed to enter the network.

toolkit
: A container where artifacts can be stored for reuse by process applications or other toolkits.

top-down development
: In web services, the process of developing a service from a Web Services Description Language (WSDL) file. See also bottom-up development.

top-down mapping
: An approach for mapping enterprise beans to database tables, in which existing enterprise beans and their design determines the database design.

topology
: The physical or logical mapping of the location of networking components or nodes within a network. Common network topologies include bus, ring, star, and tree.

track
: An optional subdivision in a process application that is based on team tasks, process application versions, or both. When enabled, tracks allow parallel development to occur with isolation from changes in other tracks. For example, using tracks one team can fix the current version of a process, while another team builds a completely new version based on new external systems and a new corporate identity.

track event
: An event that tracks certain data as it passes through the event run time.

tracking group
: A group of tracked process variables and data, such as KPIs, from one or more BPDs or process applications. Tracking groups are used to monitor performance and report analyses of information.

tracking intermediate event
: An intermediate event that indicates a point in a process when runtime data is captured for reporting. See also intermediate event.

trading partner

A company, such as a manufacturer or a supplier, that agrees to exchange information using electronic data interchange, or an entity in an organization that sends and receives documents that are translated. See also external partner.

trailer
A control structure that indicates the end of an electronic transmission.

transaction

1. A subprocess that represents a set of coordinated activities that are carried out by independent, loosely coupled systems in accordance with a contractually defined business relationship. This coordination leads to an agreed, consistent, and verifiable outcome across all participants.
2. A process in which all of the data modifications that are made during a transaction are either committed together as a unit or rolled back as a unit.

transaction class
A subcontainer of a service policy that is used for finer-grained monitoring.

transaction ID
See transaction identifier.

transaction identifier (TID, transaction ID, XID)
A unique name that is assigned to a transaction and is used to identify the actions associated with that transaction.

transaction manager
A software unit that coordinates the activities of resource managers by managing global transactions and coordinating the decision to commit them or roll them back.

transaction set
The basic business document in ANSI X12 data. Transaction sets are enclosed in an envelope that separates one transaction set from another (ST-SE envelope). Groups of transaction sets that are functionally related are enclosed in a functional group envelope (GS-GE envelope). Transaction sets are made up of segments and loops.

transcoding technology
Content adaptation to meet the specific capabilities of a client device.

transform

1. To convert a document from one form to another, such as using a purchase order formatted as an XML document to create the same purchase order formatted as an EDI document. See also translate.
2. Programming logic that converts data from one format into another format.

transform algorithm
A procedure that is used to transform the message for web services security message processing, such as the C14N (canonicalization) transform that is used for XML digital signatures.

transformation
The process of changing data from one format or structure to another format or structure.

Transformation API for XML (TrAX)
A programming interface that can transform XML and related tree-shaped data structures.

transition
A connection between two tasks in a rule flow. Transitions are unidirectional, and they can have conditions attached to them.

transition condition

1. In a rule flow, a specification of a transition that dictates when the target task can be executed.
2. A Boolean expression that determines when processing control should be passed to the targeted node.

translate
In early versions of WebSphere Data Interchange, to convert a document from one form to another. See also transform.

translation object
A source map that has been compiled to provide instructions for translating from one format to another in a way that can be interpreted by the translator.

translation table
A user-defined table that is used to translate data values that differ between the source and target documents. For example, a manufacturer and supplier with different part numbers for the same item can use a translation table to convert their part numbers to the other company part numbers during translation.

translator
A component, typically the Data Interchange Services translator component, responsible for translating a document from one format to another.

Transmission Control Protocol (TCP)
A communication protocol used in the Internet and in any network that follows the Internet Engineering Task Force (IETF) standards for internetwork protocol. TCP provides a reliable host-to-host protocol in packet-switched communication networks and in interconnected systems of such networks. See also Internet Protocol.

Transmission Control Protocol/Internet Protocol (TCP/IP)
An industry-standard, nonproprietary set of communication protocols that provides reliable end-to-end connections between applications over interconnected networks of different types.

transmission type
The largest object in an EDI type tree. A transmission might include many interchanges from or to many trading partners.

transparent decision service
A reusable decision service that is easily accessible to all participants of its life cycle: developers, business analysts, and policy managers. It is easy to adapt and contributes to an enterprise compliance strategy.

transport
The request queue between a web servers plug-in and a web container in which the web modules of an application reside. When a user requests an application from a web browser, the request is passed to the web server, then along the transport to the web container.

transport adapter
An adapter (such as an HTTP Adapter) that is used with an encoding/decoding adapter to support various protocols (for example, SOAP) in a transport-independent way. The transport adapter is used to transport the data either from the source or to the destination.

Transportation Data Coordinating Committee (TDCC)
An organization that sets standards for the motor, rail, ocean, and air industries administered by EDIA. This is the original EDI organization for the United States, and through it, the original EDI Standards were developed, published, and maintained. It has now changed its name to EDIA, and has become the national EDI user group for the United States.

transport chain
A representation of a network protocol stack that is operating within an application server.
transport channel chain
A specification of the transport channels that are used by a server for receiving information. Transport channel chains contain end points
transporting
A method of conveying data using a specified adapter following either an encode or decode command.
Transport Layer Security (TLS)
An Internet Engineering Task Force (IETF)-defined security protocol that is based on Secure Sockets Layer (SSL) and is specified in RFC 2246.
TrAX
See Transformation API for XML.
tree
A data structure whose elements are linked in a hierarchical fashion.
trend analysis
A type of analysis that displays the analysis of the changes in a given item of information over a period of time.
trigger

1. In database technology, a program that is automatically called whenever a specified action is performed on a specific table or view.
2. A mechanism that detects an occurrence and can cause additional processing in response.

