

IBM WebSphere eXtreme Scale Version 7.1

Administration Guide

June 15, 2011



This edition applies to version 7, release 1, of WebSphere eXtreme Scale and to all subsequent releases and modifications until otherwise indicated in new editions.

© **Copyright IBM Corporation 2009, 2011.**

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

Contents

About the *Administration Guide*. vii

Chapter 1. Running the getting started sample application 1

Directory conventions 6

Chapter 2. Capacity planning 9

Sizing memory and partition count calculation 9

Sizing CPU per partition for transactions 11

Sizing CPUs for parallel transactions 11

Dynamic cache capacity planning 12

Chapter 3. Installing and deploying WebSphere eXtreme Scale 17

Installing stand-alone WebSphere eXtreme Scale or WebSphere eXtreme Scale Client 18

Runtime files for WebSphere eXtreme Scale stand-alone installation 19

Running the getting started sample application 21

Installing WebSphere eXtreme Scale or WebSphere eXtreme Scale Client with WebSphere Application Server 26

Runtime files for WebSphere eXtreme Scale integrated with WebSphere Application Server 28

Using the Installation Factory plug-in to create and install customized packages 30

Creating and augmenting profiles for WebSphere eXtreme Scale 45

Installing WebSphere eXtreme Scale or WebSphere eXtreme Scale Client silently. 55

Installation parameters 56

Customizing WebSphere eXtreme Scale for z/OS 57

Installing the WebSphere Customization Tools. 58

Generating customization definitions 59

Uploading and running customized jobs. 60

Uninstalling WebSphere eXtreme Scale 61

Chapter 4. Customizing WebSphere eXtreme Scale for z/OS. 63

Installing the WebSphere Customization Tools. 63

Generating customization definitions 64

Uploading and running customized jobs. 65

Chapter 5. Upgrading and migrating WebSphere eXtreme Scale Version 7.1 . 67

Updating eXtreme Scale servers 67

Migrating to WebSphere eXtreme Scale Version 7.1 69

Using the Update Installer to install maintenance packages 70

Deprecated properties and APIs 71

Chapter 6. Planning the WebSphere eXtreme Scale environment. 73

Planning overview 73

Hardware and software requirements. 74

Java SE considerations. 75

Java EE considerations. 76

Directory conventions 77

Caching topology: In-memory and distributed caching. 78

Local in-memory cache 79

Peer-replicated local cache 80

Distributed cache 82

Embedded cache 84

Multi-master data grid replication topologies 85

Catalog service 95

High-availability catalog service 96

Catalog server quorums 98

Container servers, partitions, and shards 103

Capacity planning 105

Sizing memory and partition count calculation 105

Sizing CPU per partition for transactions 107

Sizing CPUs for parallel transactions 108

Dynamic cache capacity planning 108

Operational checklist 112

Chapter 7. Configuring the deployment environment 115

Configuration methods 115

XML files for configuration 115

Configuring data grids 115

Configuring local deployments 115

Configuring evictors 116

Plug-ins for indexing data 121

Configuring a locking strategy 125

Configuring loaders 128

Configuring write-behind loader support 133

Configuring peer-to-peer replication with JMS 146

ObjectGrid descriptor XML file 153

objectGrid.xsd file 169

Configuring deployment policies 174

Configuring distributed deployments 174

Controlling shard placement with zones 177

Configuring the heartbeat interval setting for failover detection 190

Deployment policy descriptor XML file. 192

deploymentPolicy.xsd file 197

Configuring catalog and container servers. 198

Server properties file 199

Configuring WebSphere eXtreme Scale with WebSphere Application Server. 205

Configuring the quorum mechanism 224

Best practice: Clustering the catalog service 226

Configuring multi-master replication topologies 227

Configuring ports 231

Planning for network ports. 232

Configuring ports in stand-alone mode. 233

Configuring ports in a WebSphere Application Server environment	235
Servers with multiple network cards	236
Configuring Object Request Brokers.	236
ORB properties.	237
Using the Object Request Broker with stand-alone WebSphere eXtreme Scale processes.	241
Configuring a custom Object Request Broker	241
Configuring clients	244
Client properties file	245
Configuring clients with WebSphere eXtreme Scale	247
Enabling the client invalidation mechanism	252
Configuring request retry timeout values	254
Configuring entities	256
Relationship management	256
Entity metadata descriptor XML file.	258
end.xsd file	263
Configuring cache integration	266
Configuring JPA loaders.	266
Configuring a JPA time-based data updater	268
JPA cache configuration properties	269
Configuring HTTP session managers	288
Configuring the dynamic cache provider for WebSphere eXtreme Scale	311
Configuring Spring integration	315
Spring descriptor XML file	315
Spring objectgrid.xsd file	321
Spring extension beans and namespace support	323
Starting a container server with Spring.	325
Configuring the REST data service	327
REST data service properties file	327
Administering the REST data service	339
Installing the REST data service	339

Chapter 8. Administering the deployment environment 351

Starting and stopping stand-alone servers	351
Starting stand-alone servers	351
Stopping stand-alone servers	360
Starting and stopping servers in a WebSphere Application Server environment	363
Using the embedded server API to start and stop servers	364
Embedded server API	367
Managing ObjectGrid availability.	369
Managing data center failures	371
Forcing placement to occur.	373
Administering programmatically with Managed Beans (MBeans)	374
Accessing Managed Beans (MBeans) using the wsadmin tool	375
Accessing Managed Beans (MBeans) programmatically	375

Chapter 9. Securing the deployment environment 381

Tutorial: Integrate WebSphere eXtreme Scale security with WebSphere Application Server	381
--	-----

Introduction: Integrate WebSphere eXtreme Scale security with WebSphere Application Server using the WebSphere Application Server Authentication plug-ins	381
Module 1: Prepare WebSphere Application Server	382
Module 2: Configure WebSphere eXtreme Scale to use WebSphere Application Server Authentication plug-ins	387
Module 3: Configure transport security.	393
Module 4: Use Java Authentication and Authorization Service (JAAS) authorization in WebSphere Application Server.	396
Module 5: Use the xsadmin tool to monitor data grids and maps.	402
Tutorial: Integrate WebSphere eXtreme Scale security in a mixed environment with an external authenticator	403
Introduction: Security in a mixed environment	403
Module 1: Prepare the mixed WebSphere Application Server and stand-alone environment	405
Module 2: Configure WebSphere eXtreme Scale authentication in a mixed environment.	410
Module 3: Configure transport security.	418
Module 4: Use Java Authentication and Authorization Service (JAAS) authorization in WebSphere Application Server.	422
Module 5: Use the xsadmin tool to monitor data grids and maps.	425
Security integration with WebSphere Application Server	427
Enabling local security	430
Starting and stopping secure servers.	430
Starting secure servers in a stand-alone environment.	430
Stopping secure servers	431
Starting secure servers in WebSphere Application Server	432
Data grid authentication.	433
Data grid security	433
Application client authentication	435
Application client authorization	437
Transport layer security and secure sockets layer	440
Configuring secure transport types	441
Configuring Secure Sockets Layer (SSL) parameters for clients or servers	442
Java Management Extensions (JMX) security	443
Security integration with external providers	445
Securing the REST data service	446
Security descriptor XML file	450
objectGridSecurity.xsd file	453

Chapter 10. Monitoring the deployment environment 455

Statistics overview.	455
Monitoring with the web console.	457
Starting and logging on to the web console	457
Connecting the web console to catalog servers	458
Viewing statistics with the web console.	460
Monitoring with custom reports	466
Monitoring with the statistics API	467

Statistics modules	469
Monitoring with the xsadmin utility	470
Creating a configuration profile for the xsadmin utility	473
xsadmin utility reference	474
Verbose option for the xsadmin utility	479
Monitoring with WebSphere Application Server PMI	481
Enabling PMI	481
Retrieving PMI statistics	483
PMI modules	485
Accessing Managed Beans (MBeans) using the wsadmin tool	492
Monitoring with managed beans (MBeans)	493
Monitoring with vendor tools	494
Monitoring with the IBM Tivoli Enterprise Monitoring Agent for WebSphere eXtreme Scale	494
Monitoring eXtreme Scale applications with CA Wily Introscope	500
Monitoring eXtreme Scale with Hyperic HQ	503
Monitoring eXtreme Scale information in DB2	505

Chapter 11. Tuning and performance 507

Operational checklist	507
Operating systems and network tuning	509
Planning for network ports	509
ORB properties	511
JVM tuning for WebSphere eXtreme Scale	515
Configuring the heartbeat interval setting for failover detection	517

Using WebSphere Real Time	519
WebSphere Real Time in a stand-alone environment	519
WebSphere Real Time in WebSphere Application Server	521
Tuning the dynamic cache provider	523
Tuning the cache sizing agent for accurate memory consumption estimates	524
Cache memory consumption sizing	525

Chapter 12. Troubleshooting 529

Enabling logging	529
Collecting trace	530
Trace options	531
Troubleshooting installation	533
Troubleshooting client connectivity	533
Troubleshooting loaders	534
Troubleshooting XML configuration	535
Troubleshooting security	538
IBM Support Assistant for WebSphere eXtreme Scale	538
Messages	540
Release notes	540

Notices 543

Trademarks 545

Index 547

About the *Administration Guide*

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

WebSphere eXtreme Scale library

The WebSphere eXtreme Scale library contains the following books:

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

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

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

Who should use this book

This book is intended primarily for system administrators, security administrators, and system operators.

Getting updates to this book

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

How to send your comments

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

Chapter 1. Running the getting started sample application

After you install WebSphere eXtreme Scale in a stand-alone environment, use the following steps as a simple introduction to its capability as an in-memory data grid.

The stand-alone installation of WebSphere eXtreme Scale includes a sample that you can use to verify your installation and to see how a simple data grid and client can be used. The getting started sample is in the `wxs_install_root/ObjectGrid/gettingstarted` directory.

The getting started sample provides a quick introduction to eXtreme Scale functionality and basic operation. The sample consists of shell and batch scripts designed to start a simple data grid with very little customization needed. In addition, a client program, including source, is provided to run simple create, read, update, and delete (CRUD) functions to this basic data grid.

Scripts and their functions

This sample provides the following four scripts:

The `env.sh|bat` script is called by the other scripts to set needed environment variables. Normally you do not need to change this script.

- `UNIX` `Linux` `./env.sh`
- `Windows` `env.bat`

The `runcat.sh|bat` starts the eXtreme Scale catalog service process on the local system.

- `UNIX` `Linux` `./runcat.sh`
- `Windows` `runcat.bat`

The `runcontainer.sh|bat` script starts a container server process. You can run this script multiple times with unique server names specified to start any number of containers. These instances can work together to host partitioned and redundant information in the grid.

- `UNIX` `Linux` `./runcontainer.sh unique_server_name`
- `Windows` `runcontainer.bat unique_server_name`

The `runclient.sh|bat` script runs the simple CRUD client and starts the given operation.

- `UNIX` `Linux` `./runclient.sh command value1 value2`
- `Windows` `runclient.sh 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 keyed by *value1* to *value2*
- Specify as *d* to delete object keyed by *value1*
- Specify as *g* to retrieve and display object keyed by *value1*

Note: The `installRoot/ObjectGrid/gettingstarted/src/Client.java` file is the client program that demonstrates how to connect to a catalog server, obtain an ObjectGrid instance, and use the ObjectMap API.

Basic steps

Use the following steps to start your first data grid and run a client to interact with the data grid.

1. Open a terminal session or command line window.
2. Use the following command to navigate to the gettingstarted directory:
`cd wxs_install_root/ObjectGrid/gettingstarted`
Substitute `wxs_install_root` with the path to the eXtreme Scale installation root directory or the root file path of the extracted eXtreme Scale trial `wxs_install_root`.

3. Run the following script to start a catalog service process on localhost:

- `UNIX Linux ./runcat.sh`
- `Windows runcat.bat`

The catalog service process runs in the current terminal window.

4. Open another terminal session or command line window, and run the following command to start a container server instance:

- `UNIX Linux ./runcontainer.sh server0`
- `Windows runcontainer.bat server0`

The container server runs in the current terminal window. You can repeat this step with a different server name if you want to start more container server instances to support replication.

5. Open another terminal session or command line window to run client commands.

- Add data to the data grid:
 - `UNIX Linux ./runclient.sh i key1 helloWorld`
 - `Windows runclient.bat i key1 helloWorld`

- Search and display the value:
 - `UNIX Linux ./runclient.sh g key1`
 - `Windows runclient.bat g key1`

- Update the value:
 - `UNIX Linux ./runclient.sh u key1 goodbyeWorld`
 - `Windows runclient.bat u key1 goodbyeWorld`

- Delete the value:
 - `UNIX Linux ./runclient.sh d key1`
 - `Windows runclient.bat d key1`

6. Use `<ctrl+c>` to stop the catalog service process and container servers in the respective windows.

Defining an ObjectGrid

The sample uses the `objectgrid.xml` and `deployment.xml` files that are in the `wxs_install_root/ObjectGrid/gettingstarted/xml` directory to start a container

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 ObjectGrid topology.

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 BackingMap configurations. These BackingMaps are the actual data storage for cached 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 BackingMaps.

```
<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

A deployment policy descriptor XML file is passed to an ObjectGrid container server during startup. A deployment policy must be used with an ObjectGrid XML file and must be compatible with the ObjectGrid XML that is used with it. For each `objectgridDeployment` element in the deployment policy, you must have a corresponding ObjectGrid element in your ObjectGrid XML. The `backingMap` elements that are defined within the `objectgridDeployment` element must be consistent with the `backingMaps` found in the ObjectGrid XML. Every `backingMap` must be referenced within one and only one `mapSet`.

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. Both the Map1 and Map2 BackingMaps that are defined within the Grid ObjectGrid are included in the `mapSet` `mapSet` that has the `numberOfPartitions`, `minSyncReplicas`, and `maxSyncReplicas` attributes configured.

```
<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 `mapSet`. It is an optional attribute and the default is 1. The number should be appropriate for the anticipated capacity of the data grid.

The `minSyncReplicas` attribute of `mapSet` is to specify the minimum number of synchronous replicas for each partition in the `mapSet`. It is an optional attribute and the default is 0. Primary and replica are not placed until the domain can support the minimum number of synchronous replicas. To support the `minSyncReplicas` value, you need one more container than the value of `minSyncReplicas`. If the number of synchronous replicas falls below the value of `minSyncReplicas`, write transactions are no longer allowed for that partition.

The `maxSyncReplicas` attribute of `mapSet` is to specify the maximum number of synchronous replicas for each partition in the `mapSet`. It is an optional attribute and the default 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 containers that can support this `ObjectGrid` can result in an increased number of synchronous replicas if your `maxSyncReplicas` value has not already been met. The sample set the `maxSyncReplicas` to 1 means the domain will at most place one synchronous replica. If you start more than one container server instance, there will be only one synchronous replica placed in one of the container server instances.

Using ObjectGrid

The `Client.java` file in the `wxs_install_root/ObjectGrid/gettingstarted/client/src/` directory is the client program that demonstrates how to connect to catalog server, obtain `ObjectGrid` instance, and use `ObjectMap` API.

From the perspective of a client application, using WebSphere eXtreme Scale can be divided into the following steps.

1. Connecting to the catalog service by obtaining a `ClientClusterContext` instance.
2. Obtaining a client `ObjectGrid` instance.
3. Getting a `Session` instance.
4. Getting an `ObjectMap` instance.
5. Using the `ObjectMap` methods.

- 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 `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");  
ObjectMap map1 = sess.getMap("my_simple_data_grid");
```

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(key1, value1);
```

The following code snippet demonstrates how to use the ObjectMap API with explicit transaction demarcation.

```
sess.begin();  
map1.insert(key1, value1);  
sess.commit();
```

Additional information

This sample demonstrates how to start catalog server and container server and using ObjectMap API in stand-alone environment. You can also use the EntityManager API.

In a WebSphere Application Server environment with WebSphere eXtreme Scale installed or enabled, the most common scenario is a network-attached topology. In a network-attached topology, the catalog server is hosted in the deployment manager process and each WebSphere Application Server instance hosts a container server automatically. Java Platform, Enterprise Edition applications only need to include both the ObjectGrid descriptor XML file and the ObjectGrid deployment policy descriptor XML file in the META-INF directory of any module and the ObjectGrid becomes available automatically. An application can then connect to a locally available catalog server and obtain an ObjectGrid instance to use.

Directory conventions

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

wxs_install_root

The *wxs_install_root* directory is the root directory where WebSphere eXtreme Scale product files are installed. The *wxs_install_root* directory can be the directory in which the trial zip file 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:

Example: /opt/IBM/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 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 sub-directory 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:

Example: /opt/IBM/eXtremeScale/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 sub-directory under the *wxs_home* directory.

- Example for stand-alone deployments:

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

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

samples_home

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

Example:/wxs-samples/

Chapter 2. Capacity planning

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. Although such planning helps you deploy eXtreme Scale efficiently for future changes, it allows you to maximize the elasticity of eXtreme Scale 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.

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.

```
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.

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.

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.

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.

Dynamic cache capacity planning

The Dynamic Cache API is available to Java EE applications that are deployed in WebSphere Application Server. The dynamic cache can be leveraged 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:

$$256\text{mb} * 4 \text{ containers} / (1 \text{ primary} + 1 \text{ replica}) = 512\text{mb}.$$

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

$$N = \text{Minimum}(T - 1, R)$$

Where:

- N = the number of processes that can crash simultaneously
- T = the total number of containers
- R = the total number of replicas

Formula C

$$\text{Ceiling}(T / (1+N)) = m$$

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 the [Tuning 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.

Chapter 3. Installing and deploying WebSphere eXtreme Scale

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

- Establish how WebSphere eXtreme Scale fits into your current topology.
- Verify that your environment meets the prerequisites to install eXtreme Scale. See “Hardware and software requirements” on page 74 for more information.
- For more information on environments and other requirements, see Chapter 6, “Planning the WebSphere eXtreme Scale environment,” on page 73 or “Operational checklist” on page 112.

About this task

Environment options

- **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.

7.1+ Installation types

The full installer and the separate client installer that you can download from the support site give you a variety of installation options. You can use a client-only installation on nodes that are running only the client applications that access the data grid. Use the server installation or server and client installation on nodes that run catalog servers or container servers.

- **Full installation:**

- When you are installing on WebSphere Application Server, you can choose to install the client only or both the server and the client.
- When you are installing in a stand-alone environment, you can install both the client and server. If you want to install the client only, use the WebSphere eXtreme Scale Client installation.

- **Client installation::**

You can use the client-only installation on nodes that are running the client applications. To install the client only, you can download the client only installer for the appropriate platform from the downloads section on the Support site

Installing stand-alone WebSphere eXtreme Scale or WebSphere eXtreme Scale Client

You can install stand-alone WebSphere eXtreme Scale or WebSphere eXtreme Scale Client in an environment that does not contain WebSphere Application Server or WebSphere Application Server Network Deployment.

Before you begin

- Verify that the target installation directory is empty or does not exist.

Important: If a previous version of WebSphere eXtreme Scale or the ObjectGrid component exists in the directory that you specify to install Version 7.1, the product is not installed. For example, you might have a previously existing `wxs_install_root/ObjectGrid` folder. You can either select a different installation directory or cancel the installation. Next, uninstall the previous installation and run the wizard again.

- **7.1+** An IBM® Runtime Environment is installed as a part of the stand-alone installation in the `wxs_install_root/java` folder.
- If you are installing the client only: Download the WebSphere eXtreme Scale Client for the appropriate platform from the Support site.

About this task

When you install the product as stand-alone, you install the WebSphere eXtreme Scale client and server together. With the WebSphere eXtreme Scale Client installation in stand-alone mode, you are installing a client to access the data in your data grids. Server and client processes, therefore, access all required resources locally. You can also embed WebSphere eXtreme Scale into existing Java Platform, Standard Edition (J2SE) applications by using scripts and Java archive (JAR) files.

Attention: You can also use a non-root (non-administrator) profile for WebSphere eXtreme Scale in a stand-alone environment. To use a non-root profile, 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.

Procedure

1. Use the wizard to install both the server and the client from the DVD.
 - Run the following script to start the wizard for the WebSphere eXtreme Scale full installation:
 - `Linux` `UNIX` `dvd_root/install`
 - `Windows` `dvd_root\install.bat`
 - Run the following script to start the wizard for the WebSphere eXtreme Scale Client installation. The installation files are in the zip file that you download from the Support site:
 - `Linux` `UNIX` `root/WXS_Client/install`
 - `Windows` `root\WXS_Client\install.bat`

Attention: If you use uniform naming conventions (UNC) to identify file paths in your installation command, the items you anticipate installing may not all be installed after the command runs. To avoid trouble, map the file path to a network drive. Run the **install** command against the mapped drive. Using a mapped network drive ensures that all the items are installed.

2. Follow the prompts in the wizard, and click **Finish**.

Restriction: The optional features panel lists the features from which you can select to install. However, features cannot be added incrementally to the product environment after the product is installed. If you choose not to install a feature with the initial product installation, you must uninstall and reinstall the product to add the feature.

Results

Windows 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.
```

If you see a failure with the `iscdeploy.sh` file, you can ignore the error. This error does not cause any problems.

What to do next

You can verify your installation by running the getting started sample application. See Chapter 1, “Running the getting started sample application,” on page 1 for more information.

See Chapter 7, “Configuring the deployment environment,” on page 115 to set up your client application processes and server processes.

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 used by the server runtime environment of J2SE Version 1.4.2 and later. The file is automatically included in the server runtime environment when you use the supplied scripts.

Table 1. Runtime files for WebSphere eXtreme Scale full installation (continued). 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
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 contains only the local and client runtime environments. You can use this file with J2SE Version 1.4.2 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 6.0.2 and later. This file contains only the local and client runtime environments.
wssizeagent.jar	Local, client, and server	lib	The wssizeagent.jar file is used to provide more accurate cache entry sizing information when using Java runtime environment (JRE) Version 1.5 or later.
ibmcfw.jar ibmext.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 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.

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
wxdynacache.jar	Client and server	dynacache/lib	The wxdynacache.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 contains only the local and client runtime environments. You can use this file with J2SE Version 1.4.2 and later.
ogspring.jar	Local, client, and server	lib	The ogspring.jar file contains support classes for the SpringSource Spring framework integration.

Table 2. Runtime files for WebSphere eXtreme Scale Client (continued). 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
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.
ibmcfw.jar ibmext.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 HTTP session listener.
splicer.jar	Utility	legacy/session/lib	The splicer utility for the WebSphere eXtreme Scale Version 7.0 HTTP session manager filter.

Running the getting started sample application

After you install WebSphere eXtreme Scale in a stand-alone environment, use the following steps as a simple introduction to its capability as an in-memory data grid.

The stand-alone installation of WebSphere eXtreme Scale includes a sample that you can use to verify your installation and to see how a simple data grid and client can be used. The getting started sample is in the *wxs_install_root/ObjectGrid/gettingstarted* directory.

The getting started sample provides a quick introduction to eXtreme Scale functionality and basic operation. The sample consists of shell and batch scripts designed to start a simple data grid with very little customization needed. In addition, a client program, including source, is provided to run simple create, read, update, and delete (CRUD) functions to this basic data grid.

Scripts and their functions

This sample provides the following four scripts:

The `env.sh|bat` script is called by the other scripts to set needed environment variables. Normally you do not need to change this script.

- `UNIX` `Linux` `./env.sh`
- `Windows` `env.bat`

The `runcat.sh|bat` starts the eXtreme Scale catalog service process on the local system.

- `UNIX Linux ./runcat.sh`
- `Windows runcat.bat`

The `runcontainer.sh|bat` script starts a container server process. You can run this script multiple times with unique server names specified to start any number of containers. These instances can work together to host partitioned and redundant information in the grid.

- `UNIX Linux ./runcontainer.sh unique_server_name`
- `Windows runcontainer.bat unique_server_name`

The `runclient.sh|bat` script runs the simple CRUD client and starts the given operation.

- `UNIX Linux ./runclient.sh command value1 value2`
- `Windows runclient.sh 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 keyed by *value1* to *value2*
- Specify as *d* to delete object keyed by *value1*
- Specify as *g* to retrieve and display object keyed by *value1*

Note: The `installRoot/ObjectGrid/gettingstarted/src/Client.java` file is the client program that demonstrates how to connect to a catalog server, obtain an ObjectGrid instance, and use the ObjectMap API.

Basic steps

Use the following steps to start your first data grid and run a client to interact with the data grid.

1. Open a terminal session or command line window.
2. Use the following command to navigate to the `gettingstarted` directory:
`cd wxs_install_root/ObjectGrid/gettingstarted`
Substitute `wxs_install_root` with the path to the eXtreme Scale installation root directory or the root file path of the extracted eXtreme Scale trial `wxs_install_root`.
3. Run the following script to start a catalog service process on localhost:

- `UNIX Linux ./runcat.sh`
- `Windows runcat.bat`

The catalog service process runs in the current terminal window.

4. Open another terminal session or command line window, and run the following command to start a container server instance:

- `UNIX Linux ./runcontainer.sh server0`
- `Windows runcontainer.bat server0`

The container server runs in the current terminal window. You can repeat this step with a different server name if you want to start more container server instances to support replication.

5. Open another terminal session or command line window to run client commands.
 - Add data to the data grid:
 - `UNIX Linux ./runclient.sh i key1 helloWorld`
 - `Windows runclient.bat i key1 helloWorld`
 - Search and display the value:
 - `UNIX Linux ./runclient.sh g key1`
 - `Windows runclient.bat g key1`
 - Update the value:
 - `UNIX Linux ./runclient.sh u key1 goodbyeWorld`
 - `Windows runclient.bat u key1 goodbyeWorld`
 - Delete the value:
 - `UNIX Linux ./runclient.sh d key1`
 - `Windows runclient.bat d key1`
6. Use `<ctrl+c>` to stop the catalog service process and container servers in the respective windows.

Defining an ObjectGrid

The sample uses the `objectgrid.xml` and `deployment.xml` files that are in the `wxs_install_root/ObjectGrid/gettingstarted/xml` directory to start a container 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 ObjectGrid topology.

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 BackingMap configurations. These BackingMaps are the actual data storage for cached 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 BackingMaps.

```
<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

A deployment policy descriptor XML file is passed to an ObjectGrid container server during startup. A deployment policy must be used with an ObjectGrid XML file and must be compatible with the ObjectGrid XML that is used with it. For each `objectgridDeployment` element in the deployment policy, you must have a

corresponding ObjectGrid element in your ObjectGrid XML. The backingMap elements that are defined within the objectgridDeployment element must be consistent with the backingMaps found in the ObjectGrid XML. Every backingMap must be referenced within one and only one mapSet.

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. Both the Map1 and Map2 BackingMaps that are defined within the Grid ObjectGrid are included in the mapSet mapSet that has the numberOfPartitions, minSyncReplicas, and maxSyncReplicas attributes configured.

```
<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 mapSet. It is an optional attribute and the default is 1. The number should be appropriate for the anticipated capacity of the data grid.

The minSyncReplicas attribute of mapSet is to specify the minimum number of synchronous replicas for each partition in the mapSet. It is an optional attribute and the default is 0. Primary and replica are not placed until the domain can support the minimum number of synchronous replicas. To support the minSyncReplicas value, you need one more container than the value of minSyncReplicas. If the number of synchronous replicas falls below the value of minSyncReplicas, write transactions are no longer allowed for that partition.

The maxSyncReplicas attribute of mapSet is to specify the maximum number of synchronous replicas for each partition in the mapSet. It is an optional attribute and the default 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 containers that can support this ObjectGrid can result in an increased number of synchronous replicas if your maxSyncReplicas value has not already been met. The sample set the maxSyncReplicas to 1 means the domain will at most place one synchronous replica. If you start more than one container server instance, there will be only one synchronous replica placed in one of the container server instances.

Using ObjectGrid

The Client.java file in the *wxs_install_root*/ObjectGrid/gettingstarted/client/src/ directory is the client program that demonstrates how to connect to catalog server, obtain ObjectGrid instance, and use ObjectMap API.

From the perspective of a client application, using WebSphere eXtreme Scale can be divided into the following steps.

1. Connecting to the catalog service by obtaining a `ClientClusterContext` instance.
2. Obtaining a client `ObjectGrid` instance.
3. Getting a `Session` instance.
4. Getting an `ObjectMap` instance.
5. Using the `ObjectMap` methods.

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 `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");  
ObjectMap map1 = sess.getMap("my_simple_data_grid");
```

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(key1, value1);
```

The following code snippet demonstrates how to use the ObjectMap API with explicit transaction demarcation.

```
sess.begin();  
map1.insert(key1, value1);  
sess.commit();
```

Additional information

This sample demonstrates how to start catalog server and container server and using ObjectMap API in stand-alone environment. You can also use the EntityManager API.

In a WebSphere Application Server environment with WebSphere eXtreme Scale installed or enabled, the most common scenario is a network-attached topology. In a network-attached topology, the catalog server is hosted in the deployment manager process and each WebSphere Application Server instance hosts a container server automatically. Java Platform, Enterprise Edition applications only need to include both the ObjectGrid descriptor XML file and the ObjectGrid deployment policy descriptor XML file in the META-INF directory of any module and the ObjectGrid becomes available automatically. An application can then connect to a locally available catalog server and obtain an ObjectGrid instance to use.

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

- Install WebSphere Application Server or WebSphere Application Server Network Deployment. See *Installing your application serving environment* for more information.
- Based on what version you install, Version 6.0.x, Version 6.1, or Version 7.0, apply the latest fix pack for WebSphere Application Server or WebSphere Application Server Network Deployment to update your product level. See the *Latest fix packs for WebSphere Application Server* for more information.
- 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:

Ensure that any running processes are stopped. If the running processes are not stopped, the installation proceeds, creating unpredictable results and leaving the installation in an undetermined state on some platforms.

- If you are installing the client only, you can either use the DVD to install the client or download the WebSphere eXtreme Scale Client for the specific platform from the downloads section on the Support site.

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.

Procedure

1. Use the wizard to complete the installation.
 - Run the following script to start the wizard for the WebSphere eXtreme Scale full installation. You can choose to install the client only or both the server and client:

```
- Linux UNIX dvd_root/install
```

```
- Windows dvd_root\install.bat
```

- Run the following script to start the wizard for the WebSphere eXtreme Scale Client installation. The installation files are in the zip file that you download from the downloads section on the Support site:

```
- Linux UNIX root/WXS_Client/install
```

```
- Windows root\WXS_Client\install.bat
```

Attention: If you use uniform naming conventions (UNC) to identify file paths in your installation command, the items you anticipate installing may not all be installed after the command runs. To avoid trouble, map the file path to a network drive. Run the **install** command against the mapped drive. Using a mapped network drive ensures that all the items are installed.

2. Follow the prompts in the wizard.

The optional features panel lists the features from which you can choose to install. However, features cannot be added incrementally to the product environment after the product is installed. If you choose not to install a feature with the initial product installation, you must uninstall and reinstall the product to add the feature.

The Profile augmentation panel lists existing profiles that you can select to augment with the features of eXtreme Scale. If you select existing profiles that are already in use, however, a warning panel is displayed. To continue with the installation, either stop the servers that are configured in the profiles, or click **Back** to remove the profiles from your selection.