triple Data Encryption Standard (triple DES)
A block cipher algorithm that can be used to encrypt data transmitted between managed systems and the management server. Triple DES is a security enhancement of DES that employs three successive DES block operations.
triple DES
See triple Data Encryption Standard.
TRUE
See task-related user exit.
trunk
In the CVS team development environment, the main stream of development, also referred to as the HEAD stream.
trust anchor
A trusted keystore file that contains a trusted certificate or a trusted root certificate that is used to assert the trust of a certificate.
trust association
An integrated configuration between the security server of the product and third-party security servers. A reverse proxy server acts as a front-end authentication server, while the product applies its own authorization policy onto the resulting credentials passed by the proxy server.
trust association interceptor (TAI)
The mechanism by which trust is validated in the product environment for every request received by the proxy server. The method of validation is agreed upon by the proxy server and the interceptor.
trusted identity evaluator
A mechanism that is used by a server to determine whether to trust a user identity during identity assertion.
trusted root
A certificate signed by a trusted certificate authority (CA).
trust file
A file that contains signer certificates.
trust policy
A trusted list of certificates that are used to control the trust and validity period of certificates. It enables the trust of certificates issued by a certificate authority to be limited.
trust relationship
An established and trusted communication path through which a computer in one domain can communicate with a computer in the other domain. Users in a trusted domain can access resources in the trusting domain.
truststore
In security, a storage object, either a file or a hardware cryptographic card, where public keys are stored in the form of trusted certificates, for authentication purposes in web transactions. In some applications, these trusted certificates are moved into the application keystore to be stored with the private keys. See also keystore.
truststore file
A key database file that contains the public keys for a trusted entity.
TSO
See Time Sharing Option.
TTL
See time to live.
tuple
See row.
type

1. In a WSDL document, an element that contains data type definitions using some type system (such as XSD).
2. The definition of a data object or set of data objects that is graphically represented in a type tree in the Type Designer.
3. In Java programming, a class or interface.

type checking
The action of checking the validity of business items against a business item template during process simulation or deployment. Type checking is available only with decision gateways.
type hierarchy
The complete context for a Java class or interface including its superclasses and subclasses.
type tree
In the Type Designer, the graphical representation of the definition and organization of data objects.

U

UCA
: See undercover agent.

UCS
: 1. See Uniform Communication Standard.
2. See universal character set.

UDDI
: See Universal Description, Discovery, and Integration.

UDDI Business Registry
: A collection of peer directories that contain information about businesses and services.

UDDI node
: A set of web services that supports at least one of the Universal Description, Discovery, and Integration (UDDI) APIs. A UDDI node consists of one or more instances of a UDDI application running in an application server or a cluster of application servers with an instance of the UDDI database.

UDDI node initialization
: The process by which values are set in the Universal Description, Discovery, and Integration (UDDI) database and the behavior of the UDDI node is established.

UDDI node state
: A description of the current status of the Universal Description, Discovery, and Integration (UDDI) node.

UDDI policy
: A statement of the required and expected behavior of a Universal Description, Discovery, and Integration (UDDI) registry that is specified through policy values that are defined in the UDDI specification.

UDDI property
: A characteristic or attribute that controls the behavior of a Universal Description, Discovery, and Integration (UDDI) node.

UDDI registry
: A distributed registry of businesses and their service descriptions that adheres to the Universal Description, Discovery, and Integration (UDDI) standard for managing the discovery of web services. UDDI registries come in two forms, public and private, both of which are implemented in a common XML format.

UDF
: See user-defined function.

UML
: See Unified Modeling Language.

unary operator
: An operator that changes the sign of a numeric value.

unaugment
: To remove the last template that was augmented to a profile. A profile must be unaugmented before it is deleted. See also augment.

unbound set
: The set of all possible types of data that might be listed last in a group.

uncontrolled flow
: A flow that proceeds without dependencies or conditional expressions. Typically, an uncontrolled flow is a sequence flow between two activities that do not have a conditional indicator (mini-diamond) or an intervening gateway.

undelivered message queue
: See dead-letter queue.

undercover agent (UCA)
: An agent that is attached to a message event in a business process definition (BPD) and that calls a service to handle the event. For example, when a message event is received from an external system, a UCA is needed to invoke the appropriate service in response to the message.

UN/EDIFACT
: See United Nations Electronic Data Interchange for Administration, Commerce and Transport.

Unicode
: A character encoding standard that supports the interchange, processing, and display of text that is written in the common languages around the world, plus many classical and historical texts.

Unified Modeling Language (UML)
: A standard notation for the modeling of real-world objects as a first step in developing an object-oriented design methodology.

Uniform Communication Standard (UCS)
: The EDI standard used in the grocery industry.

Uniform Resource Identifier (URI)
: 1. A unique address that is used to identify content on the web, such as a page of text, a video or sound clip, a still or animated image, or a program. The most common form of URI is the web page address, which is a particular form or subset of URI called a Uniform Resource Locator (URL). A URI typically describes how to access the resource, the computer that contains the resource, and the name of the resource (a file name) on the computer. See also Uniform Resource Name.
2. A compact string of characters for identifying an abstract or physical resource.

Uniform Resource Indicator (URI)
: A unique address that is used to identify content on the web, such as a page of text, a video or sound clip, a still or animated image, or a program. The most common form of URI is the web page address, which is a particular form or subset of URI called a Uniform Resource Locator (URL). A URI typically describes how to access the resource, the computer that contains the resource, and the name of the resource (a file name) on the computer.

Uniform Resource Locator (URL)
: The unique address of an information resource that is accessible in a network such as the Internet. The URL includes the abbreviated name of the protocol used to access the information resource and the information used by the protocol to locate the information resource.