Results

Windows 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.
```

If you see a failure with the `iscdeploy.sh` file, you can ignore the error. This error does not cause any problems.

What to do next

If you are running WebSphere Application Server Version 6.1 or Version 7.0, you can use the Profile Management Tool plug-in or the **manageprofiles** command. If you are running WebSphere Application Server Version 6.0.2, you must use the **wasprofile** command to create and augment profiles.

Deploy your application, start a catalog service, and start the containers in your WebSphere Application Server environment. See “Configuring WebSphere eXtreme Scale with WebSphere Application Server” on page 205 for more information.

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 3. 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.
wsoobjectgrid.jar	Local and client	lib	The wsoobjectgrid.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.
ogsip.jar	Server	lib	The ogsip.jar file contains the eXtreme Scale Session Initiation Protocol (SIP) session management runtime that is compatible with WebSphere Application Server Version 6.1.x.
sessionobjectgrid.jar	Client and server	lib	The sessionobjectgrid.jar file contains the eXtreme Scale HTTP session management runtime.
sessionobjectgridsip.jar	Server	lib	The sessionobjectgridsip.jar file contains the eXtreme Scale SIP session management runtime that is compatible with WebSphere Application Server Version 7.x.
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.

Table 3. Runtime files for WebSphere eXtreme Scale (continued). 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
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 4. 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
wxdynacache.jar	Client and server	lib	The wxdynacache.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.
ogsip.jar	Server	lib	The ogsip.jar file contains the eXtreme Scale Session Initiation Protocol (SIP) session management runtime that is compatible with WebSphere Application Server Version 6.1.x.
sessionobjectgrid.jar	Client and server	lib	The sessionobjectgrid.jar file contains the eXtreme Scale HTTP session management runtime.
sessionobjectgridsip.jar	Server	lib	The sessionobjectgridsip.jar file contains the eXtreme Scale SIP session management runtime that is compatible with WebSphere Application Server Version 7.x.
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.
wssizeagent.jar	Local, client, and server	lib	The wssizeagent.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.
wshyperic.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.

Table 4. Runtime files for WebSphere eXtreme Scale Client (continued). 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
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.

Using the Installation Factory plug-in to create and install customized packages

Use the IBM Installation Factory plug-in for WebSphere eXtreme Scale to create a customized installation package (CIP) or an integrated installation package (IIP). A CIP contains a single product installation package and various optional assets. An IIP combines one or more installation packages into a single installation workflow that you design.

Before you begin

Before you create and install customized packages for eXtreme Scale, you must first download the following products:

- IBM Installation Factory for WebSphere Application Server
- IBM Installation Factory plug-in for WebSphere eXtreme Scale

About this task

Using the Installation Factory, you can create a CIP by combining a single product component with maintenance packages, customization scripts, and other files. When you create an IIP, you aggregate individual components, or installation packages, into a single installation package.

Build definition file

A build definition file is an XML document that specifies how to build and install a customized installation package (CIP) or an integrated installation package (IIP). The IBM Installation Factory for WebSphere eXtreme Scale reads the package details of the build definition file to generate a CIP or an IIP.

Before you can create a CIP or an IIP, you must create a build definition file for each customized package. The build definition file describes which product components, or installation packages, to install, the location of the CIP or IIP, the maintenance packages to include, the installation scripts, and other files that you choose to include. You can also specify in the build definition file for the IIP the order in which the Installation Factory installs each installation package.

The Build definition wizard steps you through the process of creating a build definition file. You can also use the wizard to modify an existing build definition file. Each panel in the Build definition wizard prompts you for information about a customized package, such as the package identification, the installation location for the build definition, and the installation location for the customized package. All of this information is saved in the new build definition file, or modified and saved in

an existing build definition file. For more information, see the CIP Build definition wizard panels and the IIP Build definition wizard panels.

To create only the build definition file, you can use the command-line interface tool to generate the customized package outside of the GUI. See “Silently installing a CIP or an IIP” on page 37 for more information.

Creating a build definition file and generating a CIP




The IBM Installation Factory plug-in for WebSphere eXtreme Scale generates a customized installation package (CIP) according to the details that you specify in the build definition file. The build definition specifies the product package to install, the location of the CIP, the maintenance packages to include in the installation, the install script files, and any additional files to include in the CIP.

About this task

You can use the Build definition wizard to create a build definition file and generate a CIP.

Procedure

1. Run the following script from the *IF_HOME/bin* directory to start the Installation Factory:

-   `ifgui.sh`
-  `ifgui.bat`

Click the **New Build Definition** icon.

2. Select the product to include in the build definition file, and click **Finish** to start the Build definition wizard.
3. Follow the prompts in the wizard.

On the Install and Uninstall Scripts panel, click **Add Scripts...** to populate the table with any customized installation scripts. Type the location of the script files, and clear the check box to continue if an error message is displayed. The operation is stopped by default. Click **OK** to return to the panel.

Results

You created and customized the build definition file, and you generated the CIP if you chose to work in the connected mode.

If the Build definition wizard does not provide you with the option to generate the CIP from the build definition file, you can still generate it by running the `ifcli.sh|bat` script from the *IF_HOME/bin* directory.

What to do next

Install the CIP. See “Installing a CIP” for more information.

Installing a CIP:

Simplify the product installation process by installing a customized installation package (CIP). A CIP is a single product installation image that can include one or more maintenance packages, configuration scripts, and other files.

Before you begin

Before you can install a CIP, you must create a build definition file to specify what options to include in the CIP. See “Creating a build definition file and generating a CIP” on page 31 for more information.

About this task

A CIP combines and installs a single product component with maintenance packages, customization scripts, and other files.

Procedure

1. Stop all processes that are running on the workstation you are preparing for installation. To stop the deployment manager, run the following script:

- `Linux` `UNIX` `profile_root/bin/stopManager.sh`
- `Windows` `profile_root\bin\stopManager.bat`

To stop the nodes, run the following script:

- `Linux` `UNIX` `profile_root/bin/stopNode.sh`
- `Windows` `profile_root\bin\stopNode.bat`

2. Run the following script to start the installation:

- `Linux` `UNIX` `CIP_home/bin/install`
- `Windows` `CIP_home\bin\install.bat`

3. Follow the prompts in the wizard to complete the installation.

The optional features panel lists the features from which you can choose to install. However, features cannot be added incrementally to the product environment after the product is installed. If you choose not to install a feature with the initial product installation, you must uninstall and reinstall the product to add the feature.

The Profile augmentation panel lists existing profiles that you can select to augment with the features of eXtreme Scale. If you select existing profiles that are already in use, however, a warning panel is displayed. To continue with the installation, either stop the servers that are configured in the profiles, or click **Back** to remove the profiles from your selection.

Results

You successfully installed the CIP.

What to do next

If you are running WebSphere Application Server Version 6.1 or Version 7.0, you can use the Profile Management Tool plug-in or the **manageprofiles** command to create and augment profiles. If you are running WebSphere Application Server Version 6.0.2, you must use the **wasprofile** command to create and augment profiles. See “Creating and augmenting profiles for WebSphere eXtreme Scale” on page 45 for more information.

If you augmented profiles for eXtreme Scale during the installation process, you can deploy applications, start a catalog service, and start the containers in your WebSphere Application Server environment. See “Configuring WebSphere eXtreme Scale with WebSphere Application Server” on page 205 for more information.

Installing a CIP to apply maintenance to an existing product installation:

You can apply maintenance packages to an existing product installation by installing a customized installation package (CIP). The process of applying maintenance to an existing installation with a CIP is commonly referred to as a *slip installation*.

Before you begin

Create a build definition file to specify what options to include in the CIP. See “Creating a build definition file and generating a CIP” on page 31 for more information.

About this task

When applying maintenance with a CIP that contains a refresh pack, a fix pack, or both, all previously installed authorized program analysis reports (APAR) are uninstalled by the wizard. If the CIP is at the same level as the product, previously installed APARs remain only if they are packaged in the CIP. To successfully apply maintenance to an existing installation, you must include the installed features in the CIP.

Procedure

1. Stop all processes that are running on the workstation you are preparing for installation. To stop the deployment manager, run the following script:

- `Linux` `UNIX` `profile_root/bin/stopManager.sh`
- `Windows` `profile_root\bin\stopManager.bat`

To stop the nodes, run the following script:

- `Linux` `UNIX` `profile_root\bin\stopNode.sh`
- `Windows` `profile_root\bin\stopNode.bat`

2. Run the following script to start the installation:

- `Linux` `UNIX` `CIP_home/bin/install`
- `Windows` `CIP_home\bin\install.bat`

3. Follow the prompts in the wizard to complete the installation.

The installation preview summary lists the resulting product version and any applicable features and interim fixes. Next, the wizard successfully applies the maintenance, and updates the features of the product.

Results

The product binary files are copied to the `was_root/properties/version/nif/backup` directory. You can use the IBM Update Installer to uninstall the update and restore your workstation. See “Uninstalling CIP updates from an existing product installation” for more information.

Uninstalling CIP updates from an existing product installation:

You can remove CIP updates from an existing product installation without removing the entire product. Use the IBM Update Installer Version 7.0.0.4 to uninstall any CIP updates. This task is also referred to as a *slip uninstallation*.

Before you begin

You must have at least one existing copy of the product installed on the system.

Procedure

1. Download Version 7.0.0.4 of the Update Installer from the following FTP site:
`ftp://ftp.software.ibm.com/software/websphere/cw/process_server/FEP/UPDI/7004`
2. Install the Update Installer. See *Installing the Update Installer for WebSphere Software in the WebSphere Application Server Information Center* for more information.
3. Uninstall any fix pack, refresh pack, or interim fix that you added to your environment after you installed the CIP.
4. Uninstall any interim fixes that you included in the slip installation. This process is the same as uninstalling a single fix pack or refresh pack. However, the maintenance that was included in the CIP is now included in a single operation.
5. Uninstall the CIP by using the Update Installer. The maintenance levels return to the pre-update state, and the CIP is denoted by the CIP identifier that is added as a prefix to its file name. The following example shows how a CIP is displayed differently than other regular maintenance packages on the maintenance package selection panel:

CIP

```
com.ibm.ws.cip.7000.wxs.primary.ext.pak
```

Results

You successfully removed the CIP updates from an existing product installation.

Creating a build definition file and generating an IIP

The IBM Installation Factory plug-in for WebSphere eXtreme Scale generates an IIP based on the properties that the build definition file provides. The build definition file contains information such as which installation packages to include in the IIP, the order in which the Installation Factory installs each package, and the location of the IIP.

About this task

You can use the Build definition wizard to create a build definition file and generate an IIP.

Procedure

1. Run the following script from the `IF_HOME/bin` directory to start the Installation Factory:
 - `UNIX` `Linux` `ifgui.sh`
 - `Windows` `ifgui.bat`
2. Click the **Create New Integrated Installation Package** icon to start the Build definition wizard.
3. Follow the prompts in the wizard.
 - a. On the Construct the IIP panel, select a supported installation package from the list, and click **Add Installer** to add the installation package to the IIP. A panel that displays the package name, the package identifier, and the

package properties is displayed. To view specific information about the selected package, click **View Installation Package Information**. Click **Modify** to enter the directory path to the installation package for each operating system. If you are currently adding an installation package for WebSphere Extended Deployment, select the checkbox, which provides you with the option to use the same package for all supported operating systems. Click **OK** and return to the Construct the IIP panel. An invocation is created by default.

- To modify the directory path to an installation package, select the package from the Installation packages used in the IIP list, and click **Modify**.
 - To modify an invocation, select the invocation, and click **Modify**. Specify the default installation location for the invocation on each operating system. Specify the location to the response file if you select a silent installation as the default installation mode.
 - Click **Add Invocation** to add an invocation contribution to the installation package. A panel from which you can specify properties for the invocation is displayed.
 - Click **Remove** to remove installation packages or invocations.
4. Review the summary of your selections, select the **Save build definition file and generate integrated installation package** option, and click **Finish**.
Alternatively, you can save the build definition file without generating the IIP. With this option, you actually generate the IIP outside of the wizard by running the `ifcli.bat | ifcli.sh` script from the `IF_home/bin/` directory.

Results

You created and customized the build definition file for an IIP.

What to do next

Install the IIP.

Installing an IIP:

Use the IBM Installation Factory plug-in for WebSphere eXtreme Scale to install an integrated installation package (IIP). An IIP combines one or more installation packages into a single workflow that you design.

Before you begin

Before you can install a CIP, you must create a build definition file to specify what options to include in the CIP. See “Creating a build definition file and generating an IIP” on page 34 for more information.

About this task

An IIP can include one or more generally available installation packages, one or more CIPs, and other optional files and directories. By installing an IIP, you aggregate multiple installation packages, or *contributions*, into a single package, and you then install the contributions in a specific order to complete an end-to-end installation.

Procedure

1. Run the following script to start the wizard:
 - `Linux` `UNIX` `IIP_home/bin/install`
 - `Windows` `IIP_home\bin\install.bat`
2. Click **About** on the Welcome panel to view the details of the IIP, such as the package identifier, the supported operating systems, and the included installation packages.

Optional: To modify the installation options for each package, click **Modify**.

Optional: Two **View Log** buttons are displayed on the wizard panel. To view the log of each package, click the **View Log** button that is displayed next to the table that lists the installation packages. To view the overall log details of the IIP, click the **View Log** button that is displayed next to the status information.

3. Select the installation packages to run, and click **Install**. A list of all the contributions in the order of invocation that the IIP contains is displayed. To designate which contribution invocations should not be run during the installation, clear the checkbox located next to the **Installation name** field.

Results

You successfully installed an IIP.

Modifying an existing build definition file for an IIP:

You can edit or add to the properties of an IIP to further customize the installation.

About this task

To change the properties of an IIP, modify the existing build definition file.

Procedure

1. Run the following script from the `IF_HOME/bin` directory to start the Installation Factory:
 - `UNIX` `Linux` `ifgui.sh`
 - `Windows` `ifgui.bat`
2. Click the **Open Build Definition** icon, and select the build definition file that you want to modify.
3. Select the specific properties of the IIP that you want to modify. The following list contains the possible modifications that you can make:
 - Change your current mode selection. In connected mode, you create the build definition for use, and optionally generate the IIP, from your current workstation. In disconnected mode, you create the build definition file for use on another workstation.
 - Add or remove the existing operating systems that the IIP supports.
 - Edit the existing identifier and version for the IIP.
 - Edit the target location for the build definition file.
 - Edit the target location for the IIP.
 - Change whether to display an installation wizard for the IIP. The wizard provides information about the IIP and the installation options when the IIP runs.

- Add, remove, and edit the installation packages that are contained in the IIP.

Important: If you added a supported operating system and you have not updated the properties of the installation package in the IIP, you receive a warning message stating that the selected contributions do not contain installation packages that are identified for all of the operating systems that the IIP supports. Click **Yes** to continue, or click **No** to edit the installation package.

4. Review the summary of your selections, select **Save build definition file and generate integrated installation package**, and click **Finish**.

Silently installing a CIP or an IIP

You can silently install a customized installation package (CIP) or an integrated installation package (IIP) for the product by using either a fully-qualified response file, which you configure specifically to your needs, or parameters that you pass to the command line.

Before you begin

Create the build definition file for the CIP or IIP. See “Creating a build definition file and generating a CIP” on page 31 for more information.

About this task

A silent installation uses the same installation program that the graphical user interface (GUI) version uses. However, instead of displaying a wizard interface, the silent installation reads all of your responses from a file that you customize, or from parameters that you pass to the command line. If you are silently installing an IIP, you can invoke a contribution with a combination of options that you specify directly on the command line, as well as options that you specify in a response file. However, any contribution options that you pass to the command line causes the IIP installer to ignore all of the options that are specified in a specific contribution's response file. See the detailed IIP installation options for more information.

Note: You must specify the fully-qualified response file name. Specifying the relative path causes the installation to fail with no indication that an error occurred.

Procedure

1. Optional: If you choose to install the CIP or IIP using a response file, first customize the file.
 - a. Copy the response file, `wxssetup.response.txt`, from the product DVD to your disk drive.
 - b. Open and edit the response file in the text editor of your choice. The file includes comments to assist the configuration process and must include these parameters:
 - The license agreement
 - The location of the product installation

Tip: The installer uses the location that you select for your installation to determine where your WebSphere Application Server instance is installed. If you install on a node with multiple WebSphere Application Server instances, clearly define your location.

c. Run the following script to start your customized response file.

- `Linux` `UNIX` `install -options /absolute_path/response_file.txt -silent`
- `Windows` `install.bat -options C:\drive_path\response_file.txt -silent`

2. Optional: If you choose to install the CIP or IIP by passing certain parameters to the command line, run the following script to start the installation:

- `Linux` `UNIX` `install -silent -OPT silentInstallLicenseAcceptance=true -OPT installLocation=install_location`
- `Windows` `install.bat -silent -OPT silentInstallLicenseAcceptance=true -OPT installLocation=install_location`

where `install_location` is the location of your existing WebSphere Application Server installation.

3. Review the resulting logs for errors or an installation failure.

Results

You silently installed the CIP or IIP.

What to do next

If you are running WebSphere Application Server Version 6.1 or Version 7.0, you can use the Profile Management Tool plug-in or the `manageprofiles` command to create and augment profiles. If you are running WebSphere Application Server Version 6.0.2, you must use the `wasprofile` command to create and augment profiles.

If you augmented profiles for eXtreme Scale during the installation process, you can deploy applications, start a catalog service, and start the containers in your WebSphere Application Server environment. See “Configuring WebSphere eXtreme Scale with WebSphere Application Server” on page 205 for more information.

wxssetup.response.txt file:

You can use a fully qualified response file to install WebSphere eXtreme Scale or WebSphere eXtreme Scale Client silently.

CAUTION:

Do not add trailing slashes, such as / or \, to the end of the installation location paths. These paths are specified with the `installLocation` attribute. Adding a slash to the end of the installation location can cause the installation to fail. For example, the following path would cause the installation to fail:

```
-OPT installLocation="/usr/IBM/WebSphere/eXtremeScale/"
```

The path should be specified as:

```
-OPT installLocation="/usr/IBM/WebSphere/eXtremeScale"
```

Response file for WebSphere eXtreme Scale full installation

```
#####  
#  
# IBM WebSphere eXtreme Scale V7.1.0 InstallShield Options File  
#
```

```

# Wizard name: Install
# Wizard source: setup.jar
#
# This file can be used to configure Install with the options specified below
# when the wizard is run with the "-options" command line option. Read each
# setting's documentation for information on how to change its value.
# Please enclose all values within a single pair of double quotes.
#
# A common use of an options file is to run the wizard in silent mode. This lets
# the options file author specify wizard settings without having to run the
# wizard in graphical or console mode. To use this options file for silent mode
# execution, use the following command line arguments when running the wizard:
#
#   -options "D:\installImage\WXS\wxssetup.response" -silent
#
# Note that the fully qualified response file name must be used.
#
#####

#####

#
# License Acceptance
#
# Valid Values:
# true - Accepts the license. Will install the product.
# false - Declines the license. Install will not occur.
#
# If no install occurs, this will be logged to a temporary log file in the
# user's temporary directory.
#
# By changing the silentInstallLicenseAcceptance property in this response file
# to "true", you agree that you have reviewed and agree to the terms of the
# IBM International Program License Agreement accompanying this program, which
# is located at CD_ROOT\XD\wxs.primary.pak\repository\legal.xs\license.xs. If
# you do not agree to these terms, do not change the value or otherwise
# download, install, copy, access, or use the program and promptly return the
# program and proof of entitlement to the party from whom you acquired it to
# obtain a refund of the amount you paid.
#
-OPT silentInstallLicenseAcceptance="false"

#####

# Non-blocking Prerequisite Checking
#
# If you want to disable non-blocking prerequisite checking, uncomment
# the following line. This will notify the installer to continue with
# the installation and log the warnings even though the prerequisite checking
# has failed.
#
-OPT disableNonBlockingPrereqChecking="true"

#####

#
# Install Location
#
# The install location of the product. Specify a valid directory into which the
# product should be installed. If the directory contains spaces, enclose it in
# double-quotes as shown in the Windows example below. Note that spaces in the
# install location is only supported on Windows operating systems. Maximum path
# length is 60 characters for Windows.
#
# Below is the list of default install locations for each supported operating
# system when you're installing as a root user. By default, in this response
# file, the Windows install location is used. If you want to use the default
# install location for another operating system, uncomment the appropriate

```

```

# default install location entry (by removing '#') and then comment out
# (by adding '#') the Windows operating system entry below.
#
# The install location is used to determine if WebSphere eXtreme Scale should
# be installed as a stand-alone deployment or if it should be integrated with
# an existing WebSphere Application Server installation.
#
# If the location specified is an existing WebSphere Application Server or
# WebSphere Network Deployment installation, then eXtreme Scale is integrated
# with the existing WebSphere Application Server. If the location specified is
# a new or empty directory, then WebSphere eXtreme Scale is installed as a
# stand-alone deployment.
#
# Note: If the install location specified contains a previous installation of
# WebSphere eXtreme Scale, WebSphere eXtended Deployment DataGrid or
# ObjectGrid, the installation will fail.
#
# AIX Default Install Location:
#
# -OPT installLocation="/usr/IBM/WebSphere/eXtremeScale"
#
# HP-UX, Solaris or Linux Default Install Location:
#
# -OPT installLocation="/opt/IBM/WebSphere/eXtremeScale"
#
# Windows Default Install Location:
#
-OPT installLocation="C:\Program Files\IBM\WebSphere\eXtremeScale"

#
# If you are installing as a non-root user on Unix or a non-administrator on
# Windows, the following default install locations are suggested. Be sure you
# have write permission for the install location chosen.
#
# AIX Default Install Location:
#
# -OPT installLocation="<user's home>/IBM/WebSphere/eXtremeScale"
#
# HP-UX, Solaris or Linux Default Install Location:
#
# -OPT installLocation="<user's home>/IBM/WebSphere/eXtremeScale"
#
# Windows Default Install Location:
#
-OPT installLocation="C:\IBM\WebSphere\eXtremeScale"

#####
# Optional Features Installation
#
# Specify which of the optional features you wish to install by setting each
# desired feature to "true". Set any optional features you do not want to
# install to "false".
#
# The options selectServer, selectClient, selectPF, and selectXSStreamQuery are
# only valid when the installLocation option above contains an installation of
# WebSphere Application Server. The options are ignored on an WebSphere eXtreme
# Scale standalone installation.
#
# On the WebSphere eXtreme Scale standalone installation, the eXtreme Scale
# server and client are automatically installed. The feature options for the
# eXtreme Scale standalone installation are selectXSConsoleOther and
# selectXSStreamQueryOther.

#
# This option, when selected, installs the components that are required to run

```



```

# WebSphere eXtreme Scale servers and the eXtreme Scale dynamic cache service
# provider. If this option is selected, then the WebSphere eXtreme Scale Client
# must also be selected by being uncommented and set to a value of "true".
# Otherwise, silent install will FAIL.
#
-OPT selectServer="true"

#
# This option, when selected, installs the components that are required to run
# WebSphere eXtreme Scale client applications. If the Server option is selected
# above, then this option must also be selected by being uncommented and set to
# a value of "true" or silent install will FAIL.
#
-OPT selectClient="true"

#
# This option, when selected, installs the components that are required to run
# the WebSphere eXtreme Scale Console. If this option is selected, the install
# location specified above must be a new or empty directory because the console
# option is only valid for WebSphere eXtreme Scale stand-alone deployment. To
# install this option, the following option line must be uncommented and set
# to a value of "true".
#-OPT selectXSConsoleOther="false"

#
# The following options, if selected will install DEPRECATED functionality.
#
# This option selects WebSphere Partition Facility for installation.
# This functionality is DEPRECATED. To install this option, the following
# option line must be uncommented and set to a value of "true".
#
#-OPT selectPF="false"

#
# This option selects WebSphere eXtreme Scale StreamQuery for WAS for
# installation. This functionality is DEPRECATED. To install this option,
# the following option line must be uncommented and set to a value of "true".
# If this option is selected, then the WebSphere eXtreme Scale Client
# must also be selected by being uncommented and set to a value of "true".
# Otherwise, silent install will FAIL.
#
#-OPT selectXSStreamQuery="false"

#
# This option selects WebSphere eXtreme Scale StreamQuery for J2SE for
# installation. This functionality is DEPRECATED. To install this option,
# the following option line must be uncommented and set to a value of "true".
# If this option is selected, then the WebSphere eXtreme Scale Client
# must also be selected by being uncommented and set to a value of "true".
# Otherwise, silent install will FAIL.
#
#-OPT selectXSStreamQueryOther="false"

#####
# Profile list for augmentation
#
# Specify which of the existing profiles you wish to augment or comment the
# line to augment every existing profiles detected by the intallation.
#
# To specify multiple profiles, use comma to separate different profile names.
# For example, "AppSrv01,Dmgr01,Custom01". The list must not contain any spaces.
#
-OPT profileAugmentList=""

#####

```

```

# Tracing Control
#
# The trace output format can be controlled via the option
# -OPT traceFormat=ALL
#
# The choices for the format are 'text' and 'XML'. By default, both formats will
# be produced, in two different trace files.
#
# If only one format is required, use the traceFormat option to specify which
# one, as follows:
#
# Valid Values:
#
# text - Lines in the trace file will be in a plain text format for easy
#        readability.
# XML - Lines in the trace file will be in the standard Java logging XML
#        format which can be viewed using any text or XML editor or using the
#        Chainsaw tool from Apache at the following URL:
#        (http://logging.apache.org/log4j/docs/chainsaw.html).
#
# The amount of trace info captured can be controlled using the option:
# -OPT traceLevel=INFO
#
# Valid Values:
#
# Trace   Numerical
# Level   Level   Description
# -----
# OFF     0       No trace file is produced
# SEVERE  1       Only severe errors are output to trace file
# WARNING 2       Messages regarding non-fatal exceptions and warnings are
#             added to trace file
# INFO    3       Informational messages are added to the trace file
#             (this is the default trace level)
# CONFIG  4       Configuration related messages are added to the trace file
# FINE    5       Tracing method calls for public methods
# FINER   6       Tracing method calls for non public methods except
#             getters and setters
# FINEST  7       Trace all method calls, trace entry/exit will include
#             parameters and return value

```

Response file for WebSphere eXtreme Scale Client installation

```

#####
#
# IBM WebSphere eXtreme Scale Client V7.1.0 InstallShield Options File
#
# Wizard name: Install
# Wizard source: setup.jar
#
# This file can be used to configure Install with the options specified below
# when the wizard is run with the "-options" command line option. Read each
# setting's documentation for information on how to change its value.
# Please enclose all values within a single pair of double quotes.
#
# A common use of an options file is to run the wizard in silent mode. This lets
# the options file author specify wizard settings without having to run the
# wizard in graphical or console mode. To use this options file for silent mode
# execution, use the following command line arguments when running the wizard:
#
#   -options "D:\installImage\WXS_Client\wxsetup.response" -silent
#
# Note that the fully qualified response file name must be used.
#
#####
#####

```

```

#
# License Acceptance
#
# Valid Values:
# true - Accepts the license. Will install the product.
# false - Declines the license. Install will not occur.
#
# If no install occurs, this will be logged to a temporary log file in the
# user's temporary directory.
#
# By changing the silentInstallLicenseAcceptance property in this response file
# to "true", you agree that you have reviewed and agree to the terms of the
# IBM International Program License Agreement accompanying this program, which
# is located at
# CD_ROOT\WXS_Cleint\wxs.client.primary.pak\repository\legal.xs.client\license.xs.
# If you do not agree to these terms, do not change the value or otherwise
# download, install, copy, access, or use the program and promptly return the
# program and proof of entitlement to the party from whom you acquired it to
# obtain a refund of the amount you paid.
#
-OPT silentInstallLicenseAcceptance="false"

#####
# Non-blocking Prerequisite Checking
#
# If you want to disable non-blocking prerequisite checking, uncomment
# the following line. This will notify the installer to continue with
# the installation and log the warnings even though the prerequisite checking
# has failed.
#
-OPT disableNonBlockingPrereqChecking="true"

#####
#
# Install Location
#
# The install location of the product. Specify a valid directory into which the
# product should be installed. If the directory contains spaces, enclose it in
# double-quotes as shown in the Windows example below. Note that spaces in the
# install location is only supported on Windows operating systems. Maximum path
# length is 60 characters for Windows.
#
# Below is the list of default install locations for each supported operating
# system when you're installing as a root user. By default, in this response
# file, the Windows install location is used. If you want to use the default
# install location for another operating system, uncomment the appropriate
# default install location entry (by removing '#') and then comment out
# (by adding '#') the Windows operating system entry below.
#
# The install location is used to determine if WebSphere eXtreme Scale should
# be installed as a stand-alone deployment or if it should be integrated with
# an existing WebSphere Application Server installation.
#
# If the location specified is an existing WebSphere Application Server or
# WebSphere Network Deployment installation, then eXtreme Scale is integrated
# with the existing WebSphere Application Server. If the location specified is
# a new or empty directory, then WebSphere eXtreme Scale is installed as a
# stand-alone deployment.
#
# Note: If the install location specified contains a previous installation of
# WebSphere eXtreme Scale, WebSphere eXtended Deployment DataGrid or
# ObjectGrid, the installation will fail.
#
# AIX Default Install Location:
#

```

```

# -OPT installLocation="/usr/IBM/WebSphere/eXtremeScale"
#
# HP-UX, Solaris or Linux Default Install Location:
#
# -OPT installLocation="/opt/IBM/WebSphere/eXtremeScale"
#
#
# Windows Default Install Location:
#
-OPT installLocation="C:\Program Files\IBM\WebSphere\eXtremeScale"

#
# If you are installing as a non-root user on Unix or a non-administrator on
# Windows, the following default install locations are suggested. Be sure you
# have write permission for the install location chosen.
#
# AIX Default Install Location:
#
# -OPT installLocation="/IBM/WebSphere/eXtremeScale"
#
# HP-UX, Solaris or Linux Default Install Location:
#
# -OPT installLocation="/IBM/WebSphere/eXtremeScale"
#
# Windows Default Install Location:
#
# -OPT installLocation="C:\IBM\WebSphere\eXtremeScale"

#####
# Profile list for augmentation
#
# Specify which of the existing profiles you wish to augment or comment the
# line to augment every existing profiles detected by the intallation.
#
# To specify multiple profiles, use comma to separate different profile names.
# For example, "AppSrv01,Dmgr01,Custom01". The list must not contain any spaces.
#
-OPT profileAugmentList=""

#####
# Tracing Control
#
# The trace output format can be controlled via the option
# -OPT traceFormat=ALL
#
# The choices for the format are 'text' and 'XML'. By default, both formats will
# be produced, in two different trace files.
#
# If only one format is required, use the traceFormat option to specify which
# one, as follows:
#
# Valid Values:
#
# text - Lines in the trace file will be in a plain text format for easy
#        readability.
# XML - Lines in the trace file will be in the standard Java logging XML
#        format which can be viewed using any text or XML editor or using the
#        Chainsaw tool from Apache at the following URL:
#        (http://logging.apache.org/log4j/docs/chainsaw.html).
#
# The amount of trace info captured can be controlled using the option:
# -OPT traceLevel=INFO
#
# Valid Values:
#

```

# Trace Level	Numerical Level	Description
# OFF	0	No trace file is produced
# SEVERE	1	Only severe errors are output to trace file
# WARNING	2	Messages regarding non-fatal exceptions and warnings are added to trace file
# INFO	3	Informational messages are added to the trace file (this is the default trace level)
# CONFIG	4	Configuration related messages are added to the trace file
# FINE	5	Tracing method calls for public methods
# FINER	6	Tracing method calls for non public methods except getters and setters
# FINEST	7	Trace all method calls, trace entry/exit will include parameters and return value

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 “Installing WebSphere eXtreme Scale or WebSphere eXtreme Scale Client with WebSphere Application Server” on page 26 for more information.

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 using the **startOgServer** scripts.
- 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.

About this task

Running within WebSphere Application Server Version 6.0.2

If your environment contains WebSphere Application Server Version 6.0.2, use the **wasprofile** command to create or augment profiles for WebSphere eXtreme Scale as shown in the following example:

```
was_root/bin/wasprofile.sh|bat -augment -profileName dmgr_01
-templatePath "C:/ProgramFiles/IBM/WebSphere/AppServer/profileTemplates/xs_augment/dmgr"
```

See the **wasprofile** command in the WebSphere Application Server Information Center for more information.

Running within WebSphere Application Server Version 6.1 or Version 7.0

If your environment contains WebSphere Application Server Version 6.1 or Version 7.0, you can use the Profile Management Tool plug-in or the **manageprofiles** command to create and augment profiles.

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 *wxs_install_root*\firststeps\wxs\firststeps.bat directory. 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 scenarios:

- **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.
- **WebSphere Application Server Version 6.0.2:**
If you are running WebSphere Application Server Version 6.0.2 or WebSphere Application Server Network Deployment Version 6.0.2, you must use the **wasprofile** command to create or augment a profile for WebSphere eXtreme Scale as shown in the following example:

```
was_root/bin/wasprofile.sh|bat -create -profileName dmgr_01  
-templatePath "C:/ProgramFiles/IBM/WebSphere/AppServer/profileTemplates/xs_augment/dmgr"
```

See the wasprofile command in the WebSphere Application Server Version 6.0 Information Center for more information.

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” on page 205 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.

Before you begin

Note: If you are running WebSphere Application Server Version 6.0.2 or WebSphere Application Server Network Deployment Version 6.0.2, you must use the **wasprofile** command to create or augment a profile for WebSphere eXtreme Scale as shown in the following example:

```
was_root/bin/wasprofile.sh|bat -augment -profileName dmgr_01  
-templatePath "C:/ProgramFiles/IBM/WebSphere/AppServer/profileTemplates/xs_augment/dmgr"
```

See the **wasprofile** command in the WebSphere Application Server Information Center for more information.

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.
- Augment the profile with the **pf_augment** template only if you installed the partitioning facility.
- Apply both of the templates if your environment contains the eXtreme Scale client and the partitioning facility.

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” on page 205 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” on page 26 for more information.
- If your environment contains WebSphere Application Server Version 6.0.2, you must use the **wasprofile** command to create and augment profiles for eXtreme Scale as shown in the following example:

```
install_root/bin/wasprofile.sh|bat -augment -profileName dmgr_01  
-templatePath "C:/ProgramFiles/IBM/WebSphere/AppServer/profileTemplates/xs_augment/dmgr"
```

See the **wasprofile** command in the WebSphere Application Server Information Center 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.
- Augment the profile with the `pf_augment` template if you installed only the partitioning facility.
- Apply both templates if your environment contains the eXtreme Scale client and the partitioning facility.

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
```

where *template_type* is *xs_augment* or *pf_augment*.

Example

- Using the *xs_augment* template:

```
./manageprofiles.sh|bat -create -templatePath install_root/profileTemplates/xs_augment/dmgr
```

- Using the *pf_augment* template:

```
./manageprofiles.sh|bat -create -templatePath install_root/profileTemplates/pf_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
```

where *template_type* is *xs_augment* or *pf_augment*.

Example

- Using the *xs_augment* template:

```
./manageprofiles.sh|bat -create -templatePath install_root/profileTemplates/xs_augment/managed
```

- Using the *pf_augment* template:

```
./manageprofiles.sh|bat -create -templatePath install_root/profileTemplates/pf_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
```

where *template_type* is *xs_augment* or *pf_augment*.

Example

- Using the *xs_augment* template:

```
./manageprofiles.sh|bat -create -templatePath install_root/profileTemplates/xs_augment/default
```

- Using the *pf_augment* template:

```
./manageprofiles.sh|bat -create -templatePath install_root/profileTemplates/pf_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
```

where *template_type* is *xs_augment* or *pf_augment*.

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
```

where *template_type* is *xs_augment* or *pf_augment*.

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
```

- Using the *pf_augment* template:

```
./manageprofiles.sh|bat -create -templatePath install_root/profileTemplates/pf_augment/cell/dmgr  
-nodeProfilePath install_root/profiles/AppSrv01 -cellName cell01dmgr -nodeName node01dmgr  
-appServerNodeName node01
```

```
./manageprofiles.sh|bat -create -templatePath install_root/profileTemplates/pf_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
```

where *template_type* is *xs_augment* or *pf_augment*.

Example

- Using the *xs_augment* template:

```
./manageprofiles.sh|bat -augment -profileName profile01  
-templatePath install_root/profileTemplates/xs_augment/dmgr
```

- Using the *pf_augment* template:

```
./manageprofiles.sh|bat -augment -profileName profile01  
-templatePath install_root/profileTemplates/pf_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
```

where *template_type* is *xs_augment* or *pf_augment*.

Example

- Using the *xs_augment* template:

```
./manageprofiles.sh|bat -augment -profileName profile01 -templatePath install_root  
/profileTemplates/xs_augment/managed
```

- Using the *pf_augment* template:

```
./manageprofiles.sh|bat -augment -profileName profile01 -templatePath install_root  
/profileTemplates/pf_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
```

where *template_type* is *xs_augment* or *pf_augment*.

Example

- Using the *xs_augment* template:

```
./manageprofiles.sh|bat -augment -profileName profile01 -templatePath install_root  
/profileTemplates/xs_augment/default
```

- Using the *pf_augment* template:

```
./manageprofiles.sh|bat -augment -profileName profile01 -templatePath install_root  
/profileTemplates/pf_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
```

where *template_type* is *xs_augment* or *pf_augment*.

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
```

where *template_type* is *xs_augment* or *pf_augment*.

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
```

- Using the *pf_augment* template:

```
./manageprofiles.sh|bat -augment -profileName profile01 -templatePath install_root  
/profileTemplates/pf_augment/cell/dmgr
```

```
./manageprofiles.sh|bat -augment -profileName profile01 -templatePath install_root  
/profileTemplates/pf_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
```

- Using the `pf_augment` template:

```
./manageprofiles.sh|bat -unaugment -profileName profile01 -ignoreStack
-templatePath install_root/profileTemplates/pf_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 environment, one outside of 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.

Installing WebSphere eXtreme Scale or WebSphere eXtreme Scale Client silently

Use a fully qualified response file, which you configure specifically to your needs, or pass parameters to the command line to silently install WebSphere eXtreme Scale or WebSphere eXtreme Scale Client.

Before you begin

- 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:

Ensure that any running processes are stopped. If the running processes are not stopped, the installation proceeds, creating unpredictable results and leaving the installation in an undetermined state on some platforms.

- Verify that the target installation directory is empty or does not exist.

Important: If a previous version of WebSphere eXtreme Scale or the ObjectGrid component exists in the directory that you specify to install Version 7.1, the product is not installed. For example, you might have a previously existing *wxs_install_root/ObjectGrid* folder. You can either select a different installation directory or cancel the installation. Next, uninstall the previous installation and run the wizard again.

About this task

A silent installation uses the same installation program that the graphical user interface (GUI) version uses. However, instead of displaying a wizard interface, the silent installation reads all of your responses from a file that you customize, or from parameters that you pass to the command line. See an example of a “wxssetup.response.txt file” on page 38, which includes a description of each option.

Procedure

1. Optional: If you choose to install WebSphere eXtreme Scale or WebSphere eXtreme Scale Client using a response file, first customize the `wxssetup.response.txt` file.

Remember: You must specify the fully-qualified response file name. Specifying the relative path causes the installation to fail with no indication that an error occurred.

- a. Make a copy of the response file to customize.

For the WebSphere eXtreme Scale full installation, copy the response file from the product DVD to your disk drive.

For the WebSphere eXtreme Scale Client, unzip the WebSphere eXtreme Scale Client zip file onto your hard drive and find the response file.

- b. Open and edit the response file in the text editor of your choice. The previous example response file provides details on how to specify each of the parameters. You must specify the following parameters:
 - The license agreement
 - The installation directory

Tip: When you install WebSphere eXtreme Scale or WebSphere eXtreme Scale Client in a WebSphere Application Server environment, the installer uses the installation directory to determine where the existing WebSphere Application Server instance is installed. If you install on a node that contains multiple WebSphere Application Server instances, clearly define your location.

- c. Run the following script to start the installation.

For the WebSphere eXtreme Scale full installation:

```
./install.sh|bat -options C:/drive_path/response_file.txt -silent
```

For the WebSphere eXtreme Scale Client installation:

```
./WXS_Client/install.sh|bat -options C:/drive_path/response_file.txt -silent
```

You can also use the response file when you run a GUI installation. You can use the response file with a GUI installation to debug problems that are hidden with the silent installation. When you specify the `wxssetup.response` file for GUI or silent installations, you must use the fully qualified path. Run the following script to run the GUI installation with your response file:

- `Linux` `UNIX` `<install_home>/install.sh -options <full_install_path_required>/wxssetup.response`
- `Windows` `<install_home>\install.exe -options c:\<full_install_path_required>\wxssetup.response`

2. Optional: If you choose to install eXtreme Scale by passing certain parameters to the command line, run the following script to start the installation:

For the WebSphere eXtreme Scale full installation:

```
./install.sh|bat -silent -OPT silentInstallLicenseAcceptance=true -OPT installLocation=install_location
```

For the WebSphere eXtreme Scale Client installation:

```
./WXS_Client/install.sh|bat -silent -OPT silentInstallLicenseAcceptance=true -OPT installLocation=install_location
```

Installation parameters

Specify parameters at the command line to customize and configure your product installation.

Note: You must specify the fully-qualified response file name. Specifying the relative path causes the installation to fail with no indication that an error occurred.

Parameters

You can pass the following parameters during a command-line or options file installation of the product:

-silent

Suppresses the graphical user interface (GUI). Specify the **-options** parameter to indicate that the installer completes the installation according to a customized options file. If you do not specify the **-options** parameter, the default values are used instead.

Example usage


```
./install.sh|bat -silent -options options_file.txt
```

-options *path_name/file_name*
Specifies an options file that the installer uses to complete a silent installation. Properties on the command line take precedence.

Example usage
./install.sh|bat -options c:/path_name/options_file.txt

-log # !file_name @event_type
Generates an installation log file that logs the following event types:

- err
- wrn
- msg1
- msg2
- dbg
- ALL

Example usage
./install.sh|bat -log # !c:/temp/logfiles.txt @ALL

-is:log *path_name/file_name*
Creates a log file that contains the Java Virtual Machine (JVM) searches of the installer while attempting to start the GUI. The log file is not created unless specified.

Example usage
./install.sh|bat -is:log c:/logs/javalog.txt

-is:javaconsole
Displays a console window during the installation process.

Example usage
./install.sh|bat -is:javaconsole

-is:silent
Suppresses the Java initialization window that is displayed as the installer starts.

Example usage
./install.sh|bat -is:silent

-is:tempdir *path_name*
Specifies the temporary directory that the installer uses during the installation.

Example usage
./install.sh|bat -is:tempdir c:/temp

Customizing WebSphere eXtreme Scale for z/OS

Using the WebSphere Customization Tools, 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 6.1, your system must contain fix pack 31 at a minimum. See Installing your Version 6.1 application serving environment for more information.

- If you are running Version 7.0, your system must contain fix pack 9 at a minimum. See *Installing your Version 7.0 application serving environment* for more information.
- Install WebSphere eXtreme Scale for z/OS. See the *WebSphere eXtreme Scale WebSphere eXtreme Scale Program Directory* on the Library Page for more information.

About this task

Using the WebSphere Customization Tools, generate customization definitions and upload and run customized jobs to customize WebSphere eXtreme Scale for z/OS. See the following topics for more information:

Procedure

- “Installing the WebSphere Customization Tools”
- “Generating customization definitions” on page 59
- “Uploading and running customized jobs” on page 60

Installing the WebSphere Customization Tools

Install the WebSphere Customization Tools Version 7.0.0.6 or later 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.

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` and `xspf.wct` extension files 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/zWebSphereXS/util/V7R1/WCT` directory on your z/OS system.
2. Download and install the WebSphere Customization Tools Version 7.0.0.6 or later from the appropriate Web site:
 -  WebSphere Customization Tools for Windows
 -  WebSphere Customization Tools for Linux
3. Upload the `xs.wct` file to the WebSphere Customization Tools application.
 - a. Start the WebSphere Customization Tools application on your workstation.
 - b. Click **Help > Software Updates > Install Extension**.
 - c. From the WebSphere Customization Tools Extension Locations panel, click **Install new extension location**.
 - 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 Summary panel.

Note: The Install Successful panel is displayed. Before you can click **Finish**, you must copy and save the data from the location field:

```
C:\Documents and Settings\Administrator\WCT\workspace\configuration\  
com.ibm.ws.pmt.update\com.ibm.ws390.pmt.xs_7.1.0.0\ eclipse
```

- f. From the Product Configuration panel, click **Add an extension location**. Paste the data you copied in the previous step in the Location field, and click **OK**.
 - g. Click **Yes** to restart the WebSphere Customization Tools.
4. Upload the xspf.wct file to the WebSphere Customization Tools application.
 - a. Click **Help > Software Updates > Install Extension**.
 - b. From the WebSphere Customization Tools Extension Locations panel, click **Install new extension location**.
 - c. 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**.
 - d. Click **Next** on the Summary panel.

Note: The Install Successful panel is displayed. Before you can click **Finish**, you must copy and save the data from the location field:

```
C:\Documents and Settings\Administrator\WCT\workspace\configuration\  
com.ibm.ws.pmt.update\com.ibm.ws390.pmt.xs_7.1.0.0\ eclipse
```

- e. From the Product Configuration panel, click **Add an extension location**. Paste the data you copied in the previous step in the Location field, and click **OK**.
- f. Click **Yes** to restart the WebSphere Customization Tools.

What to do next

After you upload both extension files and restart the WebSphere Customization Tools, you can use the Profile Management Tool to generate customization definitions for 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 Tools to generate customization definitions and create customized jobs for WebSphere eXtreme Scale for z/OS.

Before you begin

Install the WebSphere Customization Tools and upload the xs.wct and xspf.wct extension files. See “Installing the WebSphere Customization Tools” on page 58 for more information.

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 Tools**. After the application starts, click the **Profile Management Tool** tab.
 - **Linux** Click **operating_system_menus > IBM WebSphere > WebSphere Customization Tools**. After the application starts, click the **Profile Management Tool** tab.
2. Add an existing location or create a location of the customization definition that you want to create. On the **Customization Locations** tab, click **Add**. If you create a location, the Version box refers to the existing WebSphere Application Server product version installed on your z/OS system.

Note: Do not use the same location you are using for other eXtreme Scale customization definitions.

3. Generate the customization definition. On the **Customization Definitions** tab, click **Augment**.
4. Select the type of definition environment to create:
 - Stand-alone application server node
 - Deployment manager
 - Application server
 - Managed (custom) node
5. Complete the fields on the panels. Specify the values for the parameters that are used to create your z/OS system.
6. Click **Augment** to generate the customization definition.

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 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. See “Generating customization definitions” on page 59 for more information.

About this task

Upload and run the customized jobs 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**.
2. Upload the jobs to the FTP server on your z/OS system. 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.

Uninstalling WebSphere eXtreme Scale

To remove WebSphere eXtreme Scale from your environment, you can use the wizard or you can silently uninstall the product.

Before you begin

Attention: The uninstaller removes all binary files and all maintenance, such as fix packs and interim fixes, at the same time.

Procedure

1. Stop all processes that are running eXtreme Scale.

CAUTION:

Ensure that any running processes are stopped. If the running processes are not stopped, the uninstallation proceeds, creating unpredictable results and leaving the uninstallation in an undetermined state on some platforms.

- If you installed stand-alone eXtreme Scale, read about stopping stand-alone servers to stop processes.
 - If you installed eXtreme Scale with an existing installation of WebSphere Application Server, read about command-line utilities for more information about stopping WebSphere Application Server processes.
2. Uninstall the product. You can run the uninstallation in a GUI or silently.

Note: When specifying the responsefile `wxssetup.response` file for silent or GUI uninstall or installations, the fully qualified path must always be specified. The responsefile is optional for the GUI uninstallation.

- **To run the uninstallation using the GUI:**

- `Linux` `UNIX` `<install_home>/uninstall_wxs/uninstall`
- `Windows` `<install_home>\uninstall_wxs\uninstall.exe`

If you want to run the uninstallation using the GUI and the `wxssetup.response` file, use one of the following commands:

- `Linux` `UNIX`
`<install_home>/uninstall_wxs/uninstall -options
<full_install_path_required>/wxssetup.response`

- `Windows`
`<install_home>\uninstall_wxs\uninstall.exe -options
<full_install_path_required>\wxssetup.response`

- **To run the uninstallation silently using the responsefile `wxssetup.response` script:**

- `Linux` `UNIX`
`<install_home>/uninstall_wxs/uninstall -options
<full_install_path_required>/wxssetup.response -silent`
- `Windows`
`<install_home>\uninstall_wxs\uninstall.exe -options
<full_install_path_required>\wxssetup.response -silent`

Results

You removed eXtreme Scale from your environment.

Chapter 4. Customizing WebSphere eXtreme Scale for z/OS

Using the WebSphere Customization Tools, 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 6.1, your system must contain fix pack 31 at a minimum. See *Installing your Version 6.1 application serving environment* for more information.
 - If you are running Version 7.0, your system must contain fix pack 9 at a minimum. See *Installing your Version 7.0 application serving environment* for more information.
- Install WebSphere eXtreme Scale for z/OS. See the *WebSphere eXtreme Scale WebSphere eXtreme Scale Program Directory* on the Library Page for more information.

About this task

Using the WebSphere Customization Tools, generate customization definitions and upload and run customized jobs to customize WebSphere eXtreme Scale for z/OS. See the following topics for more information:

Procedure

- “Installing the WebSphere Customization Tools” on page 58
- “Generating customization definitions” on page 59
- “Uploading and running customized jobs” on page 60

Installing the WebSphere Customization Tools

Install the WebSphere Customization Tools Version 7.0.0.6 or later 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.

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` and `xspf.wct` extension files 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/zWebSphereXS/util/V7R1/WCT` directory on your z/OS system.

2. Download and install the WebSphere Customization Tools Version 7.0.0.6 or later from the appropriate Web site:
 -  WebSphere Customization Tools for Windows
 -  WebSphere Customization Tools for Linux
3. Upload the xs.wct file to the WebSphere Customization Tools application.
 - a. Start the WebSphere Customization Tools application on your workstation.
 - b. Click **Help > Software Updates > Install Extension**.
 - c. From the WebSphere Customization Tools Extension Locations panel, click **Install new extension location**.
 - 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 Summary panel.

Note: The Install Successful panel is displayed. Before you can click **Finish**, you must copy and save the data from the location field:

```
C:\Documents and Settings\Administrator\WCT\workspace\configuration\
com.ibm.ws.pmt.update\com.ibm.ws390.pmt.xs_7.1.0.0\ eclipse
```

- f. From the Product Configuration panel, click **Add an extension location**. Paste the data you copied in the previous step in the Location field, and click **OK**.
 - g. Click **Yes** to restart the WebSphere Customization Tools.
4. Upload the xspf.wct file to the WebSphere Customization Tools application.
 - a. Click **Help > Software Updates > Install Extension**.
 - b. From the WebSphere Customization Tools Extension Locations panel, click **Install new extension location**.
 - c. 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**.
 - d. Click **Next** on the Summary panel.

Note: The Install Successful panel is displayed. Before you can click **Finish**, you must copy and save the data from the location field:

```
C:\Documents and Settings\Administrator\WCT\workspace\configuration\
com.ibm.ws.pmt.update\com.ibm.ws390.pmt.xs_7.1.0.0\ eclipse
```

- e. From the Product Configuration panel, click **Add an extension location**. Paste the data you copied in the previous step in the Location field, and click **OK**.
 - f. Click **Yes** to restart the WebSphere Customization Tools.

What to do next

After you upload both extension files and restart the WebSphere Customization Tools, you can use the Profile Management Tool to generate customization definitions for eXtreme Scale for z/OS. See “Generating customization definitions” on page 59 for more information.

Generating customization definitions

Use the Profile Management Tool function within the WebSphere Customization Tools to generate customization definitions and create customized jobs for WebSphere eXtreme Scale for z/OS.

Before you begin

Install the WebSphere Customization Tools and upload the `xs.wct` and `xspf.wct` extension files. See “Installing the WebSphere Customization Tools” on page 58 for more information.

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 Tools**. After the application starts, click the **Profile Management Tool** tab.
 - **Linux** Click **operating_system_menus > IBM WebSphere > WebSphere Customization Tools**. After the application starts, click the **Profile Management Tool** tab.
2. Add an existing location or create a location of the customization definition that you want to create. On the **Customization Locations** tab, click **Add**. If you create a location, the Version box refers to the existing WebSphere Application Server product version installed on your z/OS system.

Note: Do not use the same location you are using for other eXtreme Scale customization definitions.

3. Generate the customization definition. On the **Customization Definitions** tab, click **Augment**.
4. Select the type of definition environment to create:
 - Stand-alone application server node
 - Deployment manager
 - Application server
 - Managed (custom) node
5. Complete the fields on the panels. Specify the values for the parameters that are used to create your z/OS system.
6. Click **Augment** to generate the customization definition.

What to do next

Upload the customized job to your target z/OS system. See “Uploading and running customized jobs” on page 60 for more information.

Uploading and running customized jobs

After you generate the customization definitions, you can upload and run the customized jobs 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. See “Generating customization definitions” on page 59 for more information.

About this task

Upload and run the customized jobs 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**.
2. Upload the jobs to the FTP server on your z/OS system. 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.

Chapter 5. Upgrading and migrating WebSphere eXtreme Scale Version 7.1

You can migrate to WebSphere eXtreme Scale Version 7.1 from previous versions, and you can apply maintenance packages to WebSphere eXtreme Scale Version 7.1. 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.

Before you begin

You must have the binary 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, upgrade your catalog servers first. Then, upgrade the container servers and the clients.

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. Check for a healthy quorum status. Run the following command:

```
xsadmin -quorumstatus
Connecting to Catalog service at localhost:1099
Quorum is enabled and catalog server is in normal condition
```

This result indicates that all the catalog servers are connected. The results of the **xsadmin -quorumstatus** command also might display the following message:

```
Quorum is enabled but quorum is overridden.
```

With this message, quorum is healthy, but not all catalog servers are running.

- 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 dname
```

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 **stop0gserver** command, the **xsadmin -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, 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 **xsadmin -quorumstatus** command.

If you use the **xsadmin -teardown** command, you can filter the server names. The **stopOgServer** command requires an exact server name or list of server names to stop in parallel to be entered. You should group the shutdown process instead of calling the stop or teardown process for many servers in parallel. By grouping the servers to be shut down, the data grid can react to the servers that are being shut down by moving shards around the data grid. You can use one of the following commands to shut down your servers:

You can provide a specific list of servers to stop to the **stopOgServer** or **xsadmin -teardown** commands:

```
stopOgServer <server_name>[,<server_name>]
```

```
xsadmin -teardown <server_name>[,<server_name>]
```

With the previous examples, the **stopOgServer** or **xsadmin -teardown** commands are completing the same shutdown tasks. However, you can filter the servers to stop with the **xsadmin -teardown** command. See “Stopping servers gracefully with the xsadmin tool” on page 363 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:
 - “Migrating to WebSphere eXtreme Scale Version 7.1” on page 69
 - “Using the Update Installer to install maintenance packages” on page 70

- e. Restart the catalog server.

If you are using a stand-alone environment, see “Starting a stand-alone catalog service” on page 351 for more information. If you are using a WebSphere Application Server environment, see “Starting and stopping servers in a WebSphere Application Server environment” on page 363 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. 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. However, consider updating the servers first, then the clients, if you are using new functions in the upgrade.
 - a. Stop the container servers that you want to upgrade. You can stop the container server tier in groups with the **stopOgserver** command or the

xsadmin -teardown command. Batching teardown and starting servers up in parallel benefit placement activities because shards can be moved in larger groups.

```
$ bin/xsadmin.sh -teardown -fz DefaultZone
```

```
Connecting to Catalog service at localhost:1099
```

```
Processing filter options for Server teardown
```

```
The following servers will be torn down:
```

```
container00  
container01  
container02  
container03  
container04
```

```
Do you want to tear down the listed servers? (Y/N)
```

- b. 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:
 - “Migrating to WebSphere eXtreme Scale Version 7.1”
 - “Using the Update Installer to install maintenance packages” on page 70
 - c. Restart your container servers.
 - d. Upgrade any remaining container servers in your configuration.
3. If you are using multi-master replication, reconnect your catalog service domains. Use the **xsadmin -establishLink** command to reconnect the catalog service domains.

```
xsadmin -ch host -p 1099 -establishLink dname fdHostA:2809,fdHostB:2809
```

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 **xsadmin -establishLink** command.

Migrating to WebSphere eXtreme Scale Version 7.1

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” on page 74 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 7.1 by uninstalling the previous version and installing the new version in the same directory.

Procedure

1. Stop all processes that are using eXtreme Scale.
 - Read about stopping stand-alone servers to stop all processes that are running in your stand-alone eXtreme Scale environment.
 - 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.
4. Install eXtreme Scale Version 7.1. See Chapter 3, “Installing and deploying WebSphere eXtreme Scale,” on page 17 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 eXtreme Scale processes to begin using the product. See Chapter 8, “Administering the deployment environment,” on page 351 for more information.

Using the Update Installer to install maintenance packages

Use the IBM Update Installer to update your WebSphere eXtreme Scale environment with various types of maintenance, such as interim fixes, fix packs, and refresh packs.

About this task

Use the IBM Update Installer to install and apply various types of maintenance packages for WebSphere eXtreme Scale. Because the Update Installer undergoes regular maintenance, you must use the most current version of the tool.

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” on page 360 for more information.
 - To stop all processes that are running in your WebSphere Application Server environment, see Command-line utilities.
2. Download the latest version of the Update Installer. See Recommended fixes for more information.
3. Install the Update Installer. See Installing the Update Installer for WebSphere Software in the WebSphere Application Server Information Center for more information.

4. Download into the `updi_root/maintenance` directory the maintenance packages that you intend to install. See the Support site for more information.
5. Use the Update Installer to install the interim fix, fix pack, or refresh pack. You can install the maintenance package by running the graphical user interface (GUI), or by running the Update Installer in silent mode.

Run the following command from the `updi_root` directory to start the GUI:

- `Linux` `UNIX` `update.sh`
- `Windows` `update.bat`

Run the following command from the `updi_root` directory to run the Update Installer in silent mode:

- `Linux` `UNIX` `./update.sh -silent -options responsefile/file_name`
- `Windows` `update.bat -silent -options responsefile\file_name`

If the installation process fails, see the temporary log file, which is in the `updi_root/logs/update/tmp` directory. The Update Installer creates the `install_root/logs/update/maintenance_package.install` directory in which the installation log files are located.

Deprecated properties and APIs

The following list of properties and APIs were deprecated in the Version 7.1 release. Use the recommended migration action to determine how to update your configuration.

Deprecated items in Version 7.1

Table 5. *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.	7.1+ 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 for more information.
CoreGroupServicesMBean MBean and interface	7.1+ This MBean is deprecated starting in the Version 7.1 release. Use the CatalogServiceManagementMBean instead.
ServerMBean.updateTraceSpec() MBean operation	7.1+ This operation is deprecated starting in the Version 7.1 release. Use the TraceSpec attribute on the DynamicServerMBean instead.
CoreGroupServicesMBean MBean	7.1+ This MBean is deprecated starting in the Version 7.1 release. Use the CatalogServiceManagementMbean MBean instead.

Table 5. Deprecated properties and APIs (continued)

Deprecation	Recommended migration action
ServiceUnavailableException exception	<p>7.1+ This exception is deprecated starting in the Version 7.1 release.</p> <p>Use the <code>TargetNotAvailableException</code> exception instead.</p>
Partitioning facility (WPF): The partitioning facility is a set of programming APIs that allow Java EE applications to support asymmetric clustering.	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.	<p>You can still use these arguments, but change your system properties to the new values.</p> <ul style="list-style-type: none"> -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 <code>start0gServer</code> script. Use the -serverProps argument.

Chapter 6. Planning the WebSphere eXtreme Scale environment

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.

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” on page 67

Caching topology considerations

Your architecture can use local in-memory data caching or distributed client-server data caching. 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 “Caching topology: In-memory and distributed caching” on page 78.

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 Chapter 2, “Capacity planning,” on page 9.

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, a detailed list of formally supported hardware and software options by operating system is available on the Systems Requirements page of the product support site. If there is a conflict between the information provided in the information center and the information on the System Requirements page, the information at the Web site 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 are not required to install and deploy eXtreme Scale on a specific level of operating system. Each Java Platform, Standard Edition (J2SE) and Java Platform, Enterprise Edition (JEE) installation requires different operating system levels or fixes.

You can install and deploy the product in Java EE and J2SE environments. You can also bundle the client component with Java EE applications directly without integrating with WebSphere Application Server. WebSphere eXtreme Scale supports Java Runtime Environment (JRE) Version 1.4.2 and later and WebSphere Application Server Version 6.0.2 and later.

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

- **Without the web console**

eXtreme Scale does not require a specific operating system level. 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.

- **With the web console**

The following requirements apply for each operating system if using the console:

- Linux: 32-bit or 64-bit JVM
- Windows: 32-bit JVM only
- AIX®: 32-bit JVM only

You have to configure your choices in some cases. For example, by default, the product installer will use the 64-bit JVM in Linux unless you specify otherwise.

Web browser requirements

The web console supports the following Web browsers:

- Mozilla Firefox, version 3.5.x and later
- Mozilla Firefox, version 3.6.x and later
- Microsoft Internet Explorer, version 7 or 8

WebSphere Application Server requirements

eXtreme Scale clients and servers running in a distributed environment and local in-memory ObjectGrids are supported on WebSphere Application Server Version 6.0.2 and later.

Note: To use the dynamic cache provider, your system must meet one of the following minimum requirements:

- WebSphere Application Server Version 6.1.0.25 or higher and Interim Fix PK85622
- WebSphere Application Server Version 7.0.0.3 or higher and Interim Fix PK85622

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

Other application server requirements

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 1.4.2 or later.

Java SE considerations

WebSphere eXtreme Scale requires Java SE Version 1.4.2 or later. In general, newer versions of Java SE have better functionality and performance.

Supported versions

You can use WebSphere eXtreme Scale with Java SE Version 1.4.2 or later. The version that you use must be currently supported by the Java Runtime Environment (JRE) vendor.

A fully supported JRE 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.

WebSphere eXtreme Scale takes advantage of Java Development Kit (JDK) 5 or later functionality as it becomes available. Generally, newer versions of the Java Development Kit (JDK) and Java SE have better performance and functionality.

See Supported software for more information.

Java-dependent WebSphere eXtreme Scale features

Table 6. Features that require Java SE 5 or Java SE 6.

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	Supported in Java SE 6 and later
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: <ul style="list-style-type: none">• <code>wxssizeagent.jar</code>: Increases the accuracy of the used bytes map metrics.• <code>ogagent.jar</code>: Increases the performance of field-access entities.	X	X
Web console for monitoring		X

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 as 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 the information about accessing an ObjectGrid with

the ObjectGridManager in the *Programming Guide* 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 the information about accessing an ObjectGrid with the ObjectGridManager interface in the *Programming Guide* for more information.

Directory conventions

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

`wxs_install_root`

The `wxs_install_root` directory is the root directory where WebSphere eXtreme Scale product files are installed. The `wxs_install_root` directory can be the directory in which the trial zip file 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:

Example: `/opt/IBM/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 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 sub-directory 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:

Example: `/opt/IBM/eXtremeScale/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 sub-directory under the *wxs_home* directory.

- Example for stand-alone deployments:

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

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

samples_home

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

Example: /wxs-samples/

Caching topology: In-memory and distributed caching

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.

In-memory environments

When you deploy in a local, in-memory environment, WebSphere eXtreme Scale runs within a single Java virtual machine and is not replicated. To configure a local environment you can use an ObjectGrid XML file or the ObjectGrid APIs.

Distributed environments

When you deploy in a distributed environment, WebSphere eXtreme Scale runs across a set of Java virtual machines, increasing the performance, availability and

scalability. With this configuration, you can use data replication and partitioning. You can also add additional servers without restarting your existing eXtreme Scale servers. As with a local environment, an ObjectGrid XML file, or an equivalent programmatic configuration, is needed in a distributed environment. You must also provide a deployment policy XML file with configuration details

You can create either simple deployments or large, terabyte-sized deployments in which thousands of servers are needed.

Local in-memory cache

In the simplest case, WebSphere eXtreme Scale can be used as a local (non-distributed) in-memory data grid cache. The local case can especially benefit high-concurrency applications where multiple threads need to access and modify transient data. The data kept in a local data grid can be indexed and retrieved using queries. Queries help you to work with large in memory data sets. The support provided with the Java virtual machine (JVM), although it is ready to use, has a limited data structure.

The local in-memory cache topology for WebSphere eXtreme Scale is used to provide consistent, transactional access to temporary data within a single Java virtual machine.

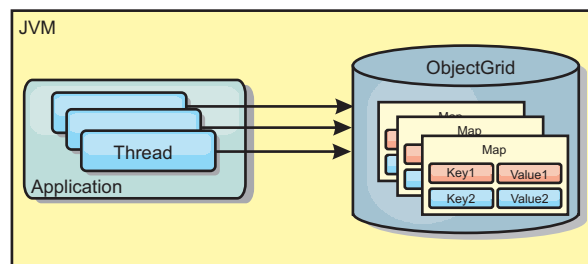


Figure 1. Local in-memory cache scenario

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.

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.

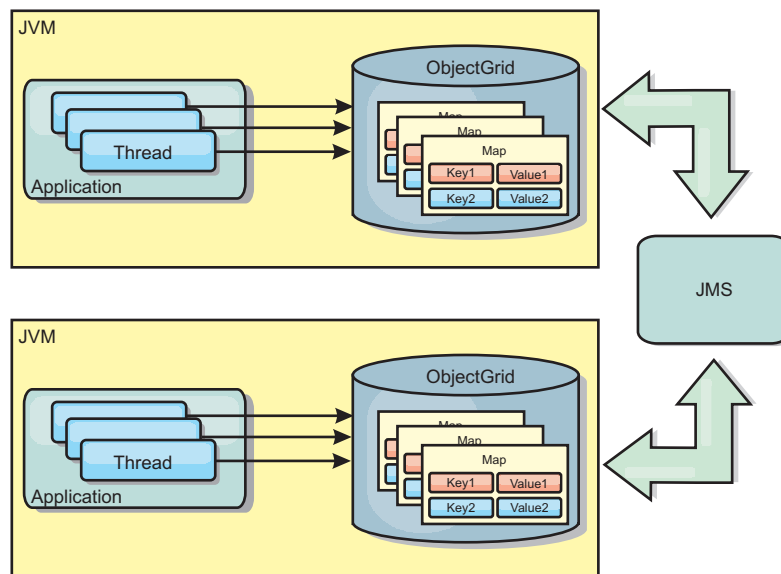


Figure 2. Peer-replicated cache with changes that are propagated with JMS

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.

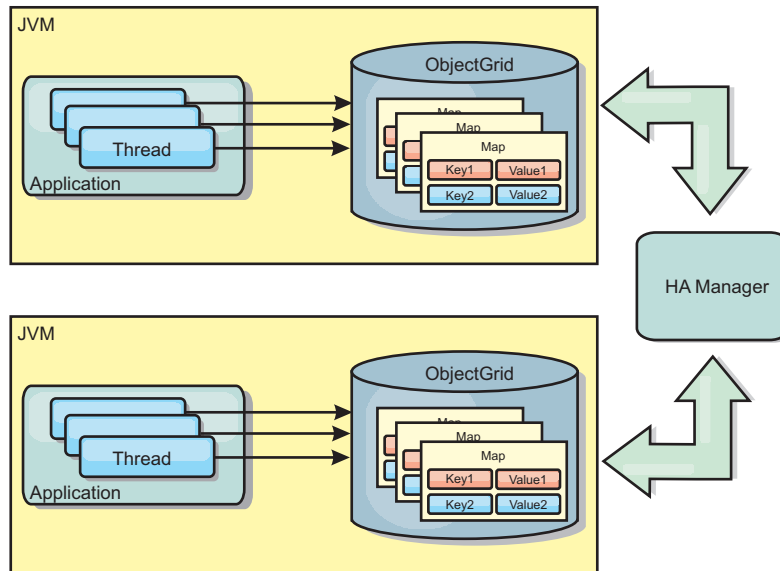


Figure 3. Peer-replicated cache with changes that are propagated with the high availability manager

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.

Distributed cache

WebSphere eXtreme Scale is most often used as a shared cache, to provide transactional access to data to multiple components where a traditional database would otherwise be used. The shared cache eliminates the need to configure a database.

Coherency of the cache

The cache is coherent because all of the clients see the same data in the cache. Each piece of data is stored on exactly one server in the cache, preventing wasteful copies of records that could potentially contain different versions of the data. A coherent cache can also hold more data as more servers are added to the data grid, and scales linearly as the grid grows in size. Because clients access data from this data grid with remote procedural calls, it can also be known as a remote cache, or far cache. Through data partitioning, each process holds a unique subset of the total data set. Larger data grids can both hold more data and service more requests for that data. Coherency also eliminates the need to push invalidation data around the data grid because no stale data exists. The coherent cache only holds the latest copy of each piece of data.

If you are running a WebSphere Application Server environment, the TranPropListener plug-in is also available. The TranPropListener plug-in uses the high availability component (HA Manager) of WebSphere Application Server to propagate the changes to each peer ObjectGrid cache instance.

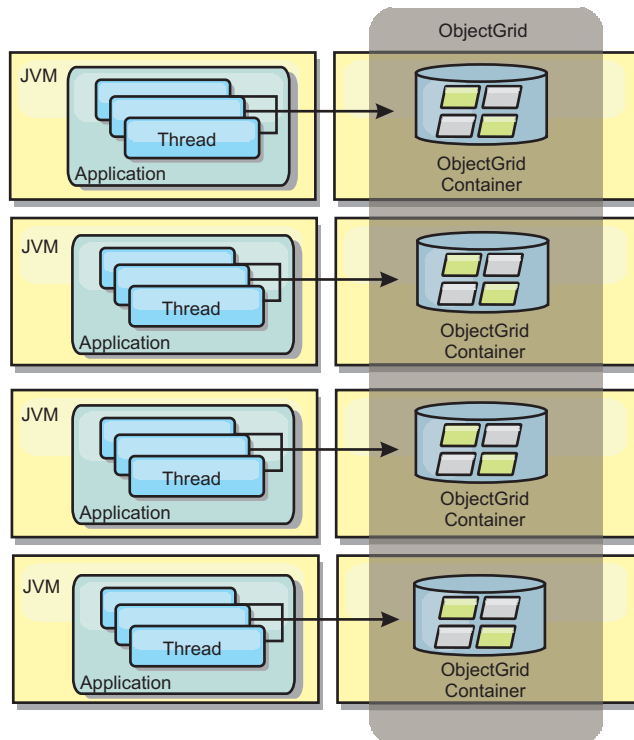


Figure 4. Distributed cache

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.