Uniform Resource Name (URN)
: A name that uniquely identifies a web service to a client. See also Uniform Resource Identifier.

United Nations Electronic Data Interchange for Administration, Commerce and Transport (UN/EDIFACT)
: An international set of electronic data interchange (EDI) standards published by the United Nations that is built upon X12 and TDI (Trade Data Interchange) standards.

United Nations Standard Products and Services Classification (UNSPSC)

An open global standard for classifying products and services based on common function, purpose, and task.

United Nations Trade Data Interchange (UNTDI)
A standard that preceded the UN/EDIFACT EDI standard.

universal character set (UCS)
The ISO standard that allows all data to be represented as 2 bytes (UCS-2) or 4 bytes (UCS-4). Encoding in the UCS-2 form can accommodate the necessary characters for most of the written languages in the world.

Universal Description, Discovery, and Integration (UDDI)
A set of standards-based specifications that enables companies and applications to quickly and easily find and use web services over the Internet. See also web service.

universal integration hub
A unified page presentation architecture that enables site designers to create web portal pages by using various components, including HTML and web content, feeds, portlets, iWidgets, and elements that are derived from frameworks such as Adobe Flex.

Universally Unique Identifier (UUID)
The 128-bit numeric identifier that is used to ensure that two components do not have the same identifier.

UNIX System Services
An element of z/OS that creates a UNIX environment that conforms to XPG4 UNIX 1995 specifications and that provides two open-system interfaces on the z/OS operating system: an application programming interface (API) and an interactive shell interface.

unmanaged node
A node that is defined in the cell topology that does not have a node agent that manages the process. An unmanaged node is typically used to manage web servers.

unmanaged web application
A web application with a life cycle that is managed outside of the administrative domain. By creating a representation of these applications that are deployed through external tools, the on demand router can prioritize and route HTTP requests to the application.

unmodeled fault
A fault message that is returned from a service that has not been modeled on the Web Services Description Language (WSDL) port type.

unrealized
Pertains to a web diagram node that is not yet associated with an actual resource. See also realize.

unrecognized screen
In the 3270 terminal service development tools, a screen that cannot be identified by any of the recognition profiles currently defined.

UNSPSC
See United Nations Standard Products and Services Classification.

UNTDI
See United Nations Trade Data Interchange.

upgradeable lock
A lock that identifies the intent to update a cache entry when using a pessimistic lock.

upstream
Pertaining to the direction of the flow, which is from the start of the process (upstream) toward the end of the process (downstream).

URI
1. See Uniform Resource Identifier.
2. See Uniform Resource Indicator.

URL
See Uniform Resource Locator.

URL scheme
A format that contains another object reference.

URN
See Uniform Resource Name.

use case
The specification of a sequence of actions that a system can perform, interacting with users of the system. Use cases are used in system analysis to identify system requirements. See also scenario.

user account
The login directory and other information that gives a user access to the system.

user-defined function (UDF)
A function that is defined to the DB2 database system by using the CREATE FUNCTION statement and that can be referenced thereafter in SQL statements. See also function.

user exit profile
A profile that defines a user-provided program or exit routine to Data Interchange Services.

user group
A group consisting of one or more defined individual users, identified by a single group name.

user name token
A type of token that is represented by a user name and optionally, by a password.

user registry
A database of known users and user-provided information that is used for authentication purposes.

UTC
See Coordinated Universal Time.

UTF-8
Unicode Transformation Format, 8-bit encoding form, which is designed for ease of use with existing ASCII-based systems. The CCSID value for data in UTF-8 format is 1208.

UUID
See Universally Unique Identifier.

V

validation

The checking of data or code for correctness or for compliance with applicable standards, rules, and conventions.

validation action
A mechanism for verifying whether the actual value of a variable at run time corresponds to the expected value of that variable. See also timing constraint.

validation map
A set of mapping instructions that describe additional validation for an EDI document. One of five supported map types.

validator
A program that checks data or code for correctness or for compliance with applicable standards, rules, and conventions.

variable

1. Data that passes from one step to another in a process. For example, a process that automates escalation of customer issues needs variables to hold information, such as the customer's name and the issue ID.
2. A representation of a changeable value. See also global variable.

variable component name
A component of a group type that includes the literal at the end of the name because it represents more than one type. The literal ANY acts like a wild card, which represents any type whose name could appear in that place.

variant action
An action that is derived from another action so that the content of the action can vary. A field in the variant action object can derive its value in a different way from the way that the same field derives its value in the base action object.

verb
See people assignment criterion.

verbalization
The process of associating terms and phrases to elements of the business object model (BOM). See also constant.

version
A separately licensed program that typically has significant new code or new function.

version control
The coordination and integration of the history of work submitted by a team. See also team support.

vertical scaling
Setting up multiple application servers on one machine, typically by creating cluster members.

vertical stacking
The process of starting more than one instance of the dynamic cluster on a node to manage bottlenecks.

view

1. A logical table that is based on data stored in an underlying set of tables. The data returned by a view is determined by a SELECT statement that is run on the underlying tables.
2. A reusable user interface that is used for a business object or human service. A view consists of one or more other views, data bindings, layout instructions, and behavior.
3. In Eclipse-based user interfaces, a pane that is outside the editor area, which can be used to look at or work with the resources in the workbench.

view synchronous high-availability manager group
A special class of high availability (HA) group that can be created and used by components that require a certain virtual synchrony (VS) quality of service (QoS) for group communication.