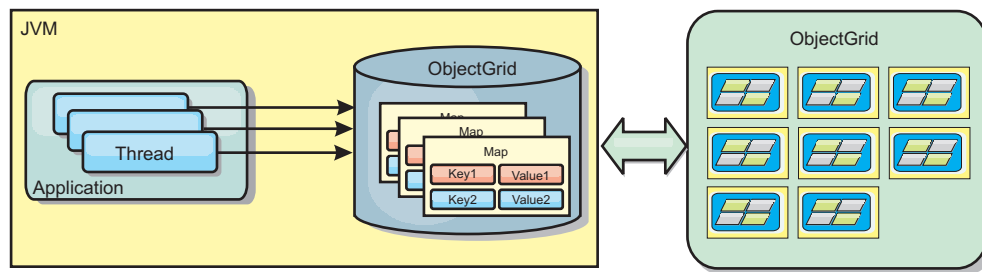


Figure 5. Near cache

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 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, set the `numberOfBuckets` attribute to 0 in the client override eXtreme Scale descriptor configuration. See the topic on map entry locking for details on eXtreme Scale lock strategies. The near cache can also be configured to have a separate eviction policy and different plug-ins using a client override eXtreme Scale descriptor configuration.

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 grid.
- Relies upon 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.

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.

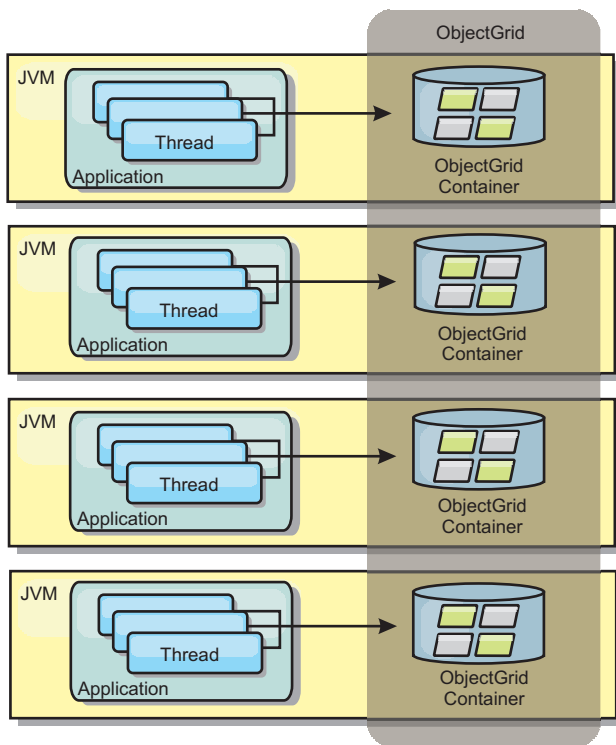


Figure 6. Embedded cache

Advantages

- Simplified administration since there are less processes to manage.
- Simplified application deployment since the grid is using the client application classloader.
- Supports partitioning and high availability.

Disadvantages

- Increased the memory footprint in client process since all of the data is collocated in the process.
- Increase CPU utilization for servicing client requests.
- More difficult to handle application upgrades since clients are using the same application Java archive files as the servers.
- Less flexible. Scaling of clients and grid servers cannot increase at the same rate. When servers are externally defined, you can have more flexibility in managing the number of processes.

When to use

Use embedded grids when there is plenty of memory free in the client process for grid data and potential failover data.

For more information, see the topic on enabling the client invalidation mechanism in the *Administration Guide*.

Multi-master data grid replication topologies

Using the multi-master asynchronous replication feature, two or more data grids can become exact mirrors of one other. This mirroring is accomplished using asynchronous replication among links connecting the data grids together. Each data grid is hosted in an independent catalog service domain, with its own catalog service, container servers, and a unique name. With the multi-master asynchronous replication feature, you can use links to interconnect a collection of these catalog service domains. Then, you can synchronize the catalog service domains with replication over the links. You can construct almost any topology because you choose how to define links among catalog service domains.

7.1+ Multi-master data grid replication is a significant new feature in Version 7.1. The feature is also called AP (availability and partitioning) replication in the context of the CAP theorem. The CAP theorem states that a distributed computer system cannot support more than two of the following three properties: consistency, availability, and partition tolerance.

See “Initial considerations for multi-master topologies” for map sets that are not replicated.

A replication data grid infrastructure is a connected graph of catalog service domains with bidirectional links among them. You can use a link between two catalog service domains to track data changes. For more information about how to set up communication between catalog service domains for multi-master replication, see “Available topologies for multi-master replication” on page 87.

Also, depending on the requirements of your environment, you can optimize the topology design for multi-master replication by taking several factors into consideration: arbitration, linking, and performance. Read more at “Topology considerations for multi-master replication” on page 90.

Initial considerations for multi-master topologies

Consider the following issues when you are deciding whether and how to use multi-master replication topologies.

- **Configuring class loaders with multiple catalog service domains**

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 JVMs 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.

- **Avoid loaders**

Loaders can be used to interface changes between a data grid and a database. It is unlikely that all data grids or domains in a topology are collocated geographically with the same database. WAN latency and other factors might render this use case undesirable.

Grid preloading also requires careful design. Usually, when a data grid is restarted, it is preloaded again. Preloading is not necessary or required when using multi-master replication. As soon as a catalog service domain is online, it

automatically reloads itself with the contents of the domains to which it is linked. As a result, you are not required to initiate a manual preload for a data grid that is a domain in a multi-master replication topology.

Loaders usually obey insert and update rules. With multi-master replication, inserts must be treated as merges. When the data is being pulled remotely after a domain restart, existing data will be merged into the local domain. Because the data might already have been in the local database, a typical insert fails with a duplicate key exception in the database. Use merge semantics instead.

WebSphere eXtreme Scale can be configured to do a shard-based preload with the preload methods on Loader plug-ins. But you should avoid this technique in a multi-master replication topology. Instead, use a client-based preload when the topology is first started. The multi-master topology refreshes any restarted domains with a current copy of what is stored in other domains in the topology. After domains have been started, the multi-master topology keeps domains synchronized.

- **EntityManager is not supported**

A map set containing an entity map is not replicated across catalog service domains.

- **Byte array maps are not supported in releases before Version 7.1.0.2**

A map set containing a map that is configured with COPY_TO_BYTES copy mode is not replicated across catalog service domains.

7.1.0.2+ In Version 7.1.0.2 or later, maps that are configured with COPY_TO_BYTES copy mode can replicate across catalog service domains. To enable this function, you must upgrade your entire configuration to Version 7.1.0.2 or later. All catalog servers, clients, and container servers, including container servers that are running only replica shards, in all domains must be upgraded. You cannot have COPY_TO_BYTES copy mode enabled on a catalog service domain that contains any servers that are running a version before Version 7.1.0.2. To upgrade your catalog service domains to support COPY_TO_BYTES copy mode, use the following steps:

1. Use the **xsadmin -dismissLink** command to remove the multi-master link between your catalog service domains. See “Configuring multi-master replication topologies” on page 227 for more information about running this command.
2. Shut down the data grid. You can use the **xsadmin -teardown** command to stop a group of catalog and container servers. See “Stopping servers gracefully with the xsadmin tool” on page 363 or “Starting and stopping servers in a WebSphere Application Server environment” on page 363 for more information. the information about stopping servers in the *Administration Guide* for more information.
3. Upgrade servers and clients in each domain to Version 7.1.0.2 or later. See “Updating eXtreme Scale servers” on page 67 for more information.
4. Update your configuration to use the COPY_TO_BYTES copy mode. See the information about byte array maps in the *Programming Guide* for more information about editing the byte array configuration.
5. Restart the data grid. See “Starting stand-alone servers” on page 351 or “Starting and stopping servers in a WebSphere Application Server environment” on page 363 for more information.
6. Use the **xsadmin -establishLink** command to reconnect the catalog service domains. See “Configuring multi-master replication topologies” on page 227 for more information about running this command. for more information.

After all of your catalog service domains are upgraded, you cannot start servers in any domains that are at a level that is lower than Version 7.1.0.2.

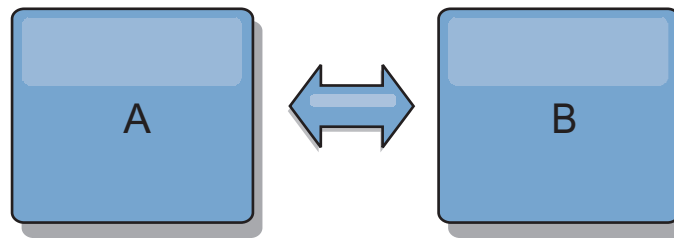
- **Write-behind is not supported**

A map set containing a map that is configured with write-behind support is not replicated across catalog service domains.

Available topologies for multi-master replication

You have several different options when choosing the topology for your deployment that incorporates multi-master replication.

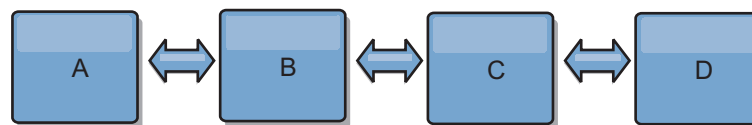
A replication data grid infrastructure is a connected graph of catalog service domains with bidirectional links among them. With a link, two catalog service domains can communicate data changes. For example, the simplest topology is a pair of catalog service domains with a single link between them. The 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, then eXtreme Scale 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 domain are copied to the other 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. Domain B pulls changes from Domain A. Domain C receives changes from Domain A through Domain B, which connects Domains A and C. Similarly, Domain D receives changes from the other domains through Domain C. This ability spreads the load of distributing changes away from the source of the changes.



Notice that if Domain C fails, the following would occur:

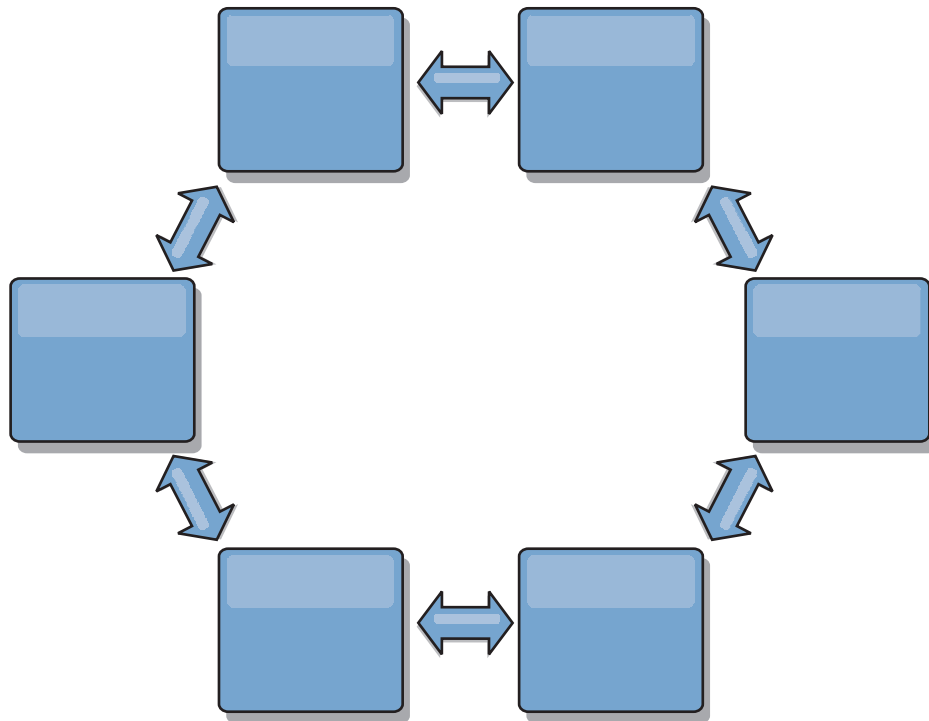
1. Domain D would be orphaned until Domain C was restarted

2. Domain C would synchronize itself with Domain B, which is a copy of Domain A
3. Domain D would use Domain C to synchronize itself with changes on Domains A and B. These changes initially occurred while Domain D was orphaned (while Domain C was down).

Ultimately, Domains A, B, C, and D would all become identical to one other again.

Ring topologies

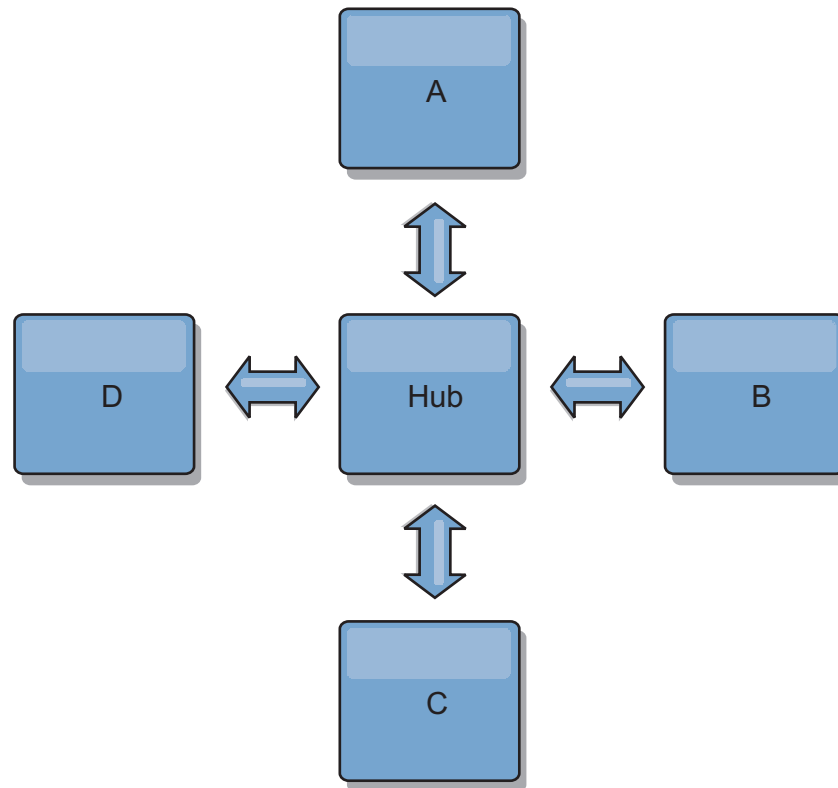
Another option you have with multi-master replication is a ring topology, which is more resilient than the topologies described in the previous sections. A catalog service domain or a single link can fail. Still, the surviving catalog service domains can obtain changes by traveling around the ring, away from the failure. Each catalog service domain has two links to the other catalog service domains. And each catalog service domain has at most two links, no matter how large the ring topology. Changes from a particular domain might travel through several domains before all of them mirror each other. Going through several domains causes potentially high latency, similar to the processes for a line topology.



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 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 domain is connected to every spoke 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 eXtreme Scale servers 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.

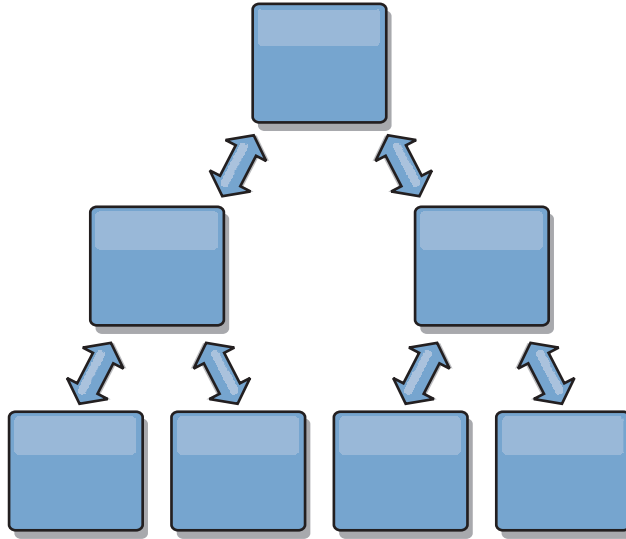
Any changes made by the client are local to the client, and are replicated asynchronously to the hub. The hub acts as an arbitration domain, distributing changes to all connected clients. The fully replicated clients topology provides a reliable L2 cache for an object relational mapper, such as OpenJPA. 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 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 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. You can use the tree topology when you have many catalog service domains such that the ring topology would overwork the hub. You can also use a tree if you require being able to 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.

Topology considerations for multi-master replication

When implementing multi-master replication, you must consider aspects in your design such as: arbitration, linking, and performance.

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:

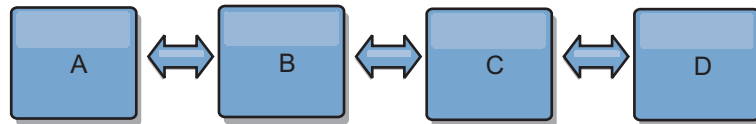
- Processor and network bandwidth on the source catalog service domain
- The number of intermediate catalog service domains and links between the source and target catalog service domain
- The processor and network resources available to the source, target, and intermediate catalog service domains

- **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.

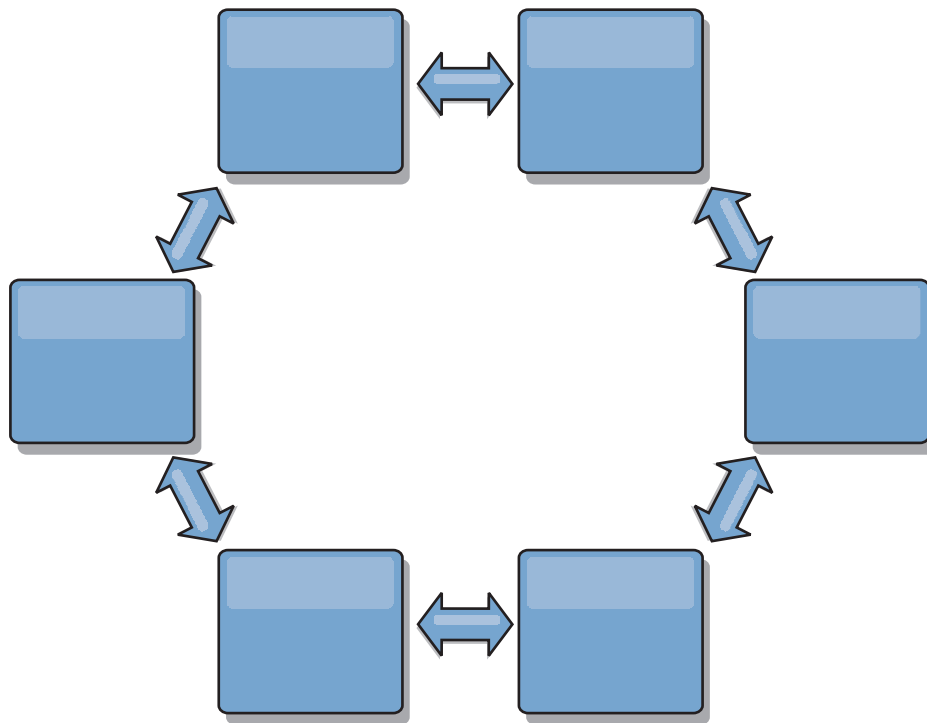
Consider the line topology with four catalog service domains A, B, C, and D:



If any of these conditions hold, Domain D does not see any changes from A:

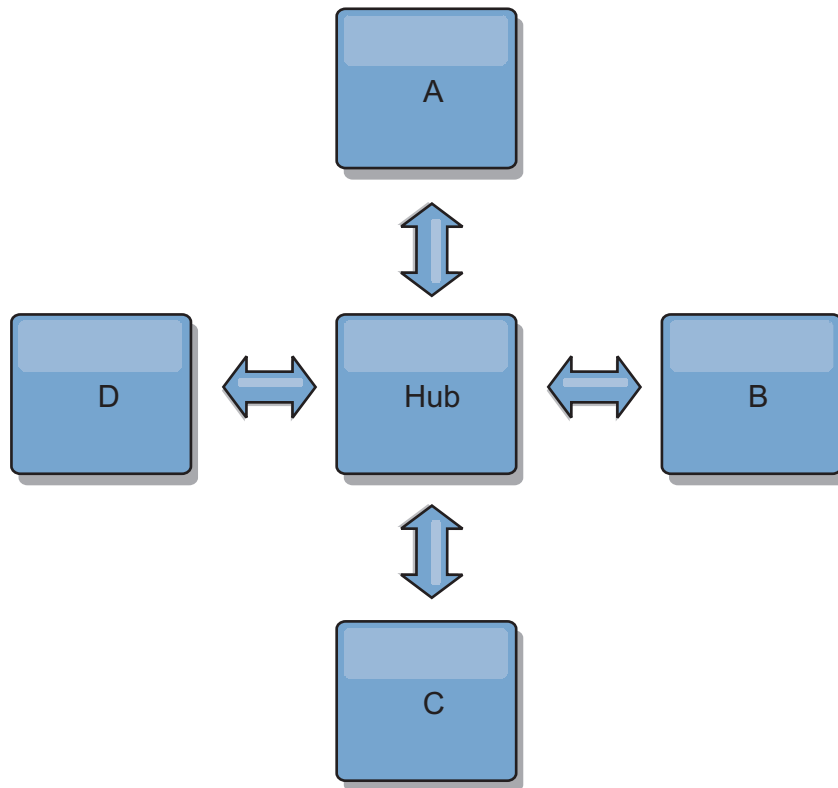
- Domain A is up and B is down
- Domains A and B are up and C is down
- The link between A and B is down
- The link between B and C is down
- The link between C and D is down

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 breakdown 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 off-loads 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.

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 7. 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.

Multi-master replication performance considerations

Take the following limitations into account when using multi-master replication topologies:

- **Change distribution tuning** (Discussed in previous section, "Linking and performance.")
- **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 overhead 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.

Catalog service

The catalog service hosts logic that should be idle during a steady state and has little influence on scalability. The catalog service is built to service hundreds of containers becoming available simultaneously and runs services to manage the containers.

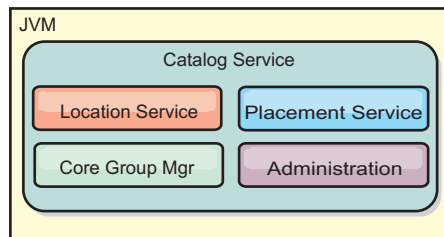


Figure 7. Catalog service

The catalog responsibilities consist of the following services:

Location service

The location service provides locality for clients that are looking for containers hosting applications and for containers that are looking to register hosted applications with the placement service. The location service runs in all of the grid members to scale out this function.

Placement service

The placement service is the central nervous system for the grid and is responsible for allocating individual shards to their host container. The placement service runs as a One of N elected service in the cluster. Because the One of N policy is used, there is always exactly one instance of the placement service running. If that instance should stop, another process takes over. All states of the catalog service are replicated across all servers hosting the catalog service for redundancy.

Core group manager

The core group manager manages peer grouping for health monitoring, organizes containers into small groups of servers, and automatically federates the groups of servers. When a container first contacts the catalog service, the container waits to be assigned to either a new or an existing group of several Java virtual machines (JVM). Each group of Java virtual machines monitors the availability of each of its members through heartbeating. One of the group members relays availability information to the catalog service to allow for reacting to failures by reallocation and route forwarding.

Administration

The four stages of administering your WebSphere eXtreme Scale environment are planning, deploying, managing, and monitoring.

For 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.

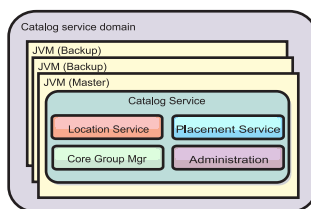


Figure 8. Catalog service domain

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 containers in your eXtreme Scale environment. The catalog service controls balancing and routing for all clients. To deploy eXtreme Scale as an in-memory database processing space, you must cluster the catalog service into a catalog service domain for high availability.

Components of the catalog service domain

When multiple catalog servers start, one of the servers is elected as the master catalog server that accepts Internet Inter-ORB Protocol (IIOP) 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. 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 an eXtreme Scale server and container contacts one of the catalog servers, the routing table for the catalog service domain is also propagated to the eXtreme Scale server and container 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.

The catalog service domain responsibilities are divided into a series of services:

Core group manager

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. With eXtreme Scale, 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 has the added responsibility of sending status to the core group manager when individual members fail. The same status mechanism is used to discover when all the members of a group fail, which causes the communication with the leader to fail.

The core group manager is a fully automatic service responsible for organizing containers into small groups of servers that are then automatically loosely federated to make an ObjectGrid. When a container 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 is a group of Java virtual machines that uses heart beating to monitor the availability of the other groups. One of these group members is elected the leader and has an added responsibility to relay availability information to the catalog service to allow for failure reaction by reallocation and route forwarding.

Placement service

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

Administration

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

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

Location service

The location service acts as the touchpoint for both clients that are searching for the containers that host the application they seek, as well as for the container servers that are registering hosted applications with the placement service. The location service runs on all of the data grid members to scale out this function.

Catalog service domain deployment

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 containers that become available simultaneously. For availability, configure the catalog service into a data grid.

Planning


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.

Starting a catalog service domain

For more information about creating a catalog service domain, see .

Connecting eXtreme Scale containers embedded in WebSphere Application Server to a stand-alone catalog service domain

You can configure eXtreme Scale containers that are embedded in a WebSphere Application Server environment to connect to a stand-alone catalog service domain.

- **7.1+** You can create catalog service domains in the WebSphere Application Server administrative console. See .
-  (deprecated) In previous releases, you connected the catalog services into a catalog service domain by creating a custom property. This property can still be used, but is deprecated. For more information about this custom property, see the information about starting the catalog service process in a WebSphere Application Server in the *Administration Guide*..

Note: Server name collision: Because this property is used to start the eXtreme Scale catalog server as well as to connect to it, catalog servers must not have the same name as any WebSphere Application Server server.

See the information about catalog server quorums in the *Product Overview* for more information.

Catalog server quorums

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

- “Important terms” on page 99

- “Heartbeats and failure detection”
- “Quorum behavior” on page 100
 - “Container behavior during quorum loss” on page 102
- “Client behavior during quorum loss” on page 103

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.
- **Heartbeat:** Heartbeats are used to determine if a given Java virtual machine (JVM) is running.

Heartbeats and failure detection

Container servers and core groups

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

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

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

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

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

Catalog service domain heart-beating

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

Failure detection

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

For more information about configuring heart-beating, see the information about configuring failover detection in the *Administration Guide*.

Quorum behavior

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

Reasons for quorum loss

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

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

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

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

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

Quorum loss from JVM failure

A catalog server that fails causes quorum to be lost. If a JVM fails, you must override quorum as fast as possible. The failed catalog service cannot rejoin the data grid until quorum has been overridden.

Quorum loss from network brownout

WebSphere eXtreme Scale is designed to expect the possibility of brownouts. A brownout is when a temporary loss of connectivity occurs between data centers. Brown outs are usually transient and clear within seconds or minutes. While WebSphere eXtreme Scale tries to maintain normal operation during the brownout period, a brownout is regarded as a single failure event. The failure is expected to be fixed and then normal operation resumes with no actions necessary.

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

Catalog service JVM cycling

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

Consequences of lost quorum

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

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

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

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

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

Quorum recovery

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

Overriding quorum

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

Administrators are the only ones with knowledge of a data center failure. WebSphere eXtreme Scale treats a brownout and a blackout similarly. You must inform the WebSphere eXtreme Scale environment of such failures with the **xsadmin** command to override quorum. This command tells the catalog service to assume that quorum is achieved with the current membership, and full recovery

takes place. When issuing an override quorum command, you are guaranteeing that the JVMs in the failed data center have truly failed and do not have a chance of recovering.

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

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

Container behavior during quorum loss

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

Synchronous replica behavior

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

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

Asynchronous replica behavior

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

Client behavior during quorum loss

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

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, which are hosted across multiple container servers. Each container server in turn hosts a subset of the complete data. A JVM might host one or more container servers and each container server 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.

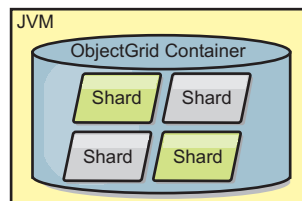


Figure 9. Container server

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

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

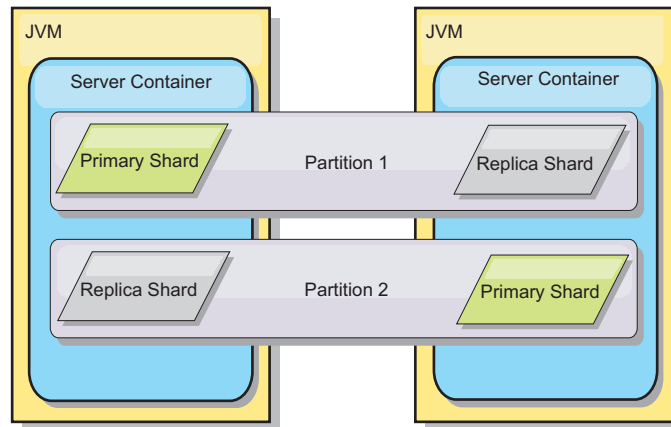


Figure 10. Partition

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.

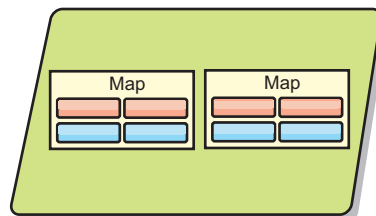


Figure 11. Shard

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.

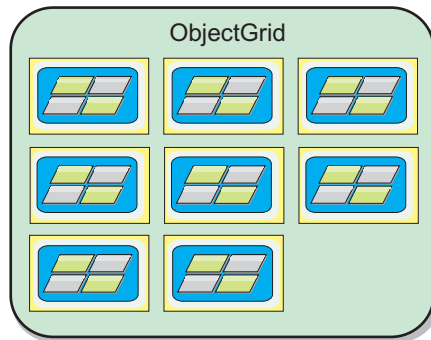


Figure 12. ObjectGrid

Capacity planning

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. Although such planning helps you deploy eXtreme Scale efficiently for future changes, it allows you to maximize the elasticity of eXtreme Scale 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.

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.

```
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:

$$\text{minNumJVMs} = (\text{numShardsPerPartition} * \text{numObjs}) / \text{numObjsPerJVM}$$

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:

$$\text{numShards} = \text{minNumJVMs} * 10 \text{ shards/JVM}$$

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.

$$\text{numPartitions} = \text{numShards} / \text{numShardsPerPartition}$$

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.

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.

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.

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.

Dynamic cache capacity planning

The Dynamic Cache API is available to Java EE applications that are deployed in WebSphere Application Server. The dynamic cache can be leveraged 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:

$$256\text{mb} * 4 \text{ containers} / (1 \text{ primary} + 1 \text{ replica}) = 512\text{mb.}$$

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

$$N = \text{Minimum}(T - 1, R)$$

Where:

- N = the number of processes that can crash simultaneously
- T = the total number of containers
- R = the total number of replicas

Formula C

$$\text{Ceiling}(T / (1+N)) = m$$

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 the [Tuning 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.

Operational checklist

Use the operational checklist to prepare your environment for deploying WebSphere eXtreme Scale.

Table 8. Operational checklist

Checklist item	For more information
<p>If you are using AIX, tune the following operating system settings:</p> <p>TCP_KEEPINTVL</p> <p>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:</p> <pre># no -o tcp_keepintvl</pre> <p>To change the current setting, run the following command:</p> <pre># no -o tcp_keepintvl=10</pre> <p>The TCP_KEEPINTVL setting is in half seconds.</p> <p>TCP_KEEPINIT</p> <p>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:</p> <pre># no -o tcp_keepinit</pre> <p>To change the current setting, run the following command:</p> <pre># no -o tcp_keepinit=40</pre> <p>The TCP_KEEPINIT setting is in half seconds.</p>	<ul style="list-style-type: none">For AIX tuning information, see Tuning AIX systems.
<p>Update the orb.properties file to modify the transport behavior of the grid. The orb.properties file is in the java/jre/lib directory.</p>	<p>"ORB properties" on page 237</p>

Table 8. Operational checklist (continued)

Checklist item	For more information
<p>Use parameters in the startOgServer script. In particular, use the following parameters:</p> <ul style="list-style-type: none"> • Set heap settings with the -jvmArgs parameter. • Set application class path and properties with the -jvmArgs parameter. • Set -jvmArgs parameters for configuring agent monitoring. <p>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:</p> <p>Listener port You can use the -listenerPort argument to specify the port that is used for communication between processes.</p> <p>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.</p> <p>JMX service port You can use the -JMXServicePort argument to specify the port that the JMX service should use.</p> <p>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.</p> <p>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</p>	<p>“startOgServer script” on page 356</p>
<p>Verify that security settings are configured correctly:</p> <ul style="list-style-type: none"> • Transport (SSL) • Application (Authentication and Authorization) <p>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.</p>	<p>“Security integration with external providers” on page 445</p>
<p>Choose how you are going to monitor your environment.</p> <ul style="list-style-type: none"> • xsAdmin tool: <ul style="list-style-type: none"> – The JMX ports of the catalog servers need to be visible to the xsAdmin tool. The container server ports also need to be accessible for some commands that gather information from the containers. • 7.1+ Monitoring console: With the monitoring console, you can chart current and historical statistics. • Vendor monitoring tools: <ul style="list-style-type: none"> – Tivoli® Enterprise Monitoring Agent – CA Wily Introscope – Hyperic HQ 	<ul style="list-style-type: none"> • “Monitoring with the xsadmin utility” on page 470 • “Java Management Extensions (JMX) security” on page 443 • 7.1+ “Monitoring with the web console” on page 457 • “Monitoring with the IBM Tivoli Enterprise Monitoring Agent for WebSphere eXtreme Scale” on page 494 • “Monitoring eXtreme Scale with Hyperic HQ” on page 503 • “Monitoring eXtreme Scale applications with CA Wily Introscope” on page 500

Chapter 7. Configuring the deployment environment

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 an 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

XML files and property files are the most common non-programmatic methods of configuring the product. See the Programming Guide for information about alternative methods, including application and system programming interfaces, plug-ins, and managed beans.

XML files for configuration

WebSphere eXtreme Scale is configured by a collection of XML files.

The following XML files are used to configure WebSphere eXtreme Scale:

“Deployment policy descriptor XML file” on page 192

Used to configure a deployment policy.

“ObjectGrid descriptor XML file” on page 153

Used to configure details for individual ObjectGrid instances.

“Entity metadata descriptor XML file” on page 258

Used to configure a set of entities and the relationships between the entities.

“Security descriptor XML file” on page 450

Used to enable security for a given deployment.

“Client properties file” on page 245

Used to specify client host names, ports, security, and other information.

“Spring descriptor XML file” on page 315

Used to enable the Spring Framework integration.

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 eXtreme Scale APIs.

About this task

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>
```

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);
```

As an alternative, you can create ObjectGrid instances programmatically without any XML. For example, you can use the following code snippet in place of the previous XML and code.

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid companyGrid = objectGridManager.createObjectGrid ("CompanyGrid", false);
BackingMap customerMap= companyGrid.defineMap("Customer");
BackingMap itemMap= companyGrid.defineMap("Item");
BackingMap orderLineMap= companyGrid.defineMap("OrderLine");
BackingMap orderMap = companyGrid.defineMap("Order");
```

For a complete description of the ObjectGrid XML file, see the eXtreme Scale configuration reference.