VIPA
See virtual IP address.

virtual application layer
A group of components in a virtual application pattern that facilitate complex virtual application design. A virtual application layer enables virtual application patterns to be reused in different contexts; one virtual application pattern is used as a reference layer in another virtual application pattern.

virtual application pattern plug-in
The resources and automation that provide the specific capabilities for a virtual application component. See also virtual application pattern type.

virtual application pattern type
A set of virtual application pattern plug-ins for a specific type of application or application capability. For example, the IBM Web Application Pattern pattern type provides the components, links, policies, and automation that are required to deploy web applications. See also virtual application pattern plug-in.

virtual host
A configuration that enables one host to resemble multiple logical hosts. Each virtual host has a logical name and a list of one or more DNS aliases by which it is known.

virtual image
The operating system and product binary files that are required to create a virtual system pattern.

virtual IP address (VIPA)
An IP address that is shared among multiple domain names or multiple servers. Virtual IP addressing enables one IP address to be used either when insufficient IP addresses are available or as a means to balance traffic to multiple servers.

virtualization
A technique that encapsulates the characteristics of resources from the way in which other systems interact with those resources.

virtual local area network (VLAN)
A logical association of switch ports based upon a set of rules or criteria, such as Medium Access Control (MAC) addresses, protocols, network address, or multicast address. This concept permits the LAN to be segmented again without requiring physical rearrangement.

virtual machine
An abstract specification for a computing device that can be implemented in different ways in software and hardware.

virtual private network (VPN)
An extension of a company intranet over the existing framework of either a public or private network. A VPN ensures that the data that is sent between the two endpoints of its connection remains secure.

virtual synchrony (VS)
A property of group communication that guarantees how messages are delivered when the view changes, for example, when existing members fail or new members join.

virtual system instance
The virtual environment that runs on a hypervisor in the cloud.

virtual system pattern
: One or more virtual images, which can include script packages, that implement a deployment topology. A virtual system pattern is a shared topology definition used for repeatable deployment.

visibility
: In a user interface, the property of a control that declares whether the control is to be displayed or not displayed during run time.

visibility service
: A type of business service that monitors and displays the performance, behavior, or metrics of a business process.

visible widget
: A fully functional widget that displays business data. A visible widget has a solid frame.

visualization
: An association between a Scalable Vector Graphics (SVG) diagram and the set of actions that describe how the diagram should be updated based on the values of metrics or key performance indicators (KPIs).

visual snippet
: A diagrammatic representation of a fragment of Java programming language that can be manipulated with the visual snippet editor.

VLAN
: See virtual local area network.

vocabulary
: 1. The set of terms and phrases that are used for rule editing.
  2. A repository for storing reusable business elements, such as terms, business item definitions, roles, messages, and errors, that are used in a business process.

VPN
: See virtual private network.

VS
: See virtual synchrony.

W

W3C
: See World Wide Web Consortium.

waiter
: A thread waiting for a connection.

WAP
: See Wireless Application Protocol.

WAR
: See web archive.

WAR file
: See web archive.

watch
: A map, including the set of events that initiate it, as defined from the Integration Flow Designer.

watchpoint
: A breakpoint that suspends execution when a specified field or expression is modified.

WBMP
: See wireless bitmap.

WCCM
: See WebSphere Common Configuration Model.

web analytics page overlay
: Web page and channel delivery analysis that is rendered in place on the website.

web application
: An application that is accessible by a web browser and that provides some function beyond static display of information, for instance by allowing the user to query a database. Common components of a web application include HTML pages, JSP pages, and servlets.

web application bridge
: A virtual web application that passes request data, including selected HTTP headers, cookies, and POST data, to the content provider. The web application bridge sends the response data back to the requester, including selected HTTP headers, cookies, and POST data. See also bridge.

web archive (WAR)
: A compressed file format, defined by the Java EE standard, for storing all the resources required to install and run a web application in a single file. See also enterprise archive, Java archive.

web component
: A servlet, JavaServer Pages (JSP) file, or a HyperText Markup Language (HTML) file. One or more web components make up a web module.

web container
: A container that implements the web component contract of the Java EE architecture. (Sun)

web container channel
: A type of channel within a transport chain that creates a bridge in the transport chain between an HTTP inbound channel and a servlet or JavaServer Pages (JSP) engine.

web crawler
: A crawler that explores the web by retrieving a web document and following the links within that document.

web diagram
: A Struts file that uses icons and other images on a free-form surface to help application developers visualize the flow structure of a Struts-based web application.

web module
: A unit that consists of one or more web components and a web deployment descriptor. (Sun)

Web Ontology Language (OWL)
: A language that is used to explicitly represent the meaning of terms in vocabularies and the relationships between those terms. OWL is intended to be used when the information contained in documents needs to be processed by applications, as opposed to situations where the content only needs to be

presented to humans. See also ontology.

web portal
: See portal.

web project
: A container for other resources such as source files and metadata that corresponds to the Java EE-defined container structure and hierarchy of files necessary for web applications to be deployed.

web property extension (WPX)
: IBM extension to the standard deployment descriptors for web applications. These extensions include Multipurpose Internet Mail Extensions (MIME) filtering and servlet caching.

web resource
: Any one of the resources that are created during the development of a web application for example web projects, HTML pages, JavaServer Pages (JSP) files, servlets, custom tag libraries, and archive files.

web resource collection
: A list of URL patterns and HTTP methods that describe a set of resources to be protected. (Sun)

web server
: A software program that is capable of servicing Hypertext Transfer Protocol (HTTP) requests.

web server plug-in
: A software module that supports the web server in communicating requests for dynamic content, such as servlets, to the application server.

web server separation
: A topology where the web server is physically separated from the application server.

web service
: 1. An application that performs specific tasks and is accessible through open protocols such as HTTP and SOAP.
  2. A self-contained, self-describing modular application that can be published, discovered, and invoked over a network using standard network protocols. Typically, XML is used to tag the data, SOAP is used to transfer the data, WSDL is used for describing the services available, and UDDI is used for listing what services are available. See also SOAP, Universal Description, Discovery, and Integration, Web Services Description Language.

web service endpoint
: An entity that is the destination for web service messages. A web service endpoint has a Uniform Resource Identifier (URI) address and is described by a Web Service Definition Language (WSDL) port element.

web service interface
: A group of operations described by the content of a Web Service Definition Language (WSDL) 1.1 port element. These operations can provide access to resource properties and metadata. (OASIS)

Web Services Business Process Execution Language (WS-BPEL)
: See Business Process Execution Language.

Web Services Description Language (WSDL)
: An XML-based specification for describing networked services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. See also web service.

Web Services Interoperability (WS-I)
: An open industry organization that is chartered to promote web services interoperability across platforms, operating systems, and programming languages.

Web Services Interoperability Organization (WSI)
: An open industry organization that promotes web services interoperability across platforms, operating systems, and programming languages.

Web Services Invocation Framework (WSIF)
: A Java API that supports dynamic invoking of web services, regardless of the format in which the service is implemented or the access mechanism.

Web Services Invocation Language (WSIL)
: An XML document format that facilitates the discovery of existing web services and provides a set of rules for how inspection-related information should be made available for consumption.

Web Services Policy Framework (WS-Policy)
: A model and framework for describing the capabilities, requirements, and general characteristics of a web service as a policy assertion or a collection of policy assertions.

Web Services Security (WS-Security)
: A flexible standard that is used to secure web services at the message level within multiple security models. SOAP messages can be secured through XML digital signature, confidentiality can be secured through XML encryption, and credential propagation can be secured through security tokens.

website
: A related collection of files available on the web that is managed by a single entity (an organization or an individual) and contains information in hypertext for its users. A website often includes hypertext links to other websites.

WebSphere BI for FN message
: A WebSphere MQ message that has a folder labeled ComIbmDni in the MQRFH2 header. This folder provides the data that is required by WebSphere BI for FN to process the message.

WebSphere Common Configuration Model (WCCM)
: A model that provides for programmatic access to configuration data.

weight
: See point.

weighted sum
: The result of multiplying the weight by the score assigned to an attribute. The weight is a percentage. An attribute with a higher weight is more significant than other attributes as it contributes a bigger share to the final score.

what you see is what you get (WYSIWYG)
: A capability of an editor to continually display pages exactly as they will be printed or otherwise rendered.

while loop
: A loop that repeats the same sequence of activities as long as some condition is satisfied. The while loop tests its condition at the beginning of every loop. If the condition is false from the start, the sequence of activities contained in the loop never runs.

widget
: A portable, reusable application or piece of dynamic content that can be placed into a web page, receive input, and communicate with an application or with another widget.

wiki

A collaborative website that anyone can edit by using a web browser.

wire

1. A connector used to pass control and data from a component or an export to a target.
2. To connect two or more components or cooperative portlets so that they work together. In an application, wiring identifies target services; for portlets, changes in the source portlet automatically update the target portlets.

Wireless Application Protocol (WAP)
: An open industry standard for mobile Internet access that allows mobile users with wireless devices to easily and instantly access and interact with information and services.

wireless bitmap (WBMP)
: A graphic format that is optimized for mobile computing devices. WBMP is part of the Wireless Application Protocol, Wireless Application Environment Specification.

Wireless Markup Language (WML)
: A markup language based on XML that is used to present content and user interfaces for wireless devices such as cellular phones, pagers, and personal digital assistants.

wizard
: An active form of help that guides users through each step of a particular task.

WLM
: See Workload Manager.

WML
: See Wireless Markup Language.

work basket
: A location where work waits for action by a user. This action can be taken either directly on the work in the work basket, or the work can be transferred to another work basket so that actions can be taken there.

workbench
: The user interface and integrated development environment (IDE) in Eclipse and Eclipse-based tools such as IBM Rational Application Developer.

work class
: A mechanism for grouping specific work together that must be associated with a common service policy or routing policy. Work classes group Uniform Resource Identifiers (URIs) or web services from an application.

workflow
: The sequence of activities performed in accordance with the business processes of an enterprise.

working data set
: A data set containing customized JCL that is used to configure and run an execution environment in Decision Server for z/OS.

working directory
: The active directory. When a file name is specified without a directory, the current directory is searched.

working memory
: A part of the rule engine that contains the current state of objects. It is this current state that determines which rules are added to the agenda, and in which order these rules are executed.

work item

1. In the human task editor, the representation of a task. Staff members can browse all work items that they have the authority to claim.
2. See also inline task, stand-alone task.

workload management
: The optimization of the distribution of incoming work requests to the application servers, enterprise beans, servlets and other objects that can effectively process the request.

Workload Manager (WLM)
: A component of z/OS that provides the ability to run multiple workloads at the same time within one z/OS image or across multiple images.

work manager
: A thread pool for Java Platform, Enterprise Edition (Java EE) applications.

work object
: A type of asynchronous bean that applications implement to run code blocks asynchronously.

workspace

1. A directory on disk that contains all project files, as well as information such as preferences.
2. In Eclipse, the collection of projects and other resources that the user is currently developing in the workbench. Metadata about these resources resides in a directory on the file system; the resources might reside in the same directory.
3. A temporary repository of configuration information that administrative clients use.

World Wide Web Consortium (W3C)
: An international industry consortium set up to develop common protocols to promote evolution and interoperability of the World Wide Web.

WPX
: See web property extension.

wrapper

1. An alternate and supported interface that hides unsupported data types required by a server object behind a thin intermediate server object.
2. An object that encapsulates and delegates to another object to alter its interface or behavior in some way. (Sun)

wrapper business object
: A top-level business object that groups child business objects for a component to use in a single operation or contains processing information about its child business object.

write-behind cache
: A cache that asynchronously writes each write operation to the database using a loader.

write-through cache
: A cache that synchronously writes each write operation to the database using a loader.