Configuring evictors

Evictors can be configured using the ObjectGrid descriptor XML file or programmatically.

About this task

For reference information on configuring evictors with XML, see "ObjectGrid descriptor XML file" on page 153.

TimeToLive (TTL) evictor

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 instance.

Enable the TTL evictor programmatically

TTL evictors are associated with BackingMap instances. The default evictor uses a time-to-live (TTL) eviction policy for each BackingMap instance. 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.

The following snippet of code uses the BackingMap interface to set the expiration time for each entry to 10 minutes after the entry was created.

programmatic time-to-live evictor

```
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.TTLType;
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogManager.createObjectGrid( "grid" );
BackingMap bm = og.defineMap( "myMap" );
bm.setTtlEvictorType( TTLType.CREATION_TIME );
bm.setTimeToLive( 600 );
```

The setTimeToLive method argument is 600 because it indicates the time-to-live value is in seconds. The preceding code must run before the initialize method is invoked on the ObjectGrid instance. These BackingMap attributes cannot be changed after the ObjectGrid instance is initialized. After the code runs, any entry that is inserted into the myMap BackingMap has an expiration time. After the expiration time is reached, the TTL evictor removes the entry.

To set the expiration time to the last access time plus 10 minutes, change the argument that is passed to the setTtlEvictorType method from TTLType.CREATION_TIME to TTLType.LAST_ACCESS_TIME. With this value, the expiration time is computed as the last access time plus 10 minutes. When an entry is first created, the last access time is the creation time. To base the expiration time on the last *update*, instead of merely the last *access* (whether or not it involved an update), substitute the TTLType.LAST_UPDATE_TIME setting for the TTLType.LAST_ACCESS_TIME setting.

When using the TTLType.LAST_ACCESS_TIME or TTLType.LAST_UPDATE_TIME setting, you can use the ObjectMap and JavaMap interfaces to override the BackingMap time-to-live value. This mechanism allows an application to use a different time-to-live value for each entry that is created. Assume that the preceding snippet of code set the ttlType attribute to LAST_ACCESS_TIME and set the time-to-live value to 10 minutes. An application can then override the time-to-live value for each entry by running the following code prior to creating or modifying an entry:

```
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.ObjectMap;
Session session = og.getSession();
ObjectMap om = session.getMap( "myMap" );
int oldTimeToLive1 = om.setTimeToLive( 1800 );
om.insert("key1", "value1" );
int oldTimeToLive2 = om.setTimeToLive( 1200 );
om.insert("key2", "value2" );
```

In the previous snippet of code, the entry with the key1 key has an expiration time of the insert time plus 30 minutes as a result of the setTimeToLive(1800) method invocation on the ObjectMap instance. The oldTimeToLive1 variable is set to 600

because the time-to-live value from the BackingMap is used as a default value if the setTimeToLive method was not previously called on the ObjectMap instance.

The entry with the key2 key has an expiration time of insert time plus 20 minutes as a result of the setTimeToLive(1200) method call on the ObjectMap instance. The oldTimeToLive2 variable is set to 1800 because the time-to-live value from the previous ObjectMap.setTimeToLive method invocation set the time-to-live value to 1800.

The previous example shows two map entries being inserted in the myMap map for keys key1 and key2. At a later point in time, the application can still update these map entries while retaining the time-to-live values that are used at insert time for each map entry. The following example illustrates how to retain the time-to-live values by using a constant defined in the ObjectMap interface:

```
Session session = og.getSession();
ObjectMap om = session.getMap( "myMap" );
om.setTimeToLive( ObjectMap.USE_DEFAULT );
session.begin();
om.update("key1", "updated value1" );
om.update("key2", "updated value2" );
om.insert("key3", "value3" );
session.commit();
```

Since the ObjectMap.USE_DEFAULT special value is used on the setTimeToLive method call, the key1 key retains its time-to-live value of 1800 seconds and the key2 key retains its time-to-live value of 1200 seconds because those values were used when these map entries were inserted by the prior transaction.

The previous example also shows a new map entry for the key3 key insert. In this case, the USE_DEFAULT special value indicates to use the default setting of time-to-live value for this map. The default value is defined by the time-to-live BackingMap attribute. See BackingMap interface attributes for information about how the time-to-live attribute is defined on the BackingMap instance.

See the API documentation for the setTimeToLive method on the ObjectMap and JavaMap interfaces. The documentation explains that an IllegalStateException exception results if the BackingMap.getTtlEvictorType method returns anything other than the TTLType.LAST_ACCESS_TIME or TTLType.LAST_UPDATE_TIME value. The ObjectMap and JavaMap interfaces can override the time-to-live value only when you are using the LAST_ACCESS_TIME or TTLType.LAST_UPDATE_TIME setting for the TTL evictor type. The setTimeToLive method cannot be used to override the time-to-live value when you are using the evictor type setting CREATION_TIME or NONE.

Enable the TTL evictor using XML configuration

Instead of using the BackingMap interface to programmatically set the BackingMap attributes to be used by the TTL evictor, you can use an XML file to configure each BackingMap instance. The following code demonstrates how to set these attributes for three different BackingMap maps:

enabling time-to-live 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="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>

```

The preceding 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 – and 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.

Plug in a pluggable evictor

Since evictors are associated with BackingMaps, use the BackingMap interface to specify the pluggable evictor.

Optional pluggable evictors

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.

Using optional pluggable evictors

To add optional pluggable evictors into the BackingMap configuration, you can use programmatic configuration or XML configuration, as described in the following section.

Programmatically plug in a pluggable evictor

Because evictors are associated with BackingMaps, use the BackingMap interface to specify the pluggable evictor. The following code snippet is an example of specifying a LRUEvictor evictor for the map1 BackingMap and a LFUEvictor evictor for the map2 BackingMap instance:

plugging in an evictor programmatically

```
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor;
import com.ibm.websphere.objectgrid.plugins.builtins.LFUEvictor;
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogManager.createObjectGrid( "grid" );
BackingMap bm = og.defineMap( "map1" );
LRUEvictor evictor = new LRUEvictor();
evictor.setMaxSize(1000);
evictor.setSleepTime( 15 );
evictor.setNumberOfLRUQueues( 53 );
bm.setEvictor(evictor);
bm = og.defineMap( "map2" );
LFUEvictor evictor2 = new LFUEvictor();
evictor2.setMaxSize(2000);
evictor2.setSleepTime( 15 );
evictor2.setNumberOfHeaps( 211 );
bm.setEvictor(evictor2);
```

The preceding snippet shows an LRUEvictor evictor being used for map1 BackingMap with an approximate maximum number of entries of 53,000 (53 * 1000). The LFUEvictor evictor is used for the map2 BackingMap with an approximate maximum number of entries of 422,000 (211 * 2000). Both the LRU and LFU evictors have a sleep time property that indicates how long the evictor sleeps before waking up and checking to see if any entries need to be evicted. The sleep time is specified in seconds. A value of 15 seconds is a good compromise between performance impact and preventing BackingMap from growing too large. The goal is to use the largest sleep time possible without causing the BackingMap to grow to an excessive size.

The setNumberOfLRUQueues method sets the LRUEvictor property that indicates how many LRU queues the evictor uses to manage LRU information. A collection of queues is used so that every entry does not keep LRU information in the same queue. This approach can improve performance by minimizing the number of map entries that need to synchronize on the same queue object. Increasing the number of queues is a good way to minimize the impact that the LRU evictor can cause on performance. A good starting point is to use ten percent of the maximum number of entries as the number of queues. Using a prime number is typically better than using a number that is not prime. The setMaxSize method indicates how many entries are allowed in each queue. When a queue reaches its maximum number of entries, the least recently used entry or entries in that queue are evicted the next time that the evictor checks to see if any entries need to be evicted.

The setNumberOfHeaps method sets the LFUEvictor property to set how many binary heap objects the LFUEvictor uses to manage LFU information. Again, a collection is used to improve performance. Using ten percent of the maximum number of entries is a good starting point and a prime number is typically better than using a number that is not prime. The setMaxSize method indicates how many entries are allowed in each heap. When a heap reaches its maximum number of entries, the least frequently used entry or entries in that heap are evicted the

next time that the evictor checks to see if any entries need to be evicted.

XML configuration approach to plug in a pluggable evictor

Instead of using various APIs to programmatically plug in an evictor and set its properties, an XML file can be used to configure each BackingMap as illustrated in the following sample:

```
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="LRU">
    <bean id="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="LFU">
    <bean id="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>
```

Memory-based eviction

All built-in evictors support memory-based eviction that can be enabled on 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 eXtreme Scale configuration reference.

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.

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 an eXtreme Scale client process, you can use the MemoryPoolMXBean. See also: containerServer.props file and Starting eXtreme Scale server processes.

During run time, 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, there is no guarantee that the eviction speed is fast enough to avoid a potential out of memory error if the system run time continues to quickly consume memory.

Plug-ins for indexing data

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.

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.

Also, note that 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="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(true);

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 HashIndex into BackingMap

In the following example, you configure the HashIndex plug-in by adding static index plug-ins to the XML file:

```
<backingMapPluginCollection id="person">
  <bean id="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 identifying 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 only supports equals functions. If the index is 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 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. These attributes define properties such as if you are using an attribute or composite HashIndex, or if range indexing is enabled.

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.

See the topic on a composite HashIndex in the *Administration Guide*.

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 the information about query optimization with indexes in the *Programming Guide*.

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 9. Support for range index. States whether `HashIndex` types support range index.

HashIndex type	Supports range index
Single-attribute <code>HashIndex</code> : indexed key or attribute is of type <code>Comparable</code>	Yes
Single-attribute <code>HashIndex</code> : indexed key or attribute is not of type <code>Comparable</code>	No
Composite <code>HashIndex</code>	No
Relationship <code>HashIndex</code>	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.

Configuring a locking strategy

You can define an optimistic, a pessimistic, or no locking strategy on each `BackingMap` in the `WebSphere eXtreme Scale` configuration.

About this task

Each BackingMap instance can be configured to use one of the following locking strategies:

1. Optimistic locking mode
2. Pessimistic locking mode
3. None

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 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.

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.

For more information about locking strategies, see the information about locking strategies in the *Product Overview*.

You can specify a locking strategy programmatically or with XML. For more information about locking, see the information about locking strategies in the *Product Overview*.

Procedure

• Configure an optimistic locking strategy

- Programmatically using the setLockStrategy method:

```
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("test");
BackingMap bm = og.defineMap("optimisticMap");
bm.setLockStrategy( LockStrategy.OPTIMISTIC );
```

- Using the lockStrategy attribute in the “ObjectGrid descriptor XML file” on page 153:

```
<?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 pessimistic locking strategy**

- Programmatically using the setLockStrategy method:

```
specify pessimistic strategy programmatically
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("test");
BackingMap bm = og.defineMap("pessimisticMap");
bm.setLockStrategy( LockStrategy.PESSIMISTIC);
```

- Using the lockStrategy attribute in the “ObjectGrid descriptor XML file” on page 153:

```
specify pessimistic strategy 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="test">
            <backingMap name="pessimisticMap"
                lockStrategy="PESSIMISTIC"/>
        </objectGrid>
    </objectGrids>
</objectGridConfig>
```

- **Configure a no locking strategy**

- Programmatically using the setLockStrategy method:

```
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("test");
BackingMap bm = og.defineMap("noLockingMap");
bm.setLockStrategy( LockStrategy.NONE);
```

- Using the lockStrategy attribute in the “ObjectGrid descriptor XML file” on page 153:

```
<?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>
```

What to do next

To avoid a java.lang.IllegalStateException exception, you must call the setLockStrategy method before calling the initialize or getSession methods on the ObjectGrid instance.

Configuring loaders

Implementing a loader requires configuration for several attributes.

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 the information about in-line caching scenarios in the *Product Overview*.

Each backing map has a boolean `preloadMode` attribute that is set to indicate if preload of a map executes 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 eXtreme Scale 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. It is possible that the `get` and `batchUpdate` methods are 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.

It is also possible that several ObjectGrid maps are stored in the same database. Each map has its own Loader and each Loader might need to connect to the same database. When connecting to the same database, each Loader wants to use the same JDBC connection so that the changes to each table are committed as part of the same database transaction. Typically, the same person who writes the Loader implementation also writes the TransactionCallback implementation. 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 `getVersionedObjectForValue` 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.

Configuring write-behind loader support

You can enable write-behind support either using the ObjectGrid descriptor XML file or programmatically using 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 configuring an ObjectGrid using 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 and/or a maximum key update count. The format of the write-behind parameter is:

```
::= <defaults> | <update time> | <update key count> | <update time> ";"  
<update key count>::= "T" <positive integer>::= "C" <positive integer>::= ""
```

- write-behind attribute
- update time
- update key count
- defaults

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 just 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, the first behind update could happen after up to twice as long as the update time. This is because ObjectGrid randomizes the update starting time so all partitions will not hit the database simultaneously.

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.

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).

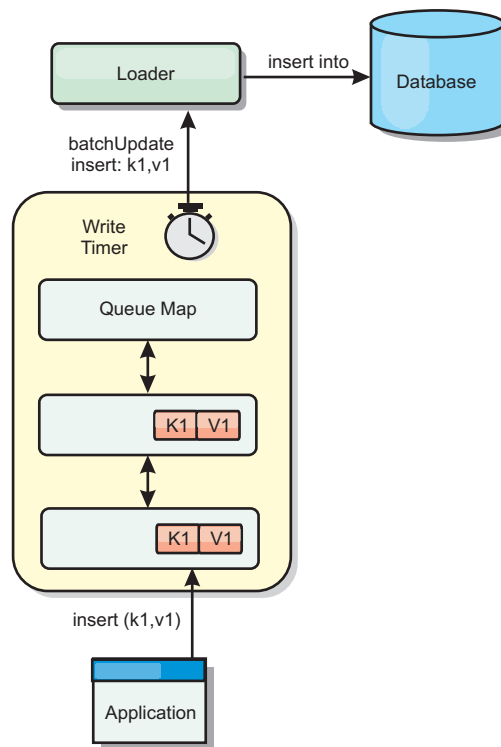


Figure 13. Write-behind caching

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.

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.

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, OPTIMISTIC, 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.

Loader failures

The loader plug-in can fail when it is unable to communicate to the database back end. This can happen if the database server or the network connection is down. The write-behind loader will queue the updates and try 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.

Therefore, the Loader implementation should be able to distinguish a data failure or a physical loader failure. Data failure should be thrown or re-thrown as a `LoaderException` or an `OptimisticCollisionException`, but a physical loader failure should be thrown or re-thrown as a `LoaderNotAvailableException`. ObjectGrid handles these two exceptions differently:

- If a `LoaderException` is caught by the write-behind loader, the write-behind loader will consider it fails due to some data failure, such as duplicate key failure. The write-behind loader will unbatch the update, and try 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 will consider it fails 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 will wait for 15 seconds and then re-try the batch update to the database.

The common mistake is to throw a `LoaderException` while a `LoaderNotAvailableException` should be thrown. All the records queued in the write-behind loader will become failed update records, which defeats the purpose of back-end failure isolation.

Performance considerations

Write-behind caching support increases 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.

Write-behind caching support

You can use write-behind caching to reduce the overhead that occurs when updating a back-end database. Write-behind caching queues updates to the Loader plug-in.

Introduction

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).

When you configure the write-behind setting on a backing map, a write-behind thread is created and wraps the configured loader. When an eXtreme Scale transaction inserts, updates, or removes an entry from an eXtreme Scale 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 will only send 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” on page 266 information about configuring a JPA loader.

Benefits

Enabling write-behind support has the following benefits:

- Backend 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.
- 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.

ObjectGrid descriptor XML

When configuring an eXtreme Scale using an eXtreme Scale 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 and/or a maximum key update count. 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, you cannot guarantee that the actual update count or update time are the same as defined in the parameters. Also, the first behind update could happen after up to twice as long as the update

time. This is because eXtreme Scale randomizes the update starting time so all partitions will not hit the database simultaneously.

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.

The table below lists some write-behind attribute examples.

Note: If you configure the write-behind loader as an empty string: `writeBehind=""`, the write-behind loader is enabled using the default values. Therefore, do not specify the `writeBehind` attribute if you do not want write-behind support enabled.

Table 10. 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).

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 the information about the `BackingMap` interface in the *Programming Guide*.

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 eXtreme Scale transaction encloses the back-end transaction. The eXtreme Scale transaction starts before the back-end transaction starts, and it ends after the back-end transaction ends.

With write-behind support enabled, the eXtreme Scale transaction finishes before the back-end transaction starts. The eXtreme Scale transaction and back-end transaction are decoupled.

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 eXtreme Scale 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.

Failed updates

Because the eXtreme Scale transaction finishes before the back-end transaction starts, it is possible to have transaction false success. For example, if you try to insert an entry in an eXtreme Scale transaction that does not exist in the backing map but does exist in the back-end, causing a duplicate key, the eXtreme Scale transaction does succeed. However, the transaction in which the write-behind thread inserts that object into the back-end fails with a duplicate key exception.

Refer to “Handling failed write-behind updates” on page 142 for how to handle such failures.

Queue map locking behavior

Another major transaction behavior difference is the locking behavior. eXtreme Scale supports three different locking strategies: PESSIMISTIC, OPTIMISTIC, 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 eXtreme Scale 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 eXtreme Scale 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

WebSphere eXtreme Scale 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 automatically retries to write failed transactions and sends 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)` method 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 in API documentation for more details.

Loader failures

The loader plug-in can fail when it is unable to communicate to the database back end. This can happen if the database server or the network connection is down. The write-behind loader will queue the updates and try to push the data changes to the loader periodically. The loader must notify the WebSphere eXtreme Scale run time that a database connectivity problem exists by throwing a `LoaderNotAvailableException` exception.

Therefore, the Loader implementation should be able to distinguish a data failure or a physical loader failure. Data failure should be thrown or re-thrown as a `LoaderException` or an `OptimisticCollisionException` exception, but a physical loader failure should be thrown or re-thrown as a `LoaderNotAvailableException` exception. WebSphere eXtreme Scale handles these two exceptions differently:

- If a `LoaderException` is caught by the write-behind loader, the write-behind loader will consider it fails due to some data failure, such as duplicate key failure. The write-behind loader will unbatch the update, and try the update one record at one time to isolate the data failure. If A `LoaderException` exception 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 will consider it fails 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 will wait for 15 seconds and then re-try the batch update to the database.

The common mistake is to throw a `LoaderException` while a `LoaderNotAvailableException` should be thrown. All the records queued in the write-behind loader will become failed update records, which defeats the purpose of back-end failure isolation. This mistake will likely happen if you write a generic loader to talk to databases.

The eXtreme Scale provided `JPALoader` is one example. The `JPALoader` uses JPA API to interact with database backends. When the network fails, the `JPALoader` gets a `javax.persistence.PersistenceException` but it does not know the essence of the failure unless the SQL state and SQL error code of the chained `SQLException` are checked. The fact that the `JPALoader` is designed to work with all types of database further complicates the problem as the SQL states and error codes are different for the network down problem. To solve this, WebSphere eXtreme Scale provides an `ExceptionMapper` API to allow users plug in an implementation to map an Exception to a more consumable exception. For example, users can map a generic `javax.persistence.PersistenceException` to a `LoaderNotAvailableException` if the SQL state or error code indicates the network is down.

Performance considerations

Write-behind caching support increases response time by removing the loader update from the transaction. It also increases database throughput since 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 pushes 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.

Handling failed write-behind updates

Since the WebSphere eXtreme Scale transaction finishes before the back-end transaction starts, it is possible to have transaction false success.

For example, if you try to insert an entry in an eXtreme Scale transaction which does not exist in the backing map but does exist in the back-end, causing a duplicate key, the eXtreme Scale transaction does succeed. However, the transaction in which the write-behind thread inserts that object into the back-end 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 1 hour after it has been inserted. So the failed-update records will be lost if they are not retrieved within 1 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` call works with a specific partition for each eXtreme Scale transaction. In a distributed environment, in order 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: Failed update map provides a way to monitor the application health. If a system produces a lot of records in the failed update map, it is a sign the application or architecture should be re-evaluated or revised to use the write-behind support. Starting from 6.1.0.5, you can use xsadmin script 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 should be added to the ObjectGrid descriptor XML as in the snippet.

```

<objectGrid name="Grid">
  <bean id="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 Write-behind dumper class sample code for more information.

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

}
}
}

```

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” on page 150.

See “Enabling the client invalidation mechanism” on page 252 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="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.

For more information about how Java Message Service (JMS) can be used to distribute transactional changes, see the information about using JMS to distribute transaction changes in the *Product Overview*.

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 the information about configuring peer-to-peer replication with JMS in the *Administration Guide*.

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 committed.
        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 (JMSEException 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");
}
}

```

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 obtain 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 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="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`.

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” on page 169 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, querySchema element, and streamQuerySet 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. Set to AUTHORIZATION_MECHANISM_CUSTOM when you are using a custom MapAuthorization plug-in. 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)

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)

entityMetadataXMLFile

Specifies the relative path to the entity descriptor XML file. The path is relative to the location of the ObjectGrid descriptor file. Use this attribute to define an entity schema using an XML file. Entities must be defined before starting eXtreme Scale so that each entity can bind with a BackingMap. (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 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" 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 the information about the BackingMap interface in the *Programming Guide*.

- Number of occurrences: Zero to many
- Child element: timeBasedDBUpdate element

Attributes

name

Specifies the name that is assigned to the backingMap instance. If this attribute is missing, the XML validation fails. (Required)

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)

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)

numberOfBuckets

Specifies the number of buckets for the BackingMap instance to use. The BackingMap instance uses a hash map for implementation. 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 remotely communicating with eXtreme Scale. When you set the value to 0 for a client, set the value in the client override ObjectGrid XML descriptor file only. (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)

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.

For more information see the information about map entry locking in the *Programming Guide*.

numberOfLockBuckets

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. The lock manager uses a hash map to track entries that are locked by one or more transactions. 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)

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)

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 the information about CopyMode best practices in the *Programming Guide*.

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)

copyKey

Specifies if 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)

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)

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)

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)

streamRef

Specifies that the backingMap is a stream source map. Any insert or update to the backingMap is converted into a streaming event to the stream query engine. This attribute must reference a valid stream name within a streamQuerySet. (Optional)

viewRef

Specifies that the backingMap is a view map. The view output from the stream query engine is converted into eXtreme Scale tuple format and put into the 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 support” on page 137.

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” on page 266 information about configuring a JPA loader.

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`. (Optional)

```
<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"
(19) evictionTriggers="MEMORY_USAGE_THRESHOLD"
/>
```

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
- `MapAuthorization` 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. (Required)

```
<bean
(1) id="TransactionCallback" | "ObjectGridEventListener" | "SubjectSource" |
    "MapAuthorization" | "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="ObjectGridEventListener"
      className="com.ibm.websphere.objectgrid.plugins.builtins.TranPropListener"/>
    <backingMap name="Customer"
      pluginCollectionRef="customerPlugins"/>
    </objectGrid>
  </objectGrids>
  <backingMapPluginCollections>
    <backingMapPluginCollection id="customerPlugins">
      <bean id="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 the topic about an introduction to plug-ins in the *Programming Guide*.

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="customerPlugins">
      <bean id="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="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 LRU Evictor. 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="customerPlugins">
      <bean id="Evictor"
        className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor"/>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="collection2">
      <bean id="Evictor"
        className="com.ibm.websphere.objectgrid.plugins.builtins.LFUEvictor"/>
      <bean id="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.

- 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);

```

streamQuerySet element

The `streamQuerySet` element is the top-level element for defining a stream query set.

- Number of occurrences: Zero to many
- Child element: stream element, view element

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"
/>
```

In the following example, the StreamQueryApp2.xml file is used to demonstrate how to configure the attributes of a streamQuerySet. The stream query set `_stockQuoteSQS_` has one stream and one view. Both the stream and view define its name, valueClass, sql, and access type. The stream also defines a basic element, which specifies that the volume attribute in the StockQuote class is mapped to the SQL column transaction volume that is defined in the SQL statement.

```
<?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="og1">
<backingMap name="stockQuote" readOnly="false" copyKey="true" streamRef="stockQuote"/>
<backingMap name="last5MinuteAvgPrice" readOnly="false" copyKey="false"
viewRef="last5MinuteAvgPrice"/>

<streamQuerySet name="stockQuoteSQS">
<stream
name="stockQuote"
valueClass="com.ibm.ws.objectgrid.streamquery.sample.guide.StockQuote"
sql="create stream stockQuote
keyed by t ( transactionvolume INTEGER, price DECIMAL (9,2), issue VARCHAR(100) );"
access="FIELD">
<basic name="volume" column="transactionvolume"/>
</stream>

<view
name="last5MinuteAvgPrice"
valueClass="com.ibm.ws.objectgrid.streamquery.sample.guide.AveragePrice"
sql="CREATE VIEW last5MinuteAvgPrice AS SELECT issue, avg(price) as avgPrice
FROM (SELECT * FROM stockQuote FETCH LATEST 5 MINUTES) group by issue;"
access="FIELD"
</view>
</streamQuerySet>
</objectGrid>
</objectGrids>
</objectGridConfig>
```

objectGrid.xsd file

Use the ObjectGrid descriptor XML schema to configure WebSphere eXtreme Scale.

See the “ObjectGrid descriptor XML file” on page 153 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” on page 315.

```
<?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:key name="streamName">
      <xsd:selector xpath="dgc:objectGrids/dgc:objectGrid/dgc:
        streamQuerySet/dgc:stream"/>
      <xsd:field xpath="@name"/>
    </xsd:key>

    <xsd:keyref name="streamRef" refer="dgc:streamName">
      <xsd:selector xpath="dgc:objectGrids/dgc:objectGrid/dgc:backingMap"/>
      <xsd:field xpath="@streamRef"/>
    </xsd:keyref>

    <xsd:key name="viewName">
      <xsd:selector xpath="dgc:objectGrids/dgc:objectGrid/dgc:streamQuerySet/dgc:view"/>
      <xsd:field xpath="@name"/>
    </xsd:key>

    <xsd:keyref name="viewRef" refer="dgc:viewName">
      <xsd:selector xpath="dgc:objectGrids/dgc:objectGrid/dgc:backingMap"/>
      <xsd:field xpath="@viewRef"/>
    </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:unique name="streamQuerySetNameUnique">
          <xsd:selector xpath="dgc:streamQuerySet"/>
          <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:schema>
```

```

</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:element maxOccurs="unbounded" minOccurs="0" name="streamQuerySet"
type="dgc:streamQuerySet">
<xsd:unique name="stream">
<xsd:selector xpath="dgc:stream"/>
<xsd:field xpath="@name"/>
</xsd:unique>
<xsd:unique name="view">
<xsd:selector xpath="dgc:view"/>
<xsd:field xpath="@name"/>
</xsd:unique>
</xsd:element>
</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: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="ttlEvictoryType" type="dgc:ttlEvictoryType" use="optional"/>
<xsd:attribute name="streamRef" type="xsd:string" use="optional"/>
<xsd:attribute name="viewRef" type="xsd:string" 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="required"/>
<xsd:attribute name="id" type="dgc:beanId" use="required"/>
</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="SubjectSource"/>
    <xsd:enumeration value="MapAuthorization"/>
    <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="MapIndexPlugin"/>

  </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: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="streamQuerySet">
  <xsd:sequence>
    <xsd:element maxOccurs="unbounded" minOccurs="0" name="stream" type="dgc:stream">
      <xsd:unique name="streamBasicColumnUnique">
        <xsd:selector xpath="dgc:basic"/>
        <xsd:field xpath="@column"/>
      </xsd:unique>
    </xsd:element>
    <xsd:element maxOccurs="unbounded" minOccurs="0" name="view" type="dgc:view">
      <xsd:unique name="viewBasicColumnUnique">
        <xsd:selector xpath="dgc:basic"/>
        <xsd:field xpath="@column"/>
      </xsd:unique>
    </xsd:element>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string" use="required"/>
  <xsd:attribute name="viewResultsToListenersOnly" type="xsd:boolean"
    default="false" use="optional"/>
  <xsd:attribute name="deployInPrimaryOnly" type="xsd:boolean" default="true"
    use="optional"/>
</xsd:complexType>

<xsd:complexType name="stream">
  <xsd:sequence>
    <xsd:element maxOccurs="unbounded" minOccurs="0" name="basic" type="dgc:basic"/>
  </xsd:sequence>
  <xsd:attribute name="valueClass" type="xsd:string" use="required"/>
  <xsd:attribute name="name" type="xsd:string" use="required"/>
  <xsd:attribute name="sql" type="xsd:string" use="optional"/>
  <xsd:attribute name="access" type="cc:accessType" use="optional"/>
</xsd:complexType>

<xsd:complexType name="view">
  <xsd:sequence>
    <xsd:element maxOccurs="unbounded" minOccurs="0" name="id" type="dgc:basic"/>
    <xsd:element element maxOccurs="unbounded" minOccurs="0" name="basic"
      type="dgc:basic"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string" use="required"/>
  <xsd:attribute name="sql" type="xsd:string" use="optional"/>
  <xsd:attribute name="valueClass" type="xsd:string" use="optional"/>
  <xsd:attribute name="access" type="cc:accessType" use="optional"/>
</xsd:complexType>

<xsd:complexType name="basic">
  <xsd:attribute name="name" type="xsd:string" use="required"/>
  <xsd:attribute name="column" type="xsd:string" use="required"/>
</xsd:complexType>

<xsd:complexType name="id">
  <xsd:attribute name="name" type="xsd:string" use="required"/>
  <xsd:attribute name="column" type="xsd:string" use="required"/>
</xsd:complexType>

<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:unique>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>

```

```

        <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:schema>

```

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.

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

For information on starting container servers, see “Starting container processes” on page 354 or “Configuring WebSphere Application Server applications to automatically start container servers” on page 222.

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

For more information on policy configuration, see “Deployment policy descriptor XML file” on page 192.

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 containers 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, 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 one and 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>
```



```
</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 containers are available to support this ObjectGrid instance. The `numInitialContainers` attribute is ignored after the specified number of containers has been reached.

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 containers. The catalog service forwards requests, allocates space in host containers, 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 in a JVM by itself or load the container into an arbitrary JVM with other containers 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.

You can also create a deployment policy programmatically when you are embedding a container in an existing Java process or application. For more information, see the DeploymentPolicy API documentation.

Controlling shard placement with 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

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.

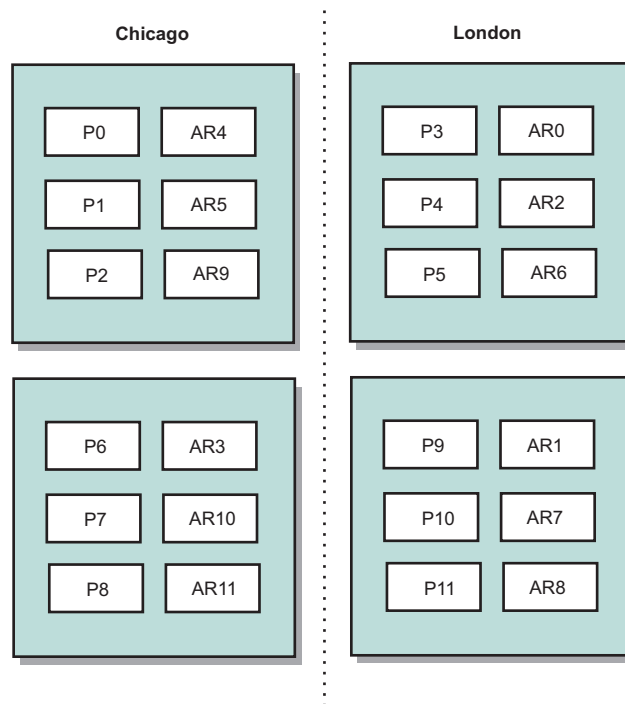


Figure 14. Primaries and replicas in zones

Three configuration settings in eXtreme Scale control shard placement:

- Set the deployment file
- Group containers
- Specify rules

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

Disable development mode

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

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

For more information about the XML file, see “Deployment policy descriptor XML file” on page 192.

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 `startOgServer` script. Multiple container servers can have the same IP address.

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

Define zones to group container servers

Container servers are assigned to zones with the `-zone` parameter on the `startOgServer` 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 containers*.

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

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

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

Specify zone rules

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

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

If more than one synchronous or asynchronous replica exist, then you must provide `shardMapping` elements of the appropriate shard type. The `exclusivePlacement` attribute of the `zoneRule` element determines the placement of shards in the same partition in zones. The `exclusivePlacement` attribute values are:

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

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

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

The default setting is `true`.

For more information, see *Zone definition examples for deployment file*.

Extended use cases

The following are various use cases for shard placement strategies:

Rolling upgrades

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

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

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

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

Upgrading eXtreme Scale

When you are upgrading eXtreme Scale software in a rolling manner with data grids that contain live data, consider the following issues. The catalog service software version must be greater than or equal to the container server software versions. Upgrade all the catalog servers first with a rolling strategy. Read more about upgrading your deployment in the topic “Updating eXtreme Scale servers” on page 67.

Changing data model

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

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

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

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

Virtualization

Cloud computing and virtualization are popular emerging technologies. By default, eXtreme Scale insures that two shards for the same partition are never placed on

the same IP address as described in Policy 1. When you are deploying on virtual images, such as VMware, many server instances, each with a unique IP address, can be run on the same physical server. To ensure that replicas can only be placed on separate physical servers, you can use zones to solve the problem. Group your physical servers into zones, and use zone placement rules to keep primary and replica shards in separate zones.

Zones for wide-area networks

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

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

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 *Configuring zones for replica placement* to get more information about some basic scenarios and how to configure your deployment policy accordingly.

Zone-preferred routing gives WebSphere 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 `cxszones.dita` 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.

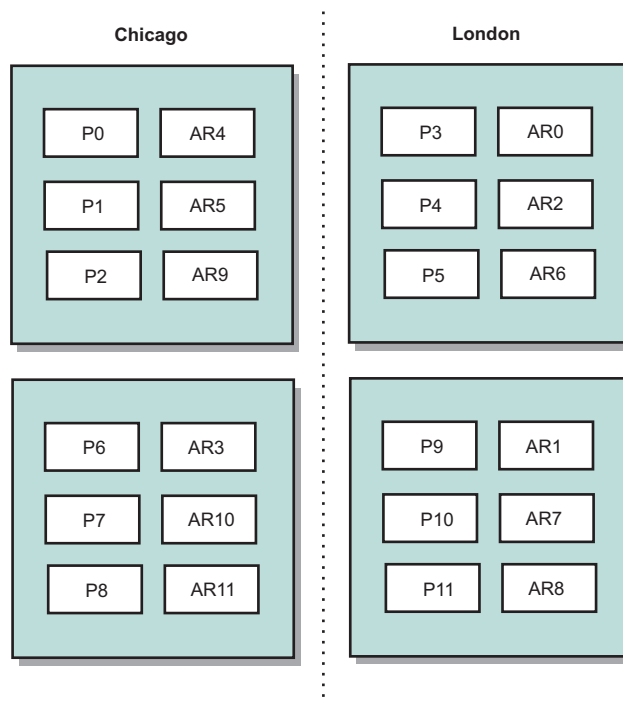


Figure 15. Primaries and replicas in zones

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.

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

- **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 “**startOgServer** script” on page 356.
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” on page 364.

- **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” on page 188 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.

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.

Viewing zone information with the xsadmin utility

You can use the **xsadmin** sample 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” on page 181 for more information.

About this task

You can determine information about your configuration related to zone settings by using the **xsadmin** utility that ships with the product.

Procedure

Use the **xsadmin** utility to determine information about the shards of data

1. Run the following command: `xsadmin.bat -containers -empties`
2. The utility displays output:

```
*** Show all online containers for grid - Grid & mapset - mapSet
Host: 9.76.25.172
  Container: serverA0_C-3, Server:serverA0, Zone:zoneA
    P:0 Primary
```

```

P:1 SynchronousReplica
P:2 SynchronousReplica
Container: serverA1_C-3, Server:serverA1, Zone:zoneA
P:1 Primary
P:2 Primary
P:0 SynchronousReplica
Container: serverB0_C-4, Server:serverB0, Zone:zoneB
P:1 AsynchronousReplica
P:2 AsynchronousReplica
Container: serverB1_C-1, Server:serverB1, Zone:zoneB
P:0 AsynchronousReplica

```

This output is from the multiple data centers scenario: Primary and synchronous replicas are in the same zone, asynchronous replicas are in a remote zone. Four container servers: serverA0, serverA1, serverB0, and serverB1 exist in two zones: zoneA and zoneB. The Grid data grid is deployed, and it has the mapSet map set. The map set has three partitions defined. Production mode requires many more than three partitions, which are sufficient for development mode purposes. Primary and synchronous replica shards exist in the same zoneA zone, and asynchronous replicas exist in the zoneB zone.

Example

You can also run a simpler scenario by using the getting started sample: *wxs_install_root/ObjectGrid/gettingstarted*. See Chapter 1, “Running the getting started sample application,” on page 1 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 **xsadmin -mapsizes** command shows the number of items in the data grid.
4. Show active containers with the **-empties** parameter for `xsadmin`. The utility displays output:

```

*** Show all online containers for grid - Grid & mapset - mapSet
Host: 9.77.129.191
Container: serverA0_C-0, Server:serverA0, Zone:zoneA
P:0 Primary
P:1 Primary
P:2 Primary
Container: serverA1_C-0, Server:serverA1, Zone:zoneA

```

In this example, serverA1 currently has no shards. The example is based on a deployment using the gettingstarted sample deployment policy XML file with three partitions but no zone rules. Two container servers were started and assigned to zoneA. Although the deployment policy XML file is configured for one synchronous replica, none is allocated because of Rule 2. In this case, you need another zone in which to allocate the replica.

- 5.

- To stop all container servers, use the following command: `xsadmin.bat -teardown`. The utility displays a list of containers to be stopped and prompts you to continue: Yes or No.
- To stop all container servers and the catalog server: `xsadmin.bat -teardown <catalog_server_name>`
The name of the catalog server in the gettingstarted sample is cs0.

Configuring 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.

About this task

Configuring failover varies depending 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.
You can configure heartbeat intervals on the command line by using the **-heartbeat** parameter in the **startOgServer** script file. Set this parameter to one of the following values:

Table 11. Heartbeat intervals

Value	Action	Description
0	Typical (default)	Failovers are typically detected within 30 seconds.
-1	Aggressive	Failovers are typically detected within 5 seconds.
1	Relaxed	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 6.0.2 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.x and 7.x:**

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.

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” on page 197 for the corresponding deployment policy XML schema.

Elements in the deploymentPolicy.xml file

```
(1) <deploymentPolicy>
(2)   <objectgridDeployment objectGridName="blah">
(3)     <mapSet
(4)       name="mapSetName"
(5)       numberOfPartitions="numberOfPartitions"
(6)       minSyncReplicas="minimumNumber"
(7)       maxSyncReplicas="maximumNumber"
(8)       maxAsyncReplicas="maximumNumber"
(9)       replicaReadEnable="true|false"
(10)      numInitialContainers="numberOfInitialContainersBeforePlacement"
(11)      autoReplaceLostShards="true|false"
(12)      developmentMode="true|false"
(13)      placementStrategy="FIXED_PARTITION|PER_CONTAINER">
(14)       <map ref="backingMapReference" />
(15)     </mapSet>
(16)     <zoneMetadata>
(17)       <shardMapping
(18)         shard="shardType"
(19)         zoneRuleRef="zoneRuleRefName" />
(20)       <zoneRule
(21)         name="zoneRuleName"
(22)         exclusivePlacement="true|false" >
(23)         <zone name="ALPHA" />
(24)         <zone name="BETA" />
(25)         <zone name="GAMMA" />
(26)       </zoneRule>
(27)     </zoneMetadata>
(28)   </objectgridDeployment>
(30) </deploymentPolicy>
```

deploymentPolicy element (line 1)

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 (line 2)

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” on page 153 which you should couple with the deployment policy file for each ObjectGrid instance.

mapSet element (line 3)

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**

Specifies the name of the mapSet. This attribute must be unique within the objectgridDeployment element. (Required)

numberOfPartitions

Specifies the number of partitions for the mapSet element. The default value is 1. The number should 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. (Optional)

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 has not already been 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 have been 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 amongst 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 bringing a data grid online from a cold startup. (Optional)

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`, provided that the `minSyncReplicas` value can also be satisfied. After the `numInitialContainers` value has been met, each container server-started event can trigger a rebalance 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 rebalance after every container server start. Placement occurs only when you reach the `numInitialContainers` value specified in the `mapSet` element.

7.1.0.2+ If you ever need 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 `xsadmin -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 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 machine. 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 `FIXED_PARTITION`, where the number of primary shards that are placed across available container servers is equal to the number of partitions defined, increased by the number of replicas. The alternate strategy is `PER_CONTAINER`, where 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 placed on other container servers. (Optional)

map element (line 14)

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, as in the following code snippet. (Required)

zoneMetadata element (line 16)

You can place shards into zones. This function allows more control over 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 additional background, see `cxszones.dita`.

- **Number of occurrences:** Zero or one
- **Child elements:**
 - shardMapping
 - zoneRule

Attributes: None

shardMapping element (line 17)

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 (line 20)

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 using 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 mapped to this zone rule are also placed in that zone. Note that using an exclusive setting with three shards mapped to the same zone rule (primary, and two synchronous replicas) would require at least 3 zones in order for all shards to be placed. (Optional)

zone element (lines 23 to 25)

The zone element is used to name a zone within a zone rule. Each zone named should correspond to a zone name that is used to launch 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>
```

Although only two container servers are required to satisfy the replication settings, 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.

Because 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 onto another container server, provided that a container server is available to host the lost shard. Shards from the same partition cannot be placed on the same machine 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.

deploymentPolicy.xsd file

Use the deployment policy XML schema to create a deployment descriptor XML file.

See the “Deployment policy descriptor XML file” on page 192 for descriptions of the elements and attributes defined in the deploymentPolicy.xsd file.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema xmlns:dp="http://www.ibm.com/ws/objectgrid/deploymentPolicy"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.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" />
  </xsd:complexType>
</xsd:schema>
```

```

        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: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: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>

```

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 together comprise the catalog service. A container server is a Java virtual machine (JVM) that stores the application data for the data grid.

About this task

Catalog and container servers can start in WebSphere Application Server processes, as stand-alone Java SE processes, or by embedding the servers in Java SE applications. How you configure the catalog and container servers depends on your topology.

Catalog servers

- **Stand-alone catalog servers:**

Configure stand-alone catalog servers with a server properties file. Control the life cycle of a catalog server with the **startOgServer** and **stopOgServer** scripts or by using the embedded server API.

- **Catalog servers that start 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.

Container servers

- **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 **startOgServer** and **stopOgServer** scripts or by using 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.

Use the following topics to configure your catalog and container servers:

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.

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 classpath.

- **For servers that run in WebSphere Application Server:**

- Use a well-named file in the classpath, 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 classpath. 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 classpath:
`-Dobjectgrid.server.props=file_name`

- **For stand-alone servers:**
 - Use a well-named file in the classpath, 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 classpath. The name that is used follows:
`objectGridServer.properties`
 - Specify the server properties file as a parameter when you run the **startOgServer** command. You can override these properties manually to specify a file in the system current directory:
`-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

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

minThreads

Specifies the minimum number of threads used by the internal thread pool in the run time for built-in evictors and DataGrid operations.

Default: 10

maxThreads

Specifies the maximum number of threads used by the internal thread pool in the run time for built-in evictors and DataGrid operations.

Default: 50

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.

Default: `*=all=disabled`

traceFile

Specifies a file name to write trace information. 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`

enableMBeans

Enables ObjectGrid container Managed Beans (MBean). This property applies to both the container server and the catalog service.

Default: `true`

serverName

Sets the server name that is used to identify the server. This property applies to both the container server and the catalog service.

zoneName

Set the name of the zone to which the server belongs. This property applies to both the container server and the catalog service.

haManagerPort

Synonymous with peer port. Specifies the port number the high availability manager uses. If this property is not set, the catalog service generates an available port automatically. This property applies to both the container server and the catalog service.

listenerHost

Specifies the host name to which the Object Request Broker (ORB) should bind. This property applies to both the container server and the catalog service.

If your configuration involves multiple network cards, set the listener host and port to let the Object Request Broker in the JVM know the IP address to which to bind. For catalog and container servers, specify the listener host and port in the server properties file. Neglecting to specify which IP address to use produces symptoms such as connection time outs, unusual API failures, and clients that seem to hang.

listenerPort

Specifies the port number to which the Object Request Broker (ORB) should bind. This property applies to both the container server and the catalog service.

JMXServicePort

Specifies the port number on which the MBean server should listen. This property applies to both the container server and the catalog service.

Container server properties**statsSpec**

Specifies the stats specification for the container server.

Example:

```
all=disabled
```

memoryThresholdPercentage

Sets the memory threshold for memory-based eviction. The percentage specifies the maximum heap that should 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. See `MemoryPoolMXBean` interface in the Java API specification for more information. However, eviction occurs only if eviction is enabled on an evictor. To enable memory based eviction, see the information about evictors in the *Product Overview*. This property only applies to a container server.

catalogServiceEndpoints

Specifies the end points to connect to the catalog service domain. This value should be in the form `host:port<,host:port>` where the host value is the `listenerHost` value and the port value is the `listenerPort` value of the catalog server. This property only applies to a container server.

Catalog service properties

domainName

Specifies the domain name that is used to uniquely identify this catalog service domain to clients when routing to multiple domains. This property only applies to the catalog service.

enableQuorum

Enables quorum for the catalog service. Quorum is used to ensure that a majority of the catalog service domain is available before allowing modification to the placement of partitions on available container servers. To enable quorum, set the value to true or enabled. The default value is disabled. This property only applies to the catalog service.

catalogClusterEndPoints

Specifies the 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 format:

```
serverName:hostName:clientPort:peerPort<serverName:hostName:clientPort:peerPort>
```

This property only applies to the catalog service.

heartBeatFrequencyLevel

Specifies how often heartbeats occur. The heartbeat frequency level is a trade-off between use of resources and failure discovery time. The more frequently heartbeats occur, more resources are used, but failures are discovered more quickly. This property applies only to the catalog service. Use one of the following values:

- 0: Specifies a heartbeat level at a typical rate. With this value, failover detection occurs at a reasonable rate without overusing resources. (Default)
- -1: Specifies an aggressive heartbeat level. With this value, failures are detected more quickly, but also uses additional processor and network resources. This level is more sensitive to missing heartbeats when the server is busy.
- 1: 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.

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 basic the properties and security properties.

General security properties

securityEnabled

Enables the container server security when set to true. The default value is false. This property should match the securityEnabled property that is specified in the objectGridSecurity.xml file that is provided to the catalog server.

credentialAuthentication

Indicates whether this server supports credential authentication. Chose 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 “Application client authentication” on page 435 for details about credential authentication.

Transport layer security settings

transportType

Specifies the server transport type. Use one of the following values:

- TCP/IP: Indicates that the server only supports TCP/IP connections.
- 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

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.

Valid values: IBMJSSE2, IBMJSSE, IBMJSSEFIPS, and so on.

protocol

Indicates the type of security protocol to use for the client. Set this protocol value based on which Java Secure Socket Extension (JSSE) provider you use. If you indicate a value that is not valid, a security exception results that indicates that the protocol value is incorrect.

Valid values: SSL, SSLv2, SSLv3, TLS, TLSv1, and so on.

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.

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.

keyStore

Specifies a fully qualified path to the keystore file.

Example:

etc/test/security/client.private

trustStore

Specifies a fully qualified path to the truststore file.

Example:

etc/test/security/server.public

keyStorePassword

Specifies the string password to the keystore. You can encode this value or use the actual value.

trustStorePassword

Specifies a string password to the truststore. You can encode this value or use the actual value.

clientAuthentication

If the property is set to true, the SSL client must be authenticated. Authenticating the SSL client is different from the client certificate authentication. Client certificate authentication means authenticating a client to a user registry based on the certificate chain. This property ensures that the server connects to the right client.

SecureTokenManager

The SecureTokenManager setting is used for protecting the secret string for server mutual authentications and for protecting the single sign-on token. “Data grid security” on page 433

secureTokenManagerType

Specifies the type of SecureTokenManager setting. You can use one of the following settings:

- none: Indicates that no secure token manager is used.
- default: Indicates that the 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.

customTokenManagerClass

Specifies the name of your SecureTokenManager implementation class, if you have specified the SecureTokenManagerType property value as custom. The implementation class must have a default constructor to be instantiated.

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.

Secure token keystore configuration**secureTokenKeyStore**

Specifies the file path name for the keystore that stores the public-private key pair and the secret key.

secureTokenKeyStoreType

Specifies the keystore type, for example, JCKES. You can set this value based on the Java Secure Socket Extension (JSSE) provider that you use. However, this 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 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 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.

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.

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” on page 45 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” on page 199 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” on page 363 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.

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” on page 26 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” on page 351 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.

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:

```
wsadmin>$AdminTask help XSDomainManagement
```

Commands

The administrative tasks for catalog service domains include the following commands:

- “createXSDomain” on page 208
- “deleteXSDomain” on page 211
- “getDefaultXSDomain” on page 211
- “listXSDomains” on page 211

- “modifyXSDomain” on page 212
- “testXSDomainConnection” on page 216
- “testXSSEServerConnection” on page 216

createXSDomain

The **createXSDomain** command registers a new catalog service domain.

Table 12. createXSDomain command arguments

Argument	Description
-name (required)	Specifies the name of the catalog service domain that you want to create.
-default	Specifies if the catalog service domain is the default for the cell. The default value is true. (Boolean: set to true or false)
-properties	Specifies custom properties for the catalog service domain.

Table 13. defineDomainServers step arguments

Argument	Description
<i>name_of_endpoint</i>	Specifies the name of the catalog service endpoint. <ul style="list-style-type: none"> • For existing application servers: The name of the endpoint must be in the following format: <i>cell_name\node_name\server_name</i> • 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.
<i>custom_properties</i>	Specifies custom properties for the catalog service domain endpoint. If you do not have any custom properties, use a set of double quotes ("") for this argument.

Table 13. *defineDomainServers* step arguments (continued)

Argument	Description
<i>endpoint_ports</i>	<p>Specifies the port numbers for the catalog service domain endpoint. The ports must be specified in the following order: <i><client_port></i>,<i><listener_port></i></p> <p>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 only and can be set to any port that is not being used elsewhere.</p> <p>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.</p> <p>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 <code>BOOTSTRAP_ADDRESS</code> port configuration.</p>

7.1.0.2+

Table 14. *configureClientSecurity* step arguments

Argument	Description
-securityEnabled	<p>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)</p>
-credentialAuthentication (optional)	<p>Indicates if credential authentication is enforced or supported.</p> <p>Never No client certificate authentication is enforced.</p> <p>Required Credential authentication is always enforced. If the server does not support credential authentication, the client cannot to connect to the server.</p> <p>Supported (Default) Credential authentication is enforced only if both the client and server support credential authentication.</p>

Table 14. *configureClientSecurity* step arguments (continued)

Argument	Description
-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 <code>com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredentialGenerator</code> implementation class, so the client retrieves the security tokens from the thread.
-credentialGeneratorProps	Specifies the properties for the <code>CredentialGenerator</code> implementation class. The properties are sent to the object with the <code>setProperties(String)</code> method. The credential generator properties value is used only when a value is specified for the Credential generator class field.

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 15. *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)
-properties	Specifies custom properties for the catalog service domain.

Table 16. *modifyEndpoints* step arguments

Argument	Description
<i>name_of_endpoint</i>	Specifies the name of the catalog service endpoint. <ul style="list-style-type: none"> • For existing application servers: The name of the endpoint must be in the following format: <i>cell_name\node_name\server_name</i> • 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.

Table 16. *modifyEndpoints* step arguments (continued)

Argument	Description
<i>endpoint_ports</i>	<p>Specifies the port numbers for the catalog service domain endpoint. The endpoints must be specified in the following order: <i><client_port>,<listener_port></i></p> <p>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 only and can be set to any port that is not being used elsewhere.</p> <p>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.</p> <p>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 <code>BOOTSTRAP_ADDRESS</code> port configuration.</p>

Table 17. *addEndpoints* step arguments

Argument	Description
<i>name_of_endpoint</i>	<p>Specifies the name of the catalog service endpoint.</p> <ul style="list-style-type: none"> • For existing application servers: The name of the endpoint must be in the following format: <i>cell_name\node_name\server_name</i> • 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.

Table 17. *addEndpoints* step arguments (continued)

Argument	Description
<i>custom_properties</i>	Specifies custom properties for the catalog service domain endpoint. If you do not have any custom properties, use a set of double quotes ("") for this argument.
<i>endpoint_ports</i>	<p>Specifies the port numbers for the catalog service domain endpoint. The endpoints must be specified in the following order: <code><client_port>,<listener_port></code></p> <p>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 only and can be set to any port that is not being used elsewhere.</p> <p>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.</p> <p>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.</p>

Table 18. *removeEndpoints* step arguments

Argument	Description
<i>name_of_endpoint</i>	Specifies the name of the catalog service endpoint to delete.

7.1.0.2+

Table 19. *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 <code>com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredentialGenerator</code> implementation class, so the client retrieves the security tokens from the thread.
-credentialGeneratorProps	Specifies the properties for the <code>CredentialGenerator</code> implementation class. The properties are sent to the object with the <code>setProperties(String)</code> method. The credential generator properties value is used only when a value is specified for the Credential generator class field.

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]]]')
```
- **7.1.0.2+** 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 true, otherwise, connection error information is returned.

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]')
```

testXSSEServerConnection

The **testXSSEServerConnection** 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.

7.1.0.2+ 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:

Batch mode example usage

- Using Jacl:

```
$Admintask testXSSTestServerConnection {-host xhost1.ibm.com -listenerPort 2809}
```
- Using Jython string:

```
AdminTask.testXSSTestServerConnection('[-host xshost3.ibm.com -listenerPort 2809]')
```

Interactive mode example usage

- Using Jacl:

```
$AdminTask testXSSTestServerConnection {-interactive}
```
- Using Jython string:

```
AdminTask.testXSSTestServerConnection ('[-interactive]')
```

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**.

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 **start0gServer** 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 only 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 inherited from the `BOOTSTRAP_ADDRESS` port configuration.

Status

Table 20. 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 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.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 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.

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 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” on page 206 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” on page 45 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” on page 199 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.

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” on page 153 for more information.

- A deployment descriptor XML file named `objectGridDeployment.xml`. See “Deployment policy descriptor XML file” on page 192 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” on page 258 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.
```

```
CW0BJ3013E: 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 define a `numInitialContainers` attribute in the `objectGridDeployment.xml` file. If you define the `numInitialContainers` attribute, then placement starts when that number of containers has started.

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)`.

Configuring the quorum mechanism

The quorum mechanism is configured for each catalog service. You must enable the quorum mechanism on all of the catalog servers in the catalog service domain.

Before you begin

Before you enable the quorum mechanism, you must configure a topology that supports this type of configuration. The configuration must support the following requirements:

- **Flat IP address space:** Any addressable element on the network must be able to connect to any other addressable element on the network unimpeded. You must use a flat IP address naming space. 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.
- **Number of catalog servers:** You must start at least one catalog server for each data center in the configuration.
- **Heartbeat interval setting:** If you do not define the heartbeat interval, the default value is 30 seconds. WebSphere eXtreme Scale checks on the JVMs in a single zone at the defined interval. For example, if a heartbeat on a container server is missed, and quorum is established, a failover event occurs to place a new container server. See “Configuring the heartbeat interval setting for failover detection” on page 190 for more information.
- **Transport security:** Because data centers are normally deployed in different geographical locations, you might want to enable transport security between the data centers for security reasons. Read about transport layer security in the *Administration Guide*.

About this task

The quorum mechanism is disabled by default. Enable the quorum mechanism in the following scenarios:

- When your catalog service domain spans a network that is unpredictable or unstable. This type of network might span multiple data centers.
- When you want to prevent the data grid from self-healing during a brownout on the network, and instead temporarily pause data grid operations from occurring.

You can leave the quorum mechanism disabled if your catalog service domain is contained within a single data center, or is on a local area network (LAN). In this type of configuration, default heart beating is used and brownouts are assumed to be shorter than 10 seconds. Because the detection period is approximately 30 seconds, any short brownouts that occur do not cause placement changes to occur in the data grid.

If you enable quorum, all the catalog servers must be available and communicating with the data grid to conduct placement operations. If a network brownout occurs, placement is paused until all the catalog servers are available. If a data center failure occurs, manual administrator actions are required to remove the failed catalog server from the quorum.

Procedure

1. **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.

```
catalogClusterEndpoints=cat0:cat0.domain.com:6600:6601,  
cat1:cat1.domain.com:6600:6601  
catalogServiceEndpoints= cat0.domain.com:2809, cat1.domain.com:2809  
enableQuorum=true
```

Figure 16. *objectGridServer.properties* file

For more information about configuring the properties file, see “Server properties file” on page 199.

- **Pass the `-quorum enabled` flag on the `startOgServer` 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 “`startOgServer` script” on page 356.

2. **Start container servers in the same zone.**

When you are running a data grid across data centers, the servers must use the zone information to identify the data center in which they reside. Setting the zone on the container servers allows WebSphere eXtreme Scale to monitor health of the container servers that are scoped to the data center, minimizing cross-data-center traffic. The container server JVMs in a core group must never span multiple LANs that are connected with links, like in a wide area network.

See “Defining zones for container servers” on page 184 for more information about defining zones for container servers.

Container server JVMs are tagged with a zone identifier. The data grid of container JVMs is automatically broken in to small core groups of JVMs. A core group only includes JVMs from the same zone. JVMs from different zones are never in the same core group.

A core group aggressively tries to detect the failure of its member JVMs.

Results

By setting the quorum mechanism to be enabled on the catalog servers within a catalog service domain, all the catalog servers must be available for data grid placement operations to occur. In the event of a short network brownout, placement operations are temporarily stopped until all the catalog servers in the quorum are available.

You can add additional catalog servers to the quorum by repeating these steps.

What to do next

- You can remove a catalog server from the quorum by stopping the catalog server with the administrative method that is required by the configuration. When a catalog server is stopped through administrative actions, quorum is automatically reestablished among the remaining catalog servers, and placement can continue. If you restart the catalog server with the steps described in this topic, the catalog server can rejoin the quorum.
- If a long-term or permanent failure of a catalog server that is in the currently defined quorum occurs, you must override the quorum mechanism so that placement can continue. See “Managing data center failures” on page 371 for more information about overriding the quorum mechanism.

Best practice: Clustering the catalog service

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.

Number of catalog servers

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.

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.
- **Base WebSphere Application Server:** You can start the catalog service on multiple stand-alone nodes. By defining multiple profiles on the same installation image with the profile management tool, you can create a set of stand-alone nodes that each have unique ports assigned. In each application server, define the catalog service domain. You can specify any other application servers by adding remote servers to the configuration. After you create this configuration on all of the stand-alone servers, you can start the set of base application servers in parallel by running the **startServer** script or by using a Windows service to start the servers.

Configuring multi-master replication topologies

With the multi-master asynchronous replication feature, you use links to interconnect a set of domains, then eXtreme Scale synchronizes the domains, using replication over the links. Because you define the links among domains, you can construct almost any topology. Define links in the properties files of the catalog servers for each domain, or define links at runtime using Java Management Extensions (JMX) programs or the **xsadmin** command line utility. However you create links, the set of current links for a domain is stored in the catalog service, enabling you to add and remove links without restarting the domain (data grid).

Before you begin

Multi-master data grid replication topologies (AP) introduces you to the characteristics of various multi-master replication topologies. The following procedure describes the mechanics of configuring various links among domains to

form the topology you choose. The section after the procedure provides some examples to illustrate how to set up specific topologies, such as a hub and spoke formation.

Also, 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 domain A and domain B to replicate the data in the map set between the catalog service domains.
- Is a FIXED_PARTITION data grid. PER_CONTAINER data grids cannot be replicated.
- Has the same number of partitions in each catalog service domain, but is unrestricted with respect to the number or type of replicas.
- Has the same data types replicated in each catalog service domain.
- Contains the same maps and dynamic map templates in each catalog service domain.

Any map sets with the preceding characteristics will be replicated after the catalog service domains in the topology have been started.

Procedure

1. Define links in the properties file for the catalog server of each domain in the topology, for bootstrap purposes.

The property file is detected automatically if you name it `objectGridServer.properties` (case sensitive on some systems) and place it on the classpath used when starting a catalog service instance. You also can specify its location on the command line to the `startOgServer` script, using the **-serverProps** parameter.

Because the property names are case sensitive, take care on capitalization when updating the property file.

Local Domain name

Specify the name of "this" domain, such as domain A:

For example:

```
domainName=A
```

An optional list of foreign domain names

Specify the names of "other" domains in the multi-master replication topology, such as domain B:

```
foreignDomains=B
```

An optional list of endpoints for the foreign domain names

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.

2. Use the **xsadmin** command utility or JMX programming to add or remove links at runtime.

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 `xsadmin` command line utility includes several options for working with links.

The `xsadmin` utility connects to a catalog service and thus a single domain. Therefore, `xsadmin` 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 new link, for example:

```
xsadmin -ch host -p 1099 -establishLink dname fdHostA:2809,fdHostB:2809
```

The command establishes a new link between the 'local' domain and the foreign domain named "dname" whose catalog service is running at `fdHostA:2809` and `fdHostB:2809`. The local domain has a catalog service JVM with a JMX address of `host:1099`. Specify all catalog endpoints from the foreign domain so that fault tolerance connectivity to the domain is possible. It is not recommended to use a single `host:port` pair for the catalog service of the foreign domain.

It does not matter which local catalog service JVM the `xsadmin` specifies using `-ch` and `-p`. Any catalog JVM will work. If the catalog 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:

```
xsadmin -ch host -p 1099 -dismissLink dname
```

The command connects to the catalog service for a domain and instructs it to stop replicating to a specific domain. A link only needs to be dismissed from one side.

Example

Suppose that you want to configure a two-domain setup involving Domains A and B.

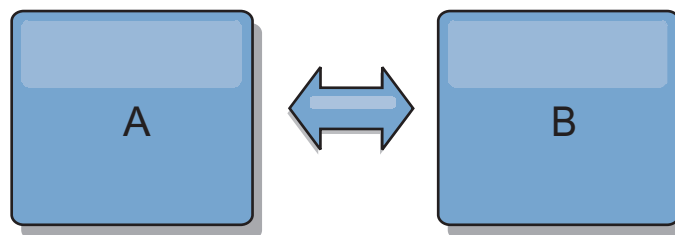


Figure 17. Link between domains

Here is the property file for the catalog server in domain A:

```
domainName=A  
foreignDomains=B  
B.endPoints=hostB1:2809, hostB2:2809
```

Here is the property 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 will be 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
- Is a FIXED_PARTITION data grid (PER_CONTAINER data grids 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 mapset name, map names, and dynamic map templates as other data grids in the domain

Note that the replication policy of a domain will be 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.

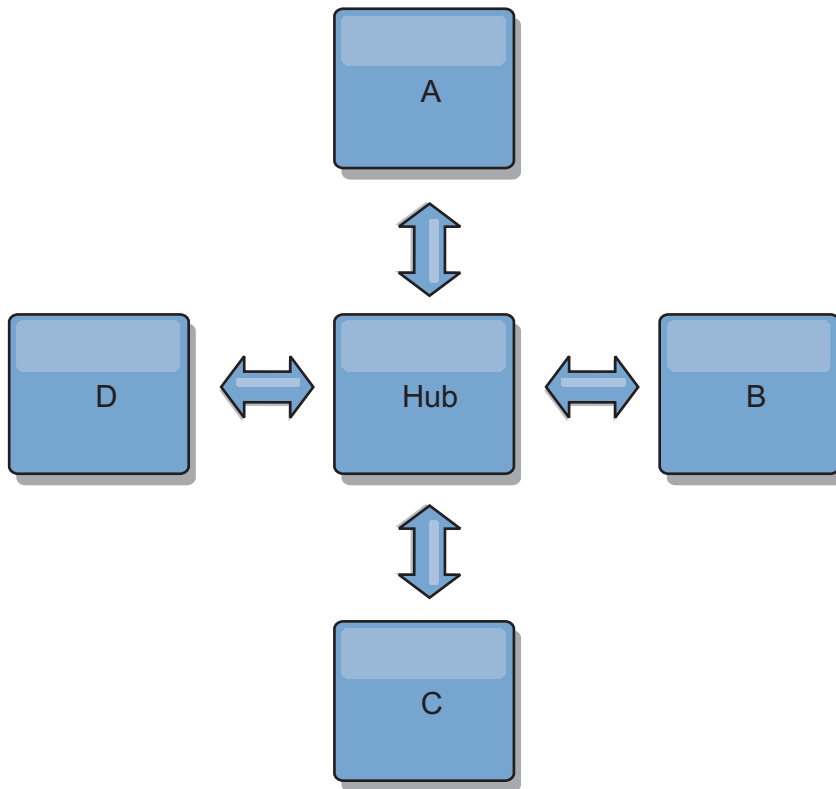


Figure 18. Hub and spoke topology

The hub and four spokes (domains A, B, C, and D) would have catalog server property files like the following examples.