WS-BPEL

See Web Services Business Process Execution Language.

WSDL
See Web Services Description Language.

WSDL document
A file that provides a set of definitions that describe a web service in Web Services Description Language (WSDL) format.

WSDL file
See WSDL document.

WS-I
See Web Services Interoperability.

WSI
See Web Services Interoperability Organization.

WSIF
See Web Services Invocation Framework.

WSIL
See Web Services Invocation Language.

WS-Policy
See Web Services Policy Framework.

WS-Security
See Web Services Security.

WYSIWYG
See what you see is what you get.

X

X.25
A CCITT standard that defines an interface to packet-switched communication services.

X.500
The directory services standard of ITU, ISO, and IEC.

X.509 certificate
A certificate that contains information that is defined by the X.509 standard.

X12
A protocol from the American National Standards Institute (ANSI) for electronic data interchange (EDI).

XA
A bidirectional interface between one or more resource managers that provide access to shared resources and a transaction manager that monitors and resolves transactions.

XACML
See Extensible Access Control Markup Language.

Xalan processor
An XSLT processor that is part of the Apache project. See also XSL Transformation.

XDoclet
An open, source code generation engine that uses special JavaDoc tags to parse Java source files and generate output such as XML descriptors or source code, based on templates.

X field
The primary data field in a chart. In a line chart, typically the X field appears along the horizontal axis. For example, an X field can represent cost data for the elements that appear along the horizontal axis of the chart.

XHTML
See Extensible Hypertext Markup Language.

XID
See transaction identifier.

xJCL
An XML-based job control language that is used to define a batch job. See also job control language.

XML
See Extensible Markup Language.

XML catalog
A catalog that contains rules specifying how an XML processor should resolve references to entities. Use of a catalog eliminates the need to change URIs within XML documents as resources are moved during development.

XML digital signature
A specification that defines the XML syntax and the processing rules to sign and verify the digital signatures for the digital content.

XML document definition
A reference to either an XML DTD document definition or an XML schema document definition.

XML encryption
A specification that defines how to encrypt the content of an XML element.

XML parser
A program that reads XML documents and provides an application with access to their content and structure.

XML Path Language (XPath)
A language that is designed to uniquely identify or address parts of source XML data, for use with XML-related technologies, such as XSLT, XQuery, and XML parsers. XPath is a World Wide Web Consortium standard.

XML schema
A mechanism for describing and constraining the content of XML files by indicating which elements are allowed and in which combinations. XML schemas are an alternative to document type definitions (DTDs) and can be used to extend functionality in the areas of data typing, inheritance, and presentation.

XML Schema Definition Language (XSD, XSD, XSDL, XSDL)
A language for describing XML files that contain XML schema.

XML Schema Infoset Model (XSD)
A library that provides an API for manipulating the components of an XML Schema, as described by the W3C XML Schema specifications.

XML token

A security token that is in an XML format, such as a Security Assertion Markup Language (SAML) token.

XOM
: See execution object model.

X/Open XA
: The X/Open Distributed Transaction Processing XA interface. A proposed standard for distributed transaction communication. The standard specifies a bidirectional interface between resource managers that provide access to shared resources within transactions, and between a transaction service that monitors and resolves transactions.

XPath
: See XML Path Language.

XPath expression
: An expression that searches through an XML document and extracts information from the nodes (any part of the document, such as an element or attribute) in that document.

XSD

1. See XML Schema Infoset Model.
2. See XML Schema Definition Language.

XSD, XSDL
: See XML Schema Definition Language.

XSDL
: See XML Schema Definition Language.

XSL
: See Extensible Stylesheet Language.

XSL style sheet
: Code that describes how an XML document should be rendered (displayed or printed).

XSLT

1. See XSL Transformation.
2. See Extensible Stylesheet Language Transformation.

XSLT function
: Function that is defined by the XSL Transform (XSLT) specification for the manipulation of numbers, strings, Boolean values, and node-sets.

XSL Transformation (XSLT)
: A standard that uses XSL style sheets to transform XML documents into other XML documents, fragments, or HTML documents. See also Xalan processor.

XU
: See execution unit.

Y

Y field
: A secondary data field in a chart. In a line chart, typically the Y fields appear along the vertical axis. For example, an Y field can represent resources whose costs are represented along the vertical axis of the chart.

Z

zone

1. A function that enables rules-based shard placement to improve grid availability by placing shards across different data centers, whether on different floors or even in different buildings or geographies.
2. A logical section within an area. A zone can overlap areas but belongs only to the area where it was created. Zones are the units on which rules can be defined and run.

z/OS
: An IBM mainframe operating system that uses 64-bit real storage.