```
domainName=Hub
```

The first spoke would have the following properties:

```
domainName=A
foreignDomains=Hub
Hub.endPoints=hostH1:2809, hostH2:2809
```

The second spoke would have the following properties:

```
domainName=B
foreignDomains=Hub
Hub.endPoints=hostH1:2809, hostH2:2809
```

The third spoke would have the following properties:

```
domainName=C
foreignDomains=Hub
Hub.endPoints=hostH1:2809, hostH2:2809
```

The fourth spoke would have the following properties:

```
domainName=D
foreignDomains=Hub
Hub.endPoints=hostH1:2809, hostH2:2809
```

Configuring ports

WebSphere eXtreme Scale is a distributed cache that requires opening ports to communicate with the Object Request Broker (ORB) and Transmission Control Protocol (TCP) stack among Java virtual machine (JVM) and other servers.

Planning for network ports

WebSphere eXtreme Scale is a distributed cache that requires opening ports to communicate with the Object Request Broker (ORB) and Transmission Control Protocol (TCP) stack among Java virtual machines. Plan and control your ports, especially in an environment that has a firewall, and when you are using a catalog service and containers on multiple ports.

Important: When you are specifying port numbers, avoid setting ports that are in the ephemeral range for your operating system. If you use a port that is in the ephemeral range, port conflicts could 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 for catalog servers to access catalog service data. In WebSphere Application Server, this port is set through the catalog service domain configuration.

listenerPort

Defines the ORB listener port for containers and clients to communicate with the catalog service through the ORB. In WebSphere Application Server, the listenerPort is inherited by the BOOTSTRAP_ADDRESS port configuration.

Default: 2809

Container servers

The WebSphere eXtreme Scale container servers also require several ports to operate. By default, the eXtreme Scale container server generates its HA manager port and ORB listener port automatically with dynamic ports. 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 peer port. (Required for WebSphere Application Server environments only.)

listenerPort

Defines the ORB listener port for containers and clients to communicate with the catalog service through the ORB.

Default: 2809

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 several other commands. Use the **listenerPort** property in the client properties file to specify the port in which the client listens for callbacks from the server.

haManagerPort

Specifies the peer port. (Required for WebSphere Application Server environments only.)

jvmArgs (optional)

Specifies a list of Java virtual machine (JVM) arguments. When security is enabled, you must use the following argument to configure the Secure Socket Layer (SSL) port: `-jvmArgs Dcom.ibm.CSI.SSLPort=<sslPort>`

listenerPort

Defines the ORB listener port for containers and clients to communicate with the catalog service through the ORB.

Default: 2809

Ports in WebSphere Application Server

- The **listenerPort** value is inherited from the **BOOTSTRAP_ADDRESS** value for each WebSphere Application Server application server.
- The **haManagerPort** and **peerPort** values inherited from the **DCS_UNICAST_ADDRESS** value for each WebSphere Application Server application server.

You can define a catalog service domain in the administrative console as described in “Creating catalog service domains in WebSphere Application Server” on page 206.

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: **Servers > Server Types > WebSphere Application Servers > server_name > Ports > port_name**

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.sh` or `startOgServer.bat` scripts or to the `java` command. 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.

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 <server:host:clientPort:peerPort,server:host:clientPort:peerPort>
```

The catalog service end points can also be set using the catalogClusterEndpoints catalog server property. The Object Request Broker (ORB) 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>  
-listenerPort <port>
```

The listener port and listener host can also be set using the listenerHost and listenerPort server property.

The JMX service port is used for communication from JMX clients. To specify the JMX service port, use the following command-line option:

```
-JMXServicePort <jmxPort>
```

The JMX service port can also be set using the JMXServicePort server property. 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>
```

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
```

Catalog service Java virtual machine (JVM) end points

WebSphere eXtreme Scale uses IIOP mainly to communicate between Java virtual machines. The catalog service Java virtual machines are the only Java virtual machines that require the explicit configuration of ports for the IIOP services and group services ports. The internal ports are specified using the **-catalogServiceEndpoints** command line option:

```
-catalogServiceEndpoints <server:host:port:port,server:host:port:port>
```

The IIOP ports are configured using the following command line options:

```
-listenerHost <host_name>  
-listenerPort <port>  
-JMXServicePort <jmxPort>
```

When each catalog service JVM is started, specify the complete set of catalog service endpoints along with a single listener port for that JVM.

2. Start container endpoints

The following command starts a container JVM to use with the example catalog service:

```
./startOgServer.sh c0 -catalogServiceEndpoints hostA:2809,hostB:2809
```

The container 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. 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 option to specify the ORB port. To specify the listener port and listener host, use the following command-line options:

```
-listenerHost <host_name>  
-listenerPort <port>
```

The listener port and listener host can also be set using the listenerHost and listenerPort server property.

The listener host is used to bind the ORB to a specific network adapter.

To specify the HA manager port, use the following command-line option:

```
-haManagerPort <port>
```

The listener port and listener host can also be set using the HAManagerPort server property.

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>
```

3. Start client endpoints

Clients only need to know the catalog service listener end points. Clients retrieve end points for container 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 should 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:

```
-Dcom.ibm.CSI.SSLPort=<sslPort>
```

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” on page 232. For details on configuring a catalog service domain, see “High-availability catalog service” on page 96.

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” on page 232. 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” on page 222.

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(catalogServerAddresses, 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.

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 will automatically choose one, which may result in connection failures or slower performance.

For catalog or container servers, you must set the listener host and listener port in one of the following ways:

- server properties
- command-line parameter on the `startOgServer.sh` | `bat` script.

For clients, you cannot use the command line, and must use client properties.

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.

Before you begin

Restriction: 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.

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 `-1` (unlimited) with the command: `ulimit -n -1`.

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
```

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.

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 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 using WebSphere eXtreme Scale security, set the com.ibm.CORBA.NoLocalInterceptors property value to false. The security infrastructure uses interceptors for authentication.

Using 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. The `orb.properties` file is in the `java/jre/lib` directory. See “ORB properties” on page 237 for descriptions of the properties and settings.

Procedure

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.”

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.

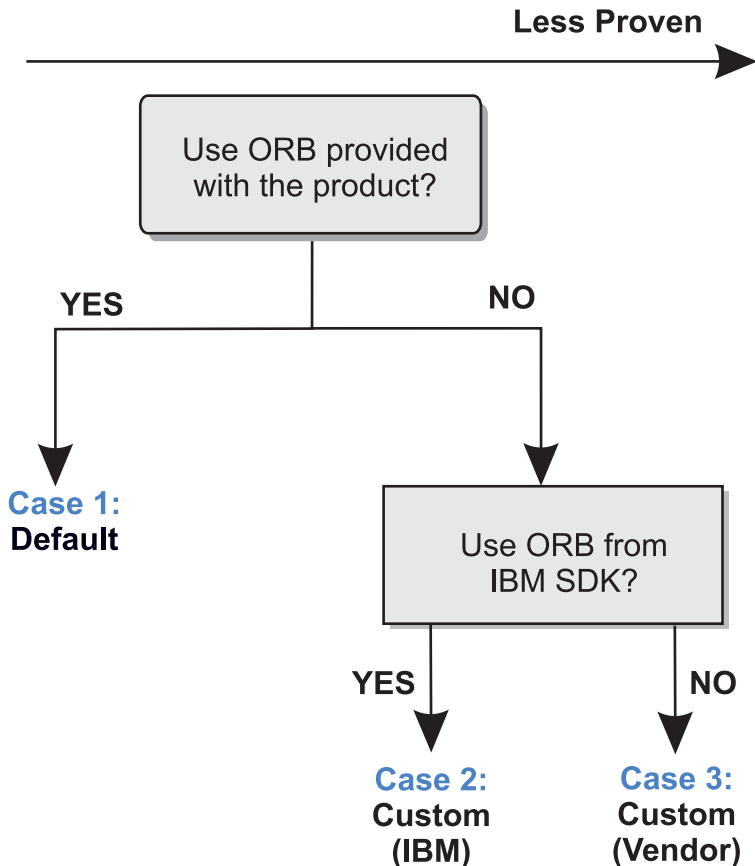


Figure 19. Choosing an ORB

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 Sun Microsystems 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.

Using IBM SDK Version 5 (or later) requires less configuration effort than does IBM SDK Version 1.4.2.

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 Sun Microsystems 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*

Tip: If using a custom ORB from an external vendor (**Case 3**), these additional JAR files might be required:

- *ibmext.jar*

2. Specify the *custom_ORB_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* under the existing endorsed directory. By placing the *custom_ORB_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* 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 *start0gServer* 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*

- Configure client or server processes to use IBM SDK, Version 1.4.2 (**Case 2**). If your environment contains a Version 1.4.2 SDK, integrate the IBM ORB into the specified SDK.
 1. Download and extract the ORB from an IBM SDK, Version 1.4.2.
If no IBM SDK is available for your platform, download and extract the IBM Developer Kit for Linux, Java Technology Edition. See IBM developer kits.
 2. Copy the ORB JAR files to the target SDK. Copy the `java/jre/lib/ibmorb.jar` and `java/jre/lib/ibmorbapi.jar` files to the `java/jre/lib/ext` directory on the target SDK.
 3. Update the ORB properties. Create or edit the `orb.properties` file, which is in the `java/jre/lib` directory of the SDK. Add the following properties or verify that the following properties exist in the file:


```
org.omg.CORBA.ORBClass=com.ibm.CORBA.iiop.ORB
org.omg.CORBA.ORBSingletonClass=com.ibm.rmi.corba.ORBSingleton
```

 For descriptions of the properties and settings, see “ORB properties” on page 237.
 4. Ensure that the XML parser is available.
 - Download Xerces2 Java 2.9 from The Apache Xerces Project - Downloads.
 - Locate the `xercesImpl.jar` and `xml-apis.jar` files.
 - Copy the files to the `lib/ext` directory.

Configuring clients

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. 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 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

You can override the following plug-ins on a client:

- **ObjectGrid plug-ins**
 - TransactionCallback plug-in
 - ObjectGridEventListener plug-in
- **BackingMap plug-ins**
 - Evictor plug-in
 - MapEventListener plug-in
 - numberOfBuckets attribute
 - ttlEvictorType attribute
 - timeToLive attribute

Client properties file

You can 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 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`
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

7.1+ listenerHost

Specifies the host name to which the Object Request Broker (ORB) binds.

For a multiple network card configuration, set the listener host and port to let the Object Request Broker in the JVM know the IP address on which to bind. For the client, use the client properties file. If you do not specify which IP address to use, the following problems might occur: connection timeouts, unusual API failures, and clients that seem to hang.

7.1+ listenerPort

Specifies the port number to which the Object Request Broker (ORB) binds.

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

requestRetryTimeout

Specifies how long to retry a request (in milliseconds). Use one of the following 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)
- A value over 0 indicates the request entry timeout value in milliseconds. Exceptions that cannot succeed even if tried again such as a DuplicateException exception are returned immediately. 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 should match with the securityEnabled setting in the WebSphere eXtreme Scale server properties file. If the settings do not match, an exception results.

Default: false

Credential authentication configuration properties

credentialAuthentication

Specifies the client credential authentication support. Use one of the following 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

credentialGeneratorClass

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.

Default: no value

credentialGeneratorProps

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.

Transport layer security configuration properties

transportType

Specifies the client transport type. The possible values are:

- 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.

Default: no value

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.

Valid values: IBMJSSE2, IBMJSSE, IBMJSSEFIPS, and so on.

protocol

Indicates the type of security protocol to use for the client. Set this protocol value based on which Java Secure Socket Extension (JSSE) provider you use. If you indicate a value that is not valid, a security exception results that indicates that the protocol value is incorrect.

Valid values: SSL, SSLv2, SSLv3, TLS, TLSv1, and so on.

keyStoreType

Indicates the type of keystore. If you indicate a value that is not valid, a runtime security exception occurs.

Valid values: JKS, JCEK, PKCS12, and so on.

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.

keyStore

Specifies a fully qualified path to the keystore file.

Example:

`etc/test/security/client.private`

trustStore

Specifies a fully qualified path to the truststore file.

Example:

`etc/test/security/server.public`

keyStorePassword

Specifies the string password to the keystore. You can encode this value or use the actual value.

trustStorePassword

Specifies a string password to the truststore. You can encode this value or use the actual value.

Configuring clients with WebSphere eXtreme Scale

You can configure a WebSphere eXtreme Scale client based on your requirements such as the need to override settings.

Override plug-ins

You can override the following plug-ins on a client:

- **ObjectGrid plug-ins**

- TransactionCallback plug-in
- ObjectGridEventListener plug-in
- **BackingMap plug-ins**
 - Evictor plug-in
 - MapEventListener plug-in
 - numberOfBuckets attribute
 - ttlEvictorType attribute
 - timeToLive attribute

Configure the client with XML

An ObjectGrid XML file can be used to alter settings on the client side. To change the settings on a WebSphere eXtreme Scale client, you must create an ObjectGrid XML file that is similar in structure to the file that was used for the WebSphere eXtreme Scale server.

Assume that the following XML file was paired with a deployment policy XML file, and these files were used to start a WebSphere eXtreme Scale 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="TransactionCallback"
        className="com.company.MyTxCallback" />
      <bean id="ObjectGridEventListener"
        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="customerPlugins">
      <bean id="Evictor"
        className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" />
      <bean id="MapEventListener"
        className="com.company.MyMapEventListener" />
    </backingMapPluginCollection>
    <backingMapPluginCollection id="orderPlugins">
      <bean id="MapIndexPlugin"
        className="com.company.MyMapIndexPlugin" />
    </backingMapPluginCollection>
  </backingMapPluginCollections>
</objectGridConfig>
```

On a WebSphere eXtreme Scale 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 running on the server. However, some of the settings can be overridden on the client, as follows:

1. Create a client-specific ObjectGrid instance.
2. Copy the ObjectGrid XML file that was used to open the server.
3. Edit the new file to customize for the client side.

- To set or update any of the attributes on the client, specify a new value or change the existing value.
- To remove a plug-in from the client, use the empty string as the value for the `className` attribute.
- To change an existing plug-in, specify a new value for the `className` attribute.
- You can also add any plug-in supported for a client override: `TRANSACTION_CALLBACK`, `OBJECTGRID_EVENT_LISTENER`, `EVICTOR`, `MAP_EVENT_LISTENER`.

4. Create a client with the newly created client-override XML file.

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="TransactionCallback"
        className="com.company.MyClientTxCallback" />
      <bean id="ObjectGridEventListener" 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="customerPlugins">
      <bean id="Evictor"
        className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" />
      <bean id="MapEventListener" className="" />
    </backingMapPluginCollection>
    <backingMapPluginCollection id="orderPlugins">
      <bean id="MapIndexPlugin"
        className="com.company.MyMapIndexPlugin" />
    </backingMapPluginCollection>
  </backingMapPluginCollections>
</objectGridConfig>
```

- The `TransactionCallback` 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 have changed
- Although a different `lockStrategy` attribute is specified, there is no effect because the `lockStrategy` attribute is not supported for a client override.

To create the CompanyGrid client using the `companyGridClientSide.xml` file, pass the ObjectGrid XML file as a URL to one of the connect methods on the `ObjectGridManager`.

Creating the client for XML

```
ObjectGridManager ogManager =
  ObjectGridManagerFactory.ObjectGridManager();
```

```
ClientClusterContext clientClusterContext =
    ogManager.connect("MyServer1.company.com:2809", null, new URL(
        "file:xml/companyGridClientSide.xml"));
```

Configure the client programmatically

You can also override client-side ObjectGrid settings programmatically. Create an ObjectGridConfiguration object that is similar in structure to the server-side ObjectGrid instance. The following code creates a client-side ObjectGrid instance that is functionally equivalent to the client override in the previous section which uses an XML file.

```
client-side override programmatically
ObjectGridConfiguration companyGridConfig = ObjectGridConfigFactory
    .createObjectGridConfiguration("CompanyGrid");
Plugin txCallbackPlugin = ObjectGridConfigFactory.createPlugin(
    PluginType.TRANSACTION_CALLBACK, "com.company.MyClientTxCallback");
companyGridConfig.addPlugin(txCallbackPlugin);

Plugin ogEventListenerPlugin = ObjectGridConfigFactory.createPlugin(
    PluginType.OBJECTGRID_EVENT_LISTENER, "");
companyGridConfig.addPlugin(ogEventListenerPlugin);

BackingMapConfiguration customerMapConfig = ObjectGridConfigFactory
    .createBackingMapConfiguration("Customer");
customerMapConfig.setNumberOfBuckets(1429);
Plugin evictorPlugin = ObjectGridConfigFactory.createPlugin(PluginType.EVICTOR,
    "com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor");
customerMapConfig.addPlugin(evictorPlugin);

companyGridConfig.addBackingMapConfiguration(customerMapConfig);

BackingMapConfiguration orderLineMapConfig = ObjectGridConfigFactory
    .createBackingMapConfiguration("OrderLine");
orderLineMapConfig.setNumberOfBuckets(701);
orderLineMapConfig.setTimeToLive(800);
orderLineMapConfig.setTtlEvictorType(TTLType.LAST_ACCESS_TIME);

companyGridConfig.addBackingMapConfiguration(orderLineMapConfig);

List ogConfigs = new ArrayList();
ogConfigs.add(companyGridConfig);

Map overrideMap = new HashMap();
overrideMap.put(CatalogServerProperties.DEFAULT_DOMAIN, ogConfigs);

ogManager.setOverrideObjectGridConfigurations(overrideMap);
ClientClusterContext client = ogManager.connect(catalogServerAddresses, null, null);
ObjectGrid companyGrid = ogManager.getObjectGrid(client, objectGridName);
```

The ogManager instance of the ObjectGridManager interface checks for overrides only in the ObjectGridConfiguration and BackingMapConfiguration objects that you include in the overrideMap Map. For instance, the previous code overrides the number of buckets on the OrderLine Map. However, the Order map remains unchanged on the client side because no configuration for that map is included.

Configure the client in the Spring Framework

Client-side ObjectGrid settings can also be overridden using the Spring Framework. The following example XML file shows how to build an ObjectGridConfiguration element, and use it to override some client side settings. This example calls the same APIs that are demonstrated in the programmatic configuration. The example is also functionally equivalent to the example in the ObjectGrid XML configuration.

```
client configuration with Spring
<?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="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="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="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" />
            </bean>
          </property>
          <constructor-arg type="java.lang.String"
            value="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" />
        </bean>
      </list>
    </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" />
  </bean>
  <property name="timeToLive" value="800" />
  <property name="ttlEvictorType">
    <value type="com.ibm.websphere.objectgrid.
      TTLType">LAST_ACCESS_TIME</value>
  </property>
  </list>
</property>
</bean>

  <bean id="client" factory-bean="manager" factory-method="connect"
    singleton="true">
    <constructor-arg type="java.lang.String">
      <value>localhost:2809</value>
    </constructor-arg>
  </bean>
  <constructor-arg
    type="com.ibm.websphere.objectgrid.security.

```

```

        config.ClientSecurityConfiguration">
        <null />
        </constructor-arg>
        <constructor-arg type="java.net.URL">
        <null />
        </constructor-arg>
        </bean>
</beans>

```

After creating the XML file, load the file and build the ObjectGrid with the following code snippet.

```

BeanFactory beanFactory = new XmlBeanFactory(new
UrlResource("file:test/companyGridSpring.xml"));

ObjectGrid companyGrid = (ObjectGrid) beanFactory.getBean("companyGrid");

```

Read about the Spring framework integration overview for more information on creating an XML descriptor file.

Disable the client near cache

The near cache is enabled by default when locking is configured as optimistic or none. Clients do not maintain a near cache when the locking setting is configured as pessimistic. To disable the near cache, you must set the `numberOfBuckets` attribute to 0 in the client override ObjectGrid descriptor file.

Enabling the client invalidation mechanism

In a distributed WebSphere eXtreme Scale environment, the client side has a near cache by default when using the optimistic locking strategy or when locking is disabled. The near cache has its own local cached data. If an eXtreme Scale client commits an update, the update goes to the client near cache and server. However, other eXtreme Scale clients do not receive the update information and might have data that is out of date.

Near cache

Applications must be aware of this stale data issue in eXtreme Scale client. You can use the built-in Java Message Service (JMS)-based `ObjectGridEventListener` `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 eXtreme Scale runtime 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, read about the “JMS event listener” on page 150.

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="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="agent">
          <bean id="ObjectTransformer" className="com.ibm.ws.objectgrid.test.scenario.AgentObjectTransformer" />
        </backingMapPluginCollection>
        <backingMapPluginCollection id="profile">
          <bean id="ObjectTransformer" className="com.ibm.ws.objectgrid.test.scenario.ProfileObjectTransformer" />
          <bean id="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="pessimisticMap" />
        <backingMapPluginCollection id="excludedMap1" />
        <backingMapPluginCollection id="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="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;peessimisticMap" 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="peessimisticMap" readOnly="false" pluginCollectionRef="peessimisticMap" 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="agent">
      <bean id="ObjectTransformer" className="com.ibm.ws.objectgrid.test.scenario.AgentObjectTransformer" />
    </backingMapPluginCollection>
    <backingMapPluginCollection id="profile">
      <bean id="ObjectTransformer" className="com.ibm.ws.objectgrid.test.scenario.ProfileObjectTransformer" />
      <bean id="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="peessimisticMap" />
    <backingMapPluginCollection id="excludedMap1" />
    <backingMapPluginCollection id="excludedMap2" />
  </backingMapPluginCollections>
</objectGridConfig>
```

Configuring request retry timeout values

With reliable maps, you can supply a retry timeout value in milliseconds to WebSphere eXtreme Scale for transaction requests.

About this task

You can configure the 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 eXtreme Scale does not attempt to try the transaction again after any type of transaction.

During run time, the transaction timeout value is used with the retry timeout value, ensuring that the 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 CORBA SystemException and eXtreme Scale TargetNotAvailable exceptions.
- For autocommit transactions, the transactions are tried again for CORBA SystemException and eXtreme Scale availability exceptions. These exceptions include the ReplicationVotedToRollbackTransactionException, TargetNotAvailable, and AvailabilityException exceptions. For more information about autocommit transactions, see the topic on using sessions to access data in the grid in the *Programming Guide*.

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
- 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 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” on page 245. 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 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 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 milliseconds or 30 seconds, regardless of the value that is set in the client properties file. For more information on the session interface, see [Using Sessions to access data in the data grid](#).

Configuring entities

A data grid 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 are bound to `BackingMaps`, indexes and other plug-ins.

Before you begin

An entity schema is a set of entities and the relationships between the entities. Read about [Defining an entity schema](#) for details about schema definition and entity configuration.

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`. With look-up or reference data, the data should 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.

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” on page 263 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” on page 259

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” on page 259

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” on page 259

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” on page 259

attributes element

Child elements

- “id element” on page 258
- “basic element” on page 258
- “version element” on page 259
- “many-to-one element” on page 260
- “one-to-many element” on page 260
- “one-to-one element” on page 259
- “many-to-many element”
- “transient element” on page 259

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” on page 259
- “attributes element” on page 261

entity-mappings element

Child elements

- **description:** Specifies a description.
- “Entity element” on page 261

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” on page 263

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.

See the “Entity metadata descriptor XML file” on page 258 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>
```

Configuring cache integration

WebSphere eXtreme Scale can integrate with other caching-related products. You can use a Java Persistence API (JPA) between WebSphere eXtreme Scale and the database to integrate changes as a loader. 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 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.
- 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.

About this task

For more information about how the Java Persistence API (JPA) Loader works, see the information in the *Product Overview*.

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 requires this parameter, but the parameter is optional for **JPAEntityLoader**. In the case of JPAEntityLoader, 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 meta-data 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 the information about loader plug-ins in the *Programming Guide*.

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="TransactionCallback"
        className="com.ibm.websphere.objectgrid.jpa.JPATxCallback">
        <property
          name="persistenceUnitName"
          type="java.lang.String"
          value="employeeEMPU" />
        </property>
      </bean>
      <backingMap name="Employee" pluginCollectionRef="Employee" />
    </objectGrid>
  </objectGrids>

  <backingMapPluginCollections>
    <backingMapPluginCollection id="Employee">
      <bean id="Loader"
        className="com.ibm.websphere.objectgrid.jpa.JPAEntityLoader">
      <property
        name="entityClassName"
        type="java.lang.String"
        value="com.ibm.ws.objectgrid.jpa.test.entity.Employee"/>
      </property>
      </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="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="jpaLoader"
entityClassName="com.ibm.ws.objectgrid.jpa.test.entity.Employee"/>
  <objectgrid:JPATxCallback id="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="TransactionCallback"
        className="{spring}jpaTxCallback"/>
      <backingMap name="Employee" pluginCollectionRef="Employee"
        writeBehind="T4"/>
    </objectGrid>
  </objectGrids>

  <backingMapPluginCollections>
    <backingMapPluginCollection id="Employee">
      <bean id="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.

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 the information in the *Programming Guide*.

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 the information about the BackingMap interface in the API documentation.

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 the information about starting the JPA time-based data updater in the *Programming Guide* for more information.

JPA cache configuration properties

WebSphere eXtreme Scale includes level 2 cache plug-ins for both OpenJPA and Hibernate Java Persistence API (JPA) providers.

You can configure the JPA cache plug-in with the following properties, all of which are optional.

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.

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.

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`.

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 the information about evictors in the *Product Overview*. For more information about the server properties file, see the *Administration Guide*.

MaxNumberOfReplicas

Valid values: greater than or equal to 1 Specifies the maximum number of replicas to be use for the cache. This value only applies to the `EMBEDDED` type. This number should be equal to or greater than the number of Java virtual machines in a system. The default value is 47.

The `NumberOfPartitions`, `ReplicaMode`, `ReplicaReadEnabled`, and `MaxNumberOfReplicas` properties are `ObjectGrid` deployment factors. The `NumberOfPartitions`, `ReplicaMode`, and `ReplicaReadEnabled` apply to the `EMBEDDED_PARTITION` type. Both `ReplicaMode` and `MaxNumberOfReplicas` apply to the `EMBEDDED` type.

EMBEDDED and EMBEDDED_PARTITION considerations

The embedded `ObjectGrid` types use the configuration properties previously described to configure and deploy a set of `ObjectGrid` container servers and a catalog service when necessary. The life cycle of the containers is bound to the JPA application and is collocated within the application class path. When an application is started, the plug-in automatically detects or starts a catalog service, starts a container, and connects 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"`

Note: 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 seems to become unresponsive.

ObjectGridType value defaults

The `ObjectGridType` value specifies the topology in which the `ObjectGrid` cache is deployed. The default and best performing type is `EMBEDDED`. The following sections describe the default properties for each of the `ObjectGridType` values.

EMBEDDED ObjectGrid JPA cache topology defaults

When you are using the `EMBEDDED` `ObjectGrid` type, 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 (should be less than or equal to the number of Java virtual machines in a distributed system)

You should specify a unique `ObjectGridName` value to avoid naming conflicts. The `MaxNumberOfReplicas` value should be equal to or greater than the total number of Java virtual machines in the system.

REMOTE ObjectGrid cache topology

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.

Catalog service considerations and configuration

When you are running in an `EMBEDDED` or `EMBEDDED_PARTITION` topology, the JPA cache plug-in automatically starts a single catalog service within one of the application processes if needed. In a production environment, you should create a catalog service domain. For more information about defining a catalog service, see the information about the high-availability catalog service in the *Product Overview*

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. For more information about defining a catalog service domain, see the information about starting the catalog service in the *Administration Guide*.

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 grid 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>
```

JPA cache plug-in

WebSphere eXtreme Scale includes level 2 (L2) cache plug-ins for both OpenJPA and Hibernate Java Persistence API (JPA) providers.

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.

With the OpenJPA and Hibernate ObjectGrid cache plug-ins, you can create three topology types: embedded, embedded-partitioned, and remote.

Embedded topology

An embedded topology creates an eXtreme Scale 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.

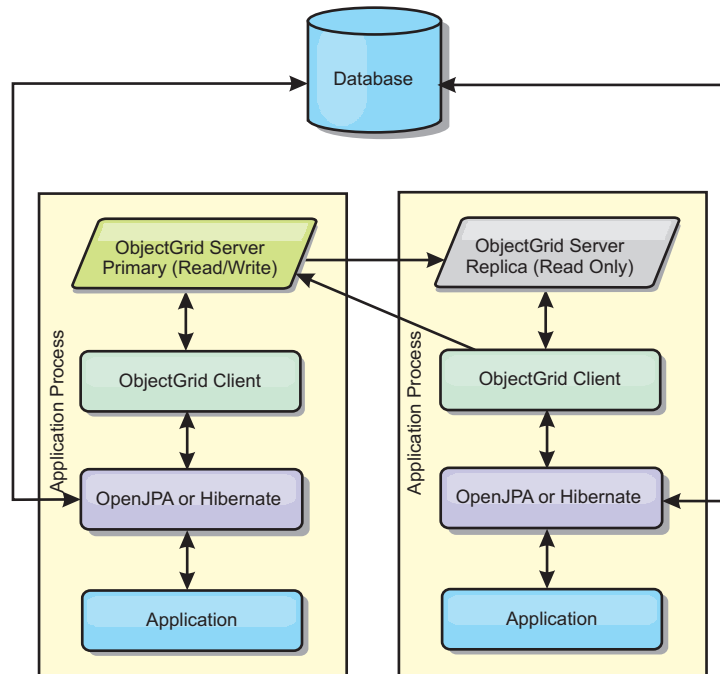


Figure 20. JPA embedded topology

Advantages:

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

Limitations:

- Amount of data is limited to the size of the process.
- All cache updates are sent to one process.

Embedded, partitioned topology

When the cached data is too large to fit in a single process, the embedded, partitioned topology uses ObjectGrid partitions to divide the data over multiple processes. Performance is not as high as the embedded topology because most cache reads are remote. However, you can still use this option when database latency is high.

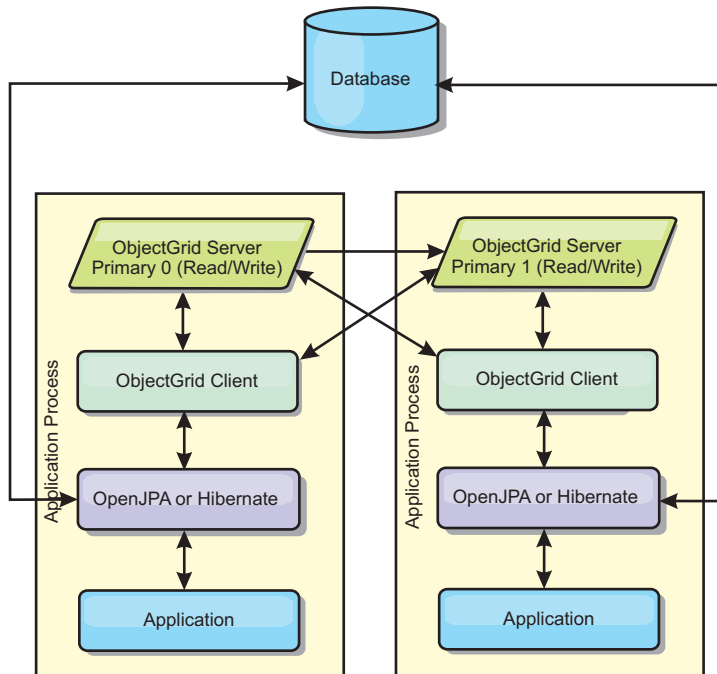


Figure 21. JPA embedded, partitioned topology

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, ten Java virtual machines are required. The number of partitions must therefore be set to 10 or more. Ideally, the number of partitions should be set to a prime number where each shard stores a reasonable amount of memory. Usually, the `numberOfPartitions` setting is equal to the number of Java virtual machines. With this setting, each JVM stores one partition. If you enable replication, you must increase the number of Java virtual machines in the system. Otherwise, each JVM also stores one replica partition, which consumes as much memory as a primary partition.

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

For example, in a system with 4 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

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. Read about monitoring your deployment environment for more information on managing an eXtreme Scale data grid deployment.

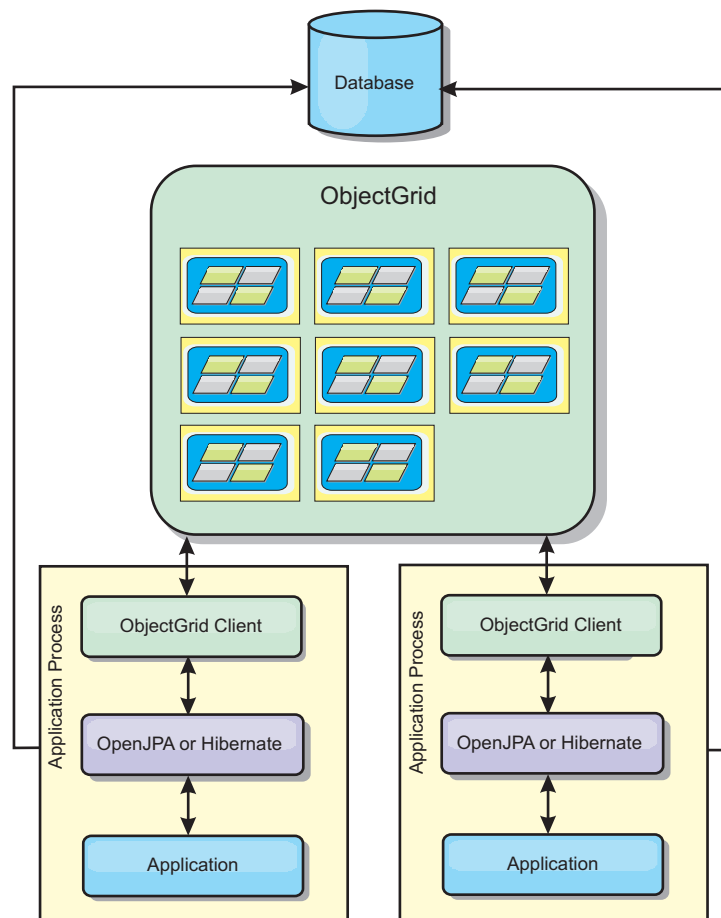


Figure 22. JPA remote topology

Advantages:

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

Limitation:

- All cache reads and updates are remote.

Hibernate cache plug-in configuration

An eXtreme Scale cache can be enabled for Hibernate by setting properties in the configuration file.

7.1+ For integration with WebSphere Application Server, the hibernate cache plug-in is packaged in `oghibernate-cache.jar` and installed in `was_root/optionalLibraries/ObjectGrid`. 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, you have to include the `oghibernate-cache.jar` file, too. If you define a shared library to include hibernate library, you have to put the `oghibernate-cache.jar` file into the shared library directory.

7.1+ eXtreme Scale, Version 7.1, 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 `cglib.jar` and include it in the classpath. For example, if your application includes all hibernate library JAR files, but excludes the `cglib.jar` available with hibernate, you must include the `cglib.jar` comes from hibernate in your application.

Settings

The syntax for setting the property in the `persistence.xml` file follows:

persistence.xml

```
<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>,.." />
```

The syntax for setting the property in the `hibernate.cfg.xml` file follows:

hibernate.cfg.xml

```
<property name="cache.provider_class">com.ibm.websphere.objectgrid.
hibernate.cache.ObjectGridHibernateCacheProvider</property>
<property name="cache.use_query_cache">true</property>
<property name="objectgrid.configuration"><property><value>,...</property>
<property name="objectgrid.hibernate.regionNames"><regionName>,...</property>
```

The `provider_class` property is the `com.ibm.websphere.objectgrid.hibernate.cache.ObjectGridHibernateCacheProvider` property. To enable query cache, set the value to `true` on the `use_query_cache` property. Use the `objectgrid.configuration` property to specify eXtreme Scale cache configuration properties.

You must specify a unique `ObjectGridName` property value to avoid potential naming conflicts. The other eXtreme Scale cache configuration properties are optional.

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.