        - Maps
        - Clients
      - IBM eXtremeMemory
      - Zones
      - Evictors
      - OSGi framework overview
    - Cache integration overview
      - Liberty profile
        - Data caching and the Liberty profile
      - JPA level 2 (L2) cache plug-in
      - HTTP session management
      - Dynamic cache provider overview
    - Database integration overview
      - Sparse and complete cache
      - Side cache
      - In-line cache
      - Write-behind caching
      - Loaders
      - Data preloading and warm-up
      - Database synchronization`
      - Data invalidation
      - Indexing
      - JPA Loaders
    - Serialization overview
      - Serialization using Java
      - ObjectTransformer plug-in
      - Serialization using the DataSerializer plug-ins
    - Scalability overview
      - Data grids, partitions, and shards
      - Partitioning
      - Placement and partitions
      - Single-partition and cross-data-grid transactions
      - Scaling in units or pods
    - Availability overview
      - High availability
        - Core groups
        - High availability catalog service
        - Catalog server quorums
        - Replication for availability
      - Replicas and shards
        - Shard placement
        - Reading from replicas
        - Load balancing across replicas
        - Shard lifecycles
        - Map sets for replication
    - Transaction processing overview
      - Transactions
        - Transaction processing in Java EE applications
      - CopyMode attribute
      - Locking strategies
      - Lock types
      - Deadlocks
      - Data access and transactions
      - Transaction isolation
      - Single-partition and cross-data-grid transactions
      - JMS for distributed transaction changes
    - Security overview
    - REST data services overview
- Scenarios
    - Using an OSGi environment to develop and run eXtreme Scale plug-ins
      - OSGi framework overview
      - Installing the Eclipse Equinox OSGi framework with Eclipse Gemini for clients and servers
        - Installing eXtreme Scale bundles
      - Running eXtreme Scale containers with non-dynamic plug-ins in an OSGi environment
      - Administering eXtreme Scale servers and applications in an OSGi environment
      - Building and running eXtreme Scale dynamic plug-ins for use in an OSGi environment
        - Building eXtreme Scale dynamic plug-ins
        - Configuring eXtreme Scale plug-ins with OSGi Blueprint
        - Installing and starting OSGi-enabled plug-ins
      - Running eXtreme Scale containers with dynamic plug-ins in an OSGi environment
        - Configuring OSGi-enabled plug-ins using the ObjectGrid descriptor XML file
        - Starting eXtreme Scale servers using the Eclipse Equinox OSGi framework
        - Administering OSGi-enabled services using the xscmd utility
        - Configuring servers with OSGi Blueprint
    - Using JCA to connect transactional applications to eXtreme Scale clients

        - Transaction processing in Java EE applications
        - Installing an eXtreme Scale resource adapter
        - Configuring eXtreme Scale connection factories
        - Configuring Eclipse environments to use eXtreme Scale connection factories
        - Configuring applications to connect with eXtreme Scale
        - Securing J2C client connections
        - Developing eXtreme Scale client components to use transactions
        - Administering J2C client connections
    - Configuring HTTP session failover in the Liberty profile
        - Enabling the eXtreme Scale web feature in the Liberty profile
        - Configuring a web server plug-in to forward requests to multiple servers in the Liberty profile
        - Merging plug-in configuration files for deployment to the application server plug-in
    - Running grid servers in the Liberty profile using Eclipse tools
        - Installing the Liberty profile developer tools for WebSphere eXtreme Scale
        - Setting up your development environment within Eclipse
            - Configuring eXtreme Scale in the Liberty profile using Eclipse tools
            - Creating an OSGi bundle project for eXtreme Scale data grid development
    - Migrating a WebSphere Application Server memory-to-memory replication or database session to use WebSphere eXtreme Scale session management
- Samples
    - Free trial
    - Sample properties files
    - Sample: xsadmin utility
        - Creating a configuration profile for the xsadmin utility
        - xsadmin utility reference
        - Verbose option for the xsadmin utility
- Tutorials
    - Tutorial: Querying a local in-memory data grid
        - ObjectQuery tutorial - step 1
        - ObjectQuery tutorial - step 2
        - ObjectQuery tutorial - step 3
        - ObjectQuery tutorial - step 4
    - Tutorial: Storing order information in entities
        - Step 1
        - Step 2
        - Step 3
        - Step 4
        - Step 5
        - Step 6
    - Tutorial: Configuring Java SE security
        - Java SE security tutorial - Step 1
        - Java SE security tutorial - Step 2
        - Java SE security tutorial - Step 3
        - Java SE security tutorial - Step 4
        - Java SE security tutorial - Step 5
        - Java SE security tutorial - Step 6
    - Tutorial: Run eXtreme Scale clients and servers in the Liberty profile
        - Liberty profile
        - Module 1: Install the Liberty profile
        - Module 2: Create a web application server in the Liberty profile
            - Lesson 2.1: Define a server to run in the Liberty profile
        - Module 3: Add the Liberty web feature to the Liberty profile
            - Lesson 3.1: Define a web application to run in the Liberty profile
        - Module 4: Configure clients to use client APIs in the Liberty profile
            - Lesson 4.1: Configure the Liberty profile to run with eXtreme Scale clients
        - Module 5: Run the data grid inside the Liberty profile
            - Lesson 5.1: Configure eXtreme Scale servers to use the Liberty profile
            - Lesson 5.2: Configuring a Liberty profile web application server to use eXtreme Scale for session replication
    - Tutorial: Integrate WebSphere eXtreme Scale security with WebSphere Application Server
        - Introduction
        - Module 1: Prepare WebSphere Application Server
            - Lesson 1.1: Understand the topology and get the tutorial files
            - Lesson 1.2: Configure the WebSphere Application Server environment
        - Module 2: Configure WebSphere eXtreme Scale to use WebSphere Application Server Authentication plug-ins
            - Lesson 2.1: Configure client server security
            - Lesson 2.2: Configure catalog server security
            - Lesson 2.3: Configure container server security
            - Lesson 2.4: Install and run the sample
        - Module 3: Configure transport security
            - Lesson 3.1: Configure CSIv2 inbound and outbound transport
            - Lesson 3.2: Add SSL properties to the catalog server properties file
            - Lesson 3.3: Run the sample
        - Module 4: Use Java Authentication and Authorization Service (JAAS) authorization in WebSphere Application Server
            - Lesson 4.1: Enable WebSphere eXtreme Scale authorization
            - Lesson 4.2: Enable user-based authorization

    - Planning for installation
        - Hardware and software requirements
        - Java SE considerations
        - Java EE considerations
        - Directory conventions
    - Planning environment capacity
        - Sizing memory and partition count calculation
        - Sizing CPU per partition for transactions
        - Sizing CPUs for parallel transactions
        - Dynamic cache capacity planning
    - Planning to develop WebSphere eXtreme Scale applications
        - Planning to develop Java applications
            - Java API overview
            - Java plug-ins overview
            - REST data services overview
            - Spring framework overview
            - Java class loader and classpath considerations
            - Relationship management
            - Cache key considerations
            - Data for different time zones
- Installing
    - Installation overview
    - Planning for installation
        - Installation topologies
        - Hardware and software requirements
        - Offering IDs for WebSphere eXtreme Scale product offerings
        - Java SE considerations
        - Java EE considerations
        - Directory conventions
        - Runtime files for WebSphere Application Server installation
        - Runtime files for installation
    - IBM Installation Manager and WebSphere eXtreme Scale product offerings
        - Installing IBM Installation Manager using the GUI
            - Installing the product with the GUI
        - Installing IBM Installation Manager using the command line
            - Installing the product using the command line
        - Installing IBM Installation Manager using response files
            - Installing the product using a response file
                - Creating a keyring
    - Installing the Eclipse Equinox OSGi framework with Eclipse Gemini for clients and servers
    - Installing the REST data service
        - Installing eXtreme Scale bundles
    - Installing within WebSphere Application Server
    - Installing the Liberty profile
        - Installing the Liberty profile application-serving environment by running a JAR file
    - Uninstalling the product using IBM Installation Manager
        - Uninstalling the product using the GUI
        - Uninstalling the product using the command line
        - Uninstalling the product using response files
    - Customizing WebSphere eXtreme Scale for z/OS
        - Installing the WebSphere Customization Toolbox
        - Generating customization definitions
        - Uploading and running customized jobs
    - Creating and augmenting profiles
        - Using the graphical user interface to create profiles
        - Using the graphical user interface to augment profiles
        - manageprofiles command
        - Non-root profiles
    - First steps after installation
    - Troubleshooting the product installation
- Updating and migrating
    - Updating eXtreme Scale servers
    - Migrating to WebSphere eXtreme ScaleVersion 8.5
    - Installing fix packs using IBM Installation Manager
        - Installing fix packs using the GUI
        - Installing fix packs using the command line
        - Installing fix packs using a response file
    - Uninstalling fix packs using IBM Installation Manager
        - Uninstalling fix packs using the GUI
        - Uninstalling fix packs using the command line
        - Uninstalling fix packs using response files
    - Updating WebSphere eXtreme Scale on WebSphere Application Server
    - xsadmin tool to xscmd tool migration
    - Deprecated properties and APIs
        - Removed properties and APIs

- Configuring
  - Configuration methods
  - Operational checklist
  - Configuring data grids
    - Configuring local deployments
    - Configuring evictors with XML files
    - Configuring a locking strategy in the ObjectGrid descriptor XML file
    - Configuring the lock timeout value
    - Configuring peer-to-peer replication with JMS
      - Distributing changes between peer JVMs
      - JMS event listener
    - Configuring dynamic maps
  - Configuring deployment policies
    - Configuring distributed deployments
    - Controlling shard placement with zones
      - Zones for replica placement
      - Zone-preferred routing
      - Defining zones for container servers
      - Example: Zone definitions in the deployment policy descriptor XML file
      - Viewing zone information with the xscmd utility
  - Configuring catalog and container servers
    - Configuring catalog servers and catalog service domains
      - Example: Configuring catalog service domains
      - Configuring WebSphere eXtreme Scale with WebSphere Application Server
        - Configuring the catalog service in WebSphere Application Server
          - Creating catalog service domains in WebSphere Application Server
            - Catalog service domain administrative tasks
            - Catalog service domain collection
            - Catalog service domain settings
            - Client security properties
            - Catalog service domain custom properties
      - Configuring the quorum mechanism
      - Tuning failover detection
    - Configuring container servers
      - Container server reconnect properties
      - Configuring container servers in WebSphere Application Server
        - Configuring WebSphere Application Server applications to automatically start container servers
    - Configuring eXtreme Scale servers to run in the Liberty profile
  - Configuring multiple data center topologies
  - Configuring ports
    - Configuring ports in stand-alone mode
    - Configuring ports in a WebSphere Application Server environment
    - Servers with multiple network cards
  - Configuring transports
    - Configuring Object Request Brokers
      - Configuring the Object Request Broker in a WebSphere Application Server environment
      - Configuring the Object Request Broker with stand-alone WebSphere eXtreme Scale processes
      - Configuring a custom Object Request Broker
  - Configuring Java clients
    - Java client overrides
    - Configuring Java clients with an XML configuration
    - Configuring the REST gateway with an XML configuration
    - Configuring Java clients programmatically
    - Configuring the near cache
    - Configuring an evictor for the near cache
    - Configuring JMS-based client synchronization
  - Configuring eXtreme Scale connection factories
    - Configuring Eclipse environments to use eXtreme Scale connection factories
  - Configuring cache integration
    - Configuring HTTP session managers
      - Configuring the HTTP session manager with WebSphere Application Server
        - Configuring WebSphere Application Server HTTP session persistence to a data grid
          - eXtreme Scale session management settings
        - Splicing a session data grid application with the addObjectGridFilter script
        - Editing the splicer.properties file
      - Configuring HTTP session manager with WebSphere Portal
      - Configuring the HTTP session manager for various application servers
      - XML files for HTTP session manager configuration
      - Servlet context initialization parameters
      - splicer.properties file
    - Configuring the dynamic cache provider for WebSphere eXtreme Scale
    - JPA level 2 (L2) cache plug-in
      - JPA cache configuration properties for both OpenJPA and Hibernate Version 3.0
      - Configuring the OpenJPA cache plug-in
        - Example: OpenJPA ObjectGrid XML files