The following are examples of the persistence.xml and hibernate.cfg.xml files:

persistence.xml

```
<persistence-unit name="testPU2">
  <provider>org.hibernate.ejb.HibernatePersistence</provider>
  <class>com.ibm.websphere.objectgrid.jpa.test.Department</class>
  <properties>
    <property name="hibernate.show_sql" value="true" />
    <property name="hibernate.connection.url" value="jdbc:derby:DB_testPU2;create=true" />
    <property name="hibernate.connection.driver_class" value="org.apache.derby.jdbc.EmbeddedDriver" />

    <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="ObjectGridName=myOGName,ObjectGridType=
      EMBEDDED,MaxNumberOfReplicas=4" writeBehind=true, writeBehindInterval=5000,
      writeBehindPoolSize=10, writeBehindMaxBatchSize=1000" />
    <property name="objectgrid.hibernate.regionNames" value="queryRegion1, queryRegion2" />
  </properties>
</persistence-unit>
```

7.1+ Version 7.1 introduces additional write behind function specific configuration options for the Hibernate cache plug-in, in addition to standard JPA cache plug-in configuration options.

writeBehind

Valid values: TRUE or FALSE

Default value: FALSE

When writeBehind is enabled, updates are temporarily stored in a JVM scope data storage until either the writeBehindInterval or writeBehindMaxBatchSize condition is met.

Attention: Unless writeBehind is enabled, the other write behind configuration settings are disregarded.

writeBehindInterval

Valid values: greater than or equal to 1

Default value: 5000 (5 seconds)

Specifies the time interval in milliseconds to flush updates to the cache.

writeBehindPoolSize

Valid values: greater than or equal to 1

Default value: 5

Specifies the maximum size of the thread pool used in flushing updates to the cache.

writeBehindMaxBatchSize

Valid values: greater than or equal to 1

Default value: 1000

Specifies the maximum batch size per region cache in flushing updates to the cache.

The preceding code example displays the following write behind function configuration:

```
writeBehind=true, writeBehindInterval=5000, writeBehindPoolSize=10, writeBehindMaxBatchSize=1000
```

where

- writeBehind=TRUE enables the write behind function
- writeBehindInterval=5000 means that updates will flush to the cache approximately every 5 seconds
- writeBehindPoolSize=10 indicates that the maximum number of threads used to perform the work is 10 threads, when flushing updates to the cache
- writeBehindMaxBatchSize=1000 means that if the updates stored in the write behind storage of a region cache exceeds 1000 entries, the updates will be flushed to the cache, even the specified writeBehindInterval condition is not met. In other words, updates will flush to cache either approximately every 5 seconds 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 will flush its updates in write behind storage to cache. A region cache usually is corresponding to an entity or a query.

Important:

Take care when using the write behind function configuration. It introduces longer latency of data synchronization across all JVMs and a higher chance of lost updates. In a system using write behind configuration with four or more JVMs, the update performed on one JVM will have 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 will lose its update.

hibernate.cfg.xml

```
<hibernate-configuration>
  <session-factory>
    <!-- Database connection settings -->
    <property name="connection.driver_class">org.apache.derby.jdbc.EmbeddedDriver</property>
    <property name="connection.url">jdbc:derby:DB_testPU2;create=true</property>
    <!-- ObjectGrid cache setting-->
    <property name="cache.provider_class">com.ibm.websphere.objectgrid.hibernate.cache.
      ObjectGridHibernateCacheProvider</property>
    <property name="cache.use_query_cache">true</property>
    <property name="objectgrid.configuration">ObjectGridName=myOGName,
      ObjectGridType=EMBEDDED,MaxNumberOfReplicas=4 </property>
    <property name="objectgrid.hibernate.regionNames">queryRegion1, queryRegion2</property>

    <mapping resource="com/ibm/websphere/objectgrid/jpa/test/Employee.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

Preloading 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 1

Using EntityManagerFactory

```
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 2

Using SessionFactory

```
org.hibernate.cfg.Configuration cfg = new Configuration();
// use addResource, addClass, and setProperty method of Configuration to prepare
// configuration required to create SessionFactory
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` in order 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.

Customizing Hibernate cache configuration with XML

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 will try 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), `hibernate-objectGridDeployment.xml` (deployment policy), and `hibernate-objectGrid-client-override.xml` (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.

1. **ObjectGrid configuration:** Use the `META-INF/hibernate-objectGrid.xml` file. 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 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.xml file. BackingMaps that are not listed in this hibernate-objectGrid.xml file use the default configuration.

2. **ObjectGridDeployment configuration:** Use the META-INF/hibernate-objectGridDeployment.xml file. 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.
3. **Client override ObjectGrid configuration:** Use the META-INF/hibernate-objectGrid-client-override.xml file. 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. If the application requires a near cache, it can provide this file and specify numberOfBuckets="xxx". The default client override disables the near cache by setting numberOfBuckets="0". The near cache can be active when resetting numberOfBuckets attribute to a value greater than 0 with the hibernate-objectGrid-client-override.xml file. The way that the hibernate-objectGrid-client-override.xml file works is similar to hibernate-objectGrid.xml: It overrides or extends the default client ObjectGrid override configuration.

Hibernate ObjectGrid XML file examples

Hibernate ObjectGrid XML files should be created based on the configuration of a persistence unit.

An example persistence.xml file that represents the configuration of a persistence unit follows:

persistence.xml

```
<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.AnnuityPersistibleObject</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>
```

The following is the hibernate-objectGrid.xml file that matches the persistence.xml file:

hibernate-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="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="com.ibm.wssvt.acme.annuity.common.bean.jpa.Annuity">
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="defaultCacheMap">
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payor">
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="com.ibm.wssvt.acme.annuity.common.bean.jpa.Contact">
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="com.ibm.wssvt.acme.annuity.common.bean.jpa.Person">
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="com.ibm.wssvt.acme.annuity.common.bean.jpa.Rider">
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payout">
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="org.hibernate.cache.UpdateTimestampsCache">
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="org.hibernate.cache.StandardQueryCache">
      <bean id="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.

The hibernate-objectGridDeployment.xml file that matches the persistence.xml follows:

hibernate-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="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>

```

Important: The org.hibernate.cache.UpdateTimestampsCache, org.hibernate.cache.StandardQueryCache and defaultCacheMap maps are required.

External system for a cache with 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. The Hibernate ObjectGrid and ObjectGridDeployment configuration XML files that are described in the Hibernate ObjectGrid XML files example section can also be used to setup an external eXtreme Scale system.

An external ObjectGrid system has both catalog service and ObjectGrid server processes. You must start a catalog service before starting container servers. See the details on starting stand-alone eXtreme Scale servers and container processes in the *Administration Guide* for more information.

Troubleshooting

1. CacheException: Failed to get ObjectGrid server

With either an EMBEDDED or EMBEDDED_PARTITION ObjectGridType, the cache tries to obtain a server instance from the eXtreme Scale runtime. In a Java Platform, Standard Edition environment, an eXtreme Scale server with embedded catalog service will be started. The embedded catalog service tries to listen to port 2809; if that port is being used by another process, this error occurs. If external catalog service endpoints are specified, for instance, with the objectGridServer.properties file, this error occurs if the host name or port is specified incorrectly.

2. CacheException: Failed to get REMOTE ObjectGrid for configured REMOTE ObjectGrid. objectGridName = [ObjectGridName], PU name = [persistenceUnitName]

This error occurs when the cache fails to obtain an ObjectGrid from the provided catalog service end points. Typically, the error is because of an incorrect host name or port.

3. CacheException: Cannot have two PUs [persistenceUnitName_1, persistenceUnitName_2] configured with same ObjectGridName [ObjectGridName] of EMBEDDED ObjectGridType

This exception occurs if you have a configuration with many persistence units and the caches of these units are configured with the same ObjectGrid name and EMBEDDED ObjectGridType. These persistence unit configurations can be in the same or different persistence.xml files. You must verify that the ObjectGrid name is unique for each persistence unit when ObjectGridType is EMBEDDED.

4. 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 BackingMaps to support the cache for the persistence unit, this exception results. For example, five entity classes are listed in the configuration for the persistence unit, but the obtained ObjectGrid only has two BackingMaps. Even though the obtained ObjectGrid might have ten BackingMaps, if any one of the five required entity BackingMaps are not found in the ten BackingMaps, this exception still occurs.

OpenJPA cache plug-in configuration

WebSphere eXtreme Scale provides both DataCache and QueryCache implementations for OpenJPA. The OpenJPA ObjectGrid cache or ObjectGrid cache in short, is a common term for both the DataCache and QueryCache implementations.

Settings

The eXtreme Scale cache is enabled or disabled for OpenJPA by setting the `openjpa.DataCache` and `openjpa.QueryCache` configuration properties in the `persistence.xml` file. The syntax for setting the property follows:

```
<property name="openjpa.DataCache"
  value="<object_grid_datacache_class(<property>=<value>,...)" />
<property name="openjpa.QueryCache"
  value="<object_grid_querycache_class(<property>=<value>,...)" />
```

Both DataCache and QueryCache can take eXtreme Scale cache properties to configure the cache used by the persistence unit.

In addition to the eXtreme Scale cache setting, the `openjpa.RemoteCommitProvider` property has to be set to `sjvm`:

```
<property name="openjpa.RemoteCommitProvider" value="sjvm" />
```

The timeout value specified with `@DataCache` annotation for each entity class is carried down to the BackingMap to which each entity is cached. However, the name value specified with `@DataCache` annotation is ignored by the eXtreme Scale cache. The fully qualified entity class name is the cache map name.

The `pin` and `unpin` methods of OpenJPA StoreCache and QueryCache are not supported and perform no function.

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" />
```

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.

Customizing OpenJPA cache configuration with XML

For most scenarios, setting eXtreme Scale cache properties should be sufficient. To further customize the ObjectGrid used by the cache, you can provide OpenJPA ObjectGrid configuration XML files in your META-INF directory similarly to the `persistence.xml` file. During cache initialization, the ObjectGrid cache tries to locate these XML files and process the files if they are found.

There are three types of OpenJPA ObjectGrid configuration XML files: the `openjpa-objectGrid.xml` (ObjectGrid configuration), `openjpa-objectGridDeployment.xml` (deployment policy), and `openjpa-objectGrid-client-override.xml` (client ObjectGrid override configuration) files. Depending on the configured ObjectGrid type, you can provide any one of these three XML files to customize the ObjectGrid.

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.

- ObjectGrid configuration:** Use the `META-INF/openjpa-objectGrid.xml` file. 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.
- ObjectGridDeployment configuration:** Use the `META-INF/openjpa-objectGridDeployment.xml` file. 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.
- Client override ObjectGrid configuration:** Use the `META-INF/openjpa-objectGrid-client-override.xml` file. 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. If an application requires a

near cache, it can provide this file and specify `numberOfBuckets="xxx"`. The default client override disables the near cache by setting `numberOfBuckets="0"`. The near cache can be active when resetting `numberOfBuckets` to a value greater than 0 with the `openjpa-objectGrid-client-override.xml` file. 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.

OpenJPA ObjectGrid XML file examples

OpenJPA ObjectGrid XML files should be created based on the configuration of the persistence unit.

A `persistence.xml` file that is an example that represents the configuration of a persistence unit follows:

`persistence.xml`

```
<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.AnnuityPersistibleObject</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>
```

The `openjpa-objectGrid.xml` file that matches the `persistence.xml` file follows:

`openjpa-objectGrid.xml`

```
<?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.AnnuityPersistibleObject"
        readOnly="false" copyKey="false"
        lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
        pluginCollectionRef="com.ibm.wssvt.acme.annuity.common.bean.jpa.AnnuityPersistibleObject" />
    </objectGrid>
  </objectGrids>
</objectGridConfig>
```

```

<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="com.ibm.wssvt.acme.annuity.common.bean.jpa.Annuity">
<bean id="ObjectTransformer"
className="com.ibm.ws.objectgrid.openjpa.ObjectGridPCDataObjectTransformer" />
<bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
</bean>
</backingMapPluginCollection>
<backingMapPluginCollection id="com.ibm.wssvt.acme.annuity.common.bean.jpa.Address">
<bean id="ObjectTransformer"
className="com.ibm.ws.objectgrid.openjpa.ObjectGridPCDataObjectTransformer" />
<bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
</bean>
</backingMapPluginCollection>
<backingMapPluginCollection id="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payor">
<bean id="ObjectTransformer"
className="com.ibm.ws.objectgrid.openjpa.ObjectGridPCDataObjectTransformer" />
<bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
</bean>
</backingMapPluginCollection>
<backingMapPluginCollection id="com.ibm.wssvt.acme.annuity.common.bean.jpa.Person">
<bean id="ObjectTransformer"
className="com.ibm.ws.objectgrid.openjpa.ObjectGridPCDataObjectTransformer" />
<bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
</bean>
</backingMapPluginCollection>
<backingMapPluginCollection id="com.ibm.wssvt.acme.annuity.common.bean.jpa.Contact">
<bean id="ObjectTransformer"
className="com.ibm.ws.objectgrid.openjpa.ObjectGridPCDataObjectTransformer" />
<bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
</bean>
</backingMapPluginCollection>
<backingMapPluginCollection
id="com.ibm.wssvt.acme.annuity.common.bean.jpa.AnnuityPersistibleObject">
<bean id="ObjectTransformer"
className="com.ibm.ws.objectgrid.openjpa.ObjectGridPCDataObjectTransformer" />
<bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
</bean>
</backingMapPluginCollection>
<backingMapPluginCollection id="com.ibm.wssvt.acme.annuity.common.bean.jpa.Rider">
<bean id="ObjectTransformer"
className="com.ibm.ws.objectgrid.openjpa.ObjectGridPCDataObjectTransformer" />
<bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
</bean>
</backingMapPluginCollection>
<backingMapPluginCollection id="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payout">
<bean id="ObjectTransformer"
className="com.ibm.ws.objectgrid.openjpa.ObjectGridPCDataObjectTransformer" />
<bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
</bean>
</backingMapPluginCollection>
<backingMapPluginCollection id="ObjectGridQueryCache">
<bean id="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="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.

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.AnnuityPersistibleObject" />
      <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.

External system for a cache with REMOTE ObjectGrid type

You must set up an external system if you want to configure an cache with REMOTE ObjectGrid type. You need both ObjectGrid and ObjectGridDeployment configuration XML files that are based on a persistence.xml file to set up an external system. The OpenJPA ObjectGrid and ObjectGridDeployment configuration XML files described in the OpenJPA ObjectGrid XML file examples section can also be used to set up an external eXtreme Scale system.

An external eXtreme Scale system has both catalog service and container server processes. You must start the catalog server before starting container servers.

Troubleshooting

1. CacheException: Failed to get ObjectGrid server

With either an EMBEDDED or EMBEDDED_PARTITION ObjectGridType, 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. 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.

2. **CacheException: Failed to get REMOTE ObjectGrid for configured REMOTE ObjectGrid. objectGridName = [ObjectGridName], PU name = [persistenceUnitName]**

This error occurs when the cache fails to obtain an ObjectGrid from the provided catalog service endpoints. Typically, the error is because of an incorrect host name or port.

3. **CacheException: Cannot have two PUs [persistenceUnitName_1, persistenceUnitName_2] configured with same ObjectGridName [ObjectGridName] of EMBEDDED ObjectGridType**

This exception results if you have a many persistence unit configuration and the eXtreme Scale caches of these units are configured with the same ObjectGrid name and EMBEDDED ObjectGridType. These persistence unit configurations could be in the same or different persistence.xml files. You must verify that the ObjectGrid name is unique for each persistence unit when ObjectGridType is EMBEDDED.

4. **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 BackingMaps 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 ten BackingMaps, if any one of the five required entity BackingMaps is not found in the ten BackingMaps, this exception still occurs.

Note: The OpenJPA eXtreme Scale cache has changed data format to improve performance. Any systems that are hosting OpenJPA applications that are configured with eXtreme Scale as an L2 cache must be stopped before migrating to WebSphere eXtreme Scale Version 7.0.

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

Attention: **7.1.0.3+** With Version 7.1.0.3 and later, you can persist sessions that use URL rewriting or cookies as a session tracking mechanism. Before Version 7.1.0.3, you can save only sessions that use cookies as the session tracking mechanism to the data grid.

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. See “Installing WebSphere eXtreme Scale or WebSphere eXtreme Scale Client with WebSphere Application Server” on page 26 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” on page 199. For more information about configuring global security, see Global security settings.
- **7.1.0.3+** 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.

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 WebSphere eXtreme Scale 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 depending on if 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 run 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 eXtreme Scale 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:

- **7.1+ Auto-splice with WebSphere Application Server**

You can configure your application to use the WebSphere eXtreme Scale HTTP session manager when you install your application. You can also edit the application or server configuration to use the WebSphere eXtreme Scale HTTP session manager. See “Automatically splicing applications for HTTP session management in WebSphere Application Server” on page 292 for more information.

Attention: **7.1.0.3+** If you want to enable persistence for sessions that use URL rewriting, you must edit the `splicer.properties` file after your auto-splice the application. In the `splicer.properties` file, set the **useURLEncoding** property to true. Synchronize the nodes to propagate the changes to other nodes in the configuration.

- **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.

Add a custom property to either a cell or a server to set the `splicer.properties` file for all of the Web applications at that scope. Use the following steps to configure the custom property:

- a. In the WebSphere Application Server administrative console, navigate to the correct path for where you want to set the custom property to indicate the location of the `splicer.properties` file.
 - To set the custom property for all applications or a specific application, click **System administration > Cell > Custom properties**.
 - To set the custom property to apply to all the applications on a specific application server, click **Application server > <server_name> > Administration > Custom properties**. The property name is `com.ibm.websphere.xs.sessionFilterProps`, and its 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`.
- b. Add the `com.ibm.websphere.xs.sessionFilterProps` custom property. This custom property value gives the location of the `splicer.properties` file to edit. 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.
- c. Edit the `splicer.properties` file that is in the path for the custom property on the deployment manager profile.
- d. Synchronize your nodes so the updated `splicer.properties` file is propagated to your nodes. Click **System Administration > Nodes**. Choose the nodes on which the application is installed, and click **Synchronize**.

Important: Ensure that the updated `splicer.properties` file is on the same path on all nodes containing an application server hosting the application or applications that are being spliced for session replication."

The cell, server, and application scope are available scopes and are only available when running in a deployment manager. If you require a different scope, manually splice your Web applications.

Remember: Also, note that the auto-splice option works only if all of the nodes running the application contain the `splicer.properties` file at the same path. For mixed environments containing Windows and UNIX nodes, this option is not possible, so you must manually splice the application.

- **Splice the application with the `addObjectGridFilter` script**

Use a command-line script provided along with eXtreme Scale to splice an application with filter declarations and configuration in the form of servlet context initialization parameters. For a WebSphere Application Server deployment, this script is located in `<was_home>/optionalLibraries/ObjectGrid/session/bin/addObjectGridFilter.bat/sh`. For a stand-alone deployment, the script is at `WXS_HOME/ObjectGrid/session/bin/addObjectGridFilter.sh/bat`. The **`addObjectGridFilter`** script takes two parameters:

- Application - absolute path to the enterprise archive file to be spliced
- Absolute path to the splicer properties file that contains various configuration properties.

The usage format of this script is as follows:

Windows

```
addObjectGridFilter.bat [location of ear file] [location of splicer properties file]
```

UNIX

```
addObjectGridFilter.sh [location of ear file] [location of splicer properties file]
```

UNIX

Example using eXtreme Scale installed on WebSphere Application Server on UNIX:

- `cd was_home/optionalLibraries/ObjectGrid/session/bin`
- `addObjectGridFilter.sh /tmp/mySessionTest.ear was_root/optionalLibraries/ObjectGrid/session/samples/splicer.properties`

UNIX

Example using eXtreme Scale installed in a stand-alone directory on UNIX:

- `cd was_root/session/bin`
- `addObjectGridFilter.sh /tmp/mySessionTest.ear was_root/session/samples/splicer.properties`

The servlet filter that is spliced maintains defaults for configuration values. You can override these default values with configuration options that you specify in the properties file in the second argument. For a list of the parameters that you can use, see “Servlet context initialization parameters” on page 308.

You can modify and use the sample `splicer.properties` file that is provided with eXtreme Scale installation. You can also use the **`addObjectGridServlets`** script, which inserts the session manager by extending each servlet. However, the recommended script is the **`addObjectGridFilter`** script.

- **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, invoke the Ant process by running the following command:

- `ant.sh, ws_ant.sh`
- `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” on page 308.

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 a majority 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.

7.1.0.3+ 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, 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.

Automatically splicing applications for HTTP session management in WebSphere Application Server:

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” on page 288 for information about creating a session data grid.
- If the catalog service that you want to use to manage your sessions is outside of the cell in which you are installing your session application, you must create a catalog service domain. See “Creating catalog service domains in WebSphere Application Server” on page 206 for more information.
- If you are configuring a catalog service domain, you might need to 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. See “Configuring client security on a catalog service domain” on page 296 for more information about configuring these settings.
- Global security enabled in the WebSphere Application Server administrative console, if you want to support one of the following 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” on page 199. For more information about configuring global security, see Global security settings.

- **7.1.0.3+** 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.

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#.*\s1=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 microdglapp.war,WEB-INF/web.xml
default_host] [MicroDG2App microdgp2app.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#.*\s1=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 microdglapp.war,WEB-INF/web.xml
default_host] [MicroDG2App microdgp2app.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#.*\s1=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 microdglapp.war,WEB-INF/web.xml
default_host] [MicroDG2App microdgp2app.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:**
 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]]]')
```

For the eXtreme Scale embedded scenario with default configuration:


```
AdminApp.edit('DefaultApplication','[-SessionManagement[[[true  
XSEmbeddedSessionManagement :!: :!:default]]]')
```

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]]]')
```

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.

- If you want to edit other aspects of the HTTP session configuration, you can edit the `splicer.properties` file. You can get the path location of the `splicer.properties` file by locating the `com.ibm.websphere.xs.sessionFilterProps` custom property in one of the following locations:
 - In a WebSphere Application Server Network Deployment environment: A custom property on the cell
 - In a stand-alone WebSphere Application Server environment: A custom property on the application server

You can open the indicated file, make changes, and synchronize the nodes so the updated properties file gets propagated to the other nodes in the configuration. All application server nodes require the `splicer.properties` file to be in the specified path to properly persist sessions.

Attention: **7.1.0.3+** If you want to enable persistence for sessions that use URL rewriting, set the **useURLEncoding** property to true in the `splicer.properties` file.

For more information about the properties in the `splicer.properties` file, see “`splicer.properties` file” on page 297.

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.

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” on page 206 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” on page 219 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 “Automatically splicing applications for HTTP session management in WebSphere Application Server” on page 292 for more information.

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 2. 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 = 2

# An integer value that defines the number of session references
# kept in memory. The default is 2000. 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), 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 = 2000

# 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 WebSphere 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=

# Allows for the override of the servlet spec designated
# cookie / encoded URL name of "JSESSIONID"
# If the cookieName value is changed from JSESSIONID, the
# application servers must be configured to use this value
# as well.

# cookieName = JSESSIONID

# 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=

# 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

# Set to true if you want to enable urlRewriting. Default is
# false, which means cookies will be used to store data. The
# value of this should reflect what is set in the web container
# settings for session management.

# useURLEncoding = false

```

Using WebSphere eXtreme Scale for SIP session management:

You can use WebSphere eXtreme Scale as a Session Initiation Protocol (SIP) replication mechanism as a reliable alternative to the data replication service (DRS) for SIP session replication.

SIP session management configuration

To use WebSphere eXtreme Scale as the SIP replication mechanism, set the `com.ibm.sip.ha.replicator.type` custom property. In the administrative console, select **Application servers > my_application_server > SIP container > Custom properties** for each server to add the custom property. Type `com.ibm.sip.ha.replicator.type` for the Name and `OBJECTGRID` for the Value.

Use the following properties to customize the behavior of the ObjectGrid that is used to store SIP sessions. In the administrative console, click **Application servers > my_application_server > SIP container > Custom properties** for each server to add the custom property. Type the **Name** and **Value**. Each server must have the same properties set to function properly.

Table 21. Custom properties for SIP session management with ObjectGrid

Property	Value	Default
<code>com.ibm.sip.ha.replicator.type</code>	OBJECTGRID: use ObjectGrid as SIP session store	
<code>min.synchronous.replicas</code>	Minimum number of synchronous replicas	0
<code>max.synchronous.replicas</code>	Maximum number of synchronous replicas	0
<code>max.asynchronous.replicas</code>	Maximum number of asynchronous replicas	1
<code>auto.replace.lost.shards</code>	See "Configuring distributed deployments" on page 174 for more information.	true
<code>development.mode</code>	<ul style="list-style-type: none"> • true - allow replicas to be active on same node as primaries • false - replicas must be on different node than primaries 	false

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 that 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, which means that the container server starts in the web container tier, the `objectGrid.xml` and `objectGridDeployment.xml` files are provided by default. You can update these files to customize the behavior of the HTTP session manager.

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">
```

```

<bean id="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="objectgridSessionMetadata">
    <bean id="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 grids, those applications should have different ObjectGrid name attribute values. The name of the ObjectGrid is the only thing that can be changed in this file.

Values that 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 grid in this map.
- **objectgridSessionTTL*:** This value cannot be changed and is for future use.
- **objectgridSessionAttribute.*** The objectgridSessionAttribute.* line defines a dynamic map that 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 that is used to splice the web application. 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.
- The **MapEventListener** line is internal and cannot be modified

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:

- **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 grids, those applications should have different ObjectGrid name attribute values.

- **mapSet element attributes:** You can change all mapSet properties can be changed 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`. 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 standalone processes, you must use the `objectGridStandAlone.xml` and `objectGridDeploymentStandAlone.xml` files to start the processes. You can update these files to modify the configuration.

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">
<bean id="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="objectgridSessionMetadata">
```

```

        <bean id="MapEventListener" className="com.ibm.ws.xs.sessionmanager.MetadataMapListener"/>
    </backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>

```

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 `objectGridStandAlone.xml` file.

If you have multiple applications, and you want the session data to be stored in different grids, those applications should have different ObjectGrid name attribute values. The name of the ObjectGrid is the only thing that can be changed in this file.

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 grid in this map.
- **objectgridSessionTTL*:** This value cannot be changed and is for future use.
- **objectgridSessionAttribute.*** The `objectgridSessionAttribute.` line defines a dynamic map that 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 that is used to splice the web application. 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.
- The **MetadataMapListener** line is internal and cannot be modified

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="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:

- **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 grids, those applications should 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.

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:

- **Fix 1+** WebSphere eXtreme Scale Version 7.1 must have Fix 1 or later applied.

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” on page 26 for more information.

- WebSphere Portal Version 7.0.0.0.

- 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.

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” on page 288 for more information about splicing the application.

2. If you are using a remote scenario where container servers run in different processes than the servlets, set the `timeout.resume.session` custom property in the WebSphere Application Server administrative console. By default, WebSphere Portal verifies the HTTP session ID at various times during a request. When you are running a WebSphere eXtreme Scale remote scenario and the number of HTTP sessions is higher than the configured `sessionTableSize` parameter, the session ID can change. For more information about the `sessionTableSize` attribute, see “Servlet context initialization parameters” on page 308. You must configure the `timeout.resume.session` within the `WP_ConfigService` resource environment provider to prevent users from being logged out if their session is invalidated from the WebSphere session cache.
 - a. In the WebSphere Application Server administrative console, click **Resources > Resource Environment > Resource Environment Providers > WP_ConfigService > Custom Properties > New**.
 - b. Create a custom property with a name of `timeout.resume.session` and a value of `true`.

3. 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:

```
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 processes” on page 354. If you are using an embedded scenario, see “Configuring container servers in WebSphere Application Server” on page 221 for more information about configuring and starting container servers.

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.

7.1.0.3+ 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.

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 and 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.

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**

Use a command-line script provided along with eXtreme Scale to splice an application with filter declarations and configuration in the form of servlet context initialization parameters. The `wxs_home/session/bin/addObjectGridFilter.sh|bat` script takes two parameters: the absolute path to the enterprise archive (EAR) file that you want to splice, and the absolute path to the splicer properties file that contains various configuration properties. The usage format of this script is as follows:

Windows

```
addObjectGridFilter.bat [location of ear file] [location of properties file]
```

UNIX

```
addObjectGridFilter.sh [location of ear file] [location of properties file]
```

UNIX

Example using eXtreme Scale installed in a stand-alone directory on UNIX:

- a. `cd wxs_home/session/bin`
- b. `addObjectGridFilter.sh /tmp/mySessionTest.ear wxs_home/session/samples/splicer.properties`

The servlet filter that is spliced in maintains defaults for configuration values. You can override these default values with configuration options that you specify in the properties file in the second argument. For a list of the parameters that you can use, see “Servlet context initialization parameters” on page 308.

You can modify and use the sample `splicer.properties` file that is provided with the eXtreme Scale installation. You can also use the **addObjectGridServlets** script, which inserts the session manager by extending each servlet. However, the recommended script is 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 container processes” on page 354 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.

7.1.0.3+ 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.

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

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 containers.

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>, 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 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 2. 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 2000.

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 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 fragmentedSession to "true" if the web application session has many attributes or attributes with large sizes. Set fragmentedSession to "false" only 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.

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 **setProperty(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. The built-in XML file packaged in the eXtreme Scale library will be 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.

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.

Fix 1+**cookieName**

Overrides the servlet specification that is designated as the cookie or encoded URL name of JSESSIONID. If the cookieName value is changed from JSESSIONID, you must configure the application servers to use this value as well.

Fix 1+**cookieDomain**

Specifies if you require sessions to be accessible across hosts. Set the value to the name of the common domain between the hosts.

Fix 1+**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.

Fix 1+**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.

7.1.0.3+ 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.

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” on page 26 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” on page 199. 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 dynamic cache service. 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.

7.1+ 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” on page 206.

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 processes” on page 354 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. See the steps to start stand-alone container servers in the *Administration Guide* for more information. 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 `wxs_install_root/customLibraries/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” on page 12 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://XS 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.

Configuring Spring integration

You can configure Spring to manage WebSphere eXtreme Scale transactions and configure clients and servers.

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” on page 321 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="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 traceFile to create and use.

statspec

Indicates the statistic specification for the server.

jmxport

Designates the unused port number through which you want to enable JMX/RMI connections. JMX enables monitoring and management from remote systems.

isCatalog

Specifies whether the particular server hosts a catalog service. The default value is false.

name

Specifies the name of the server.

haManagerPort

Sets the port number for the High Availability Manager (HA Manager).

listenerHost

Sets the host name to which the ORB should bind.

listenerPort

Sets the port to which the ORB should bind.

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 should use for all default settings.

zoneName

Defines the zone to which this server belongs.

enableSystemStreamToFile

Defines whether SystemOut and SystemErr should be sent to a file.

enableMBeans

Determines whether or not the ObjectGrid will register MBeans in this process.

serverPropertyFile

Loads the server properties from a file.

catalogServerProperties

Specifies the catalog server that hosts server.

```
<server
  id="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 which the client can connect to.

```
<catalog
  host="catalog service host name"
  port="catalog service port number"
/>
```

catalog server element

Use the catalog Server properties element to define a catalog server service.

- Number of occurrences: Zero to many
- Child element: None

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 grid to clients when routing to multiple domains.

foreignDomains

A bidirectional connection will be established to each of the foreign domains for the purpose of exchanging data in a multi-primary scenario.

clusterSecurityURL

Sets the location of the security file specific to the catalog service.

```
<catalog server
  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 grid"
  domainEndpoints="a reference to the domain name endpoints"
  foreignDomains="the name of the foreign domain"
  clusterSecurityURL="the The cluster security file location."
/>
```

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 to determine 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 meta-data 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 should reference a bean.

exceptionMapper

Specifies the ExceptionMapper plug-in that can be used for JPA-specific or database-specific exception mapping functions. This attribute should reference 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 an evictor's 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 an evictor's 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 JavaBean-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, which is useful when the key is a composite key and the value does not have the key embedded within it. 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.

See the “Spring descriptor XML file” on page 315 for descriptions of the elements and attributes defined in the Spring objectgrid.xsd file.

Spring objectgrid.xsd file

```
<xsd:schema xmlns="http://www.ibm.com/schema/objectgrid"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:beans="http://www.springframework.org/schema/beans"
  targetNamespace="http://www.ibm.com/schema/objectgrid"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">

  <xsd:import namespace="http://www.springframework.org/schema/beans"/>

  <xsd:element name="transactionManager">
    <xsd:complexType>
      <xsd:attribute name="id" type="xsd:ID"/>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="register">
    <xsd:complexType>
      <xsd:attribute name="id" type="xsd:ID"/>
      <xsd:attribute name="gridname" type="xsd:string"/>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="server">
    <xsd:complexType>
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element ref="catalog"/>
      </xsd:choice>
      <xsd:attribute name="id" type="xsd:ID"/>
      <xsd:attribute name="tracespec" type="xsd:string"/>
      <xsd:attribute name="tracefile" type="xsd:string"/>
      <xsd:attribute name="statspec" type="xsd:string"/>
      <xsd:attribute name="jmxport" type="xsd:integer"/>
      <xsd:attribute name="isCatalog" type="xsd:boolean"/>
      <xsd:attribute name="name" type="xsd:string"/>
      <xsd:attribute name="haManagerPort" type="xsd:integer"/>
      <xsd:attribute name="listenerHost" type="xsd:string"/>
      <xsd:attribute name="listenerPort" type="xsd:integer"/>
      <xsd:attribute name="maximumThreadPoolSize" type="xsd:integer"/>
      <xsd:attribute name="memoryThresholdPercentage" type="xsd:integer"/>
      <xsd:attribute name="minimumThreadPoolSize" type="xsd:integer"/>
      <xsd:attribute name="workingDirectory" type="xsd:string"/>
      <xsd:attribute name="zoneName" type="xsd:string"/>
      <xsd:attribute name="enableSystemStreamToFile" type="xsd:boolean"/>
      <xsd:attribute name="enableMBeans" type="xsd:boolean"/>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

```

        <xsd:attribute name="serverPropertyFile" type="xsd:string"/>
        <xsd:attribute name="catalogServerProperties" type="xsd:string"/>
    </xsd:complexType>
</xsd:element>

<xsd:element name="catalog">
    <xsd:complexType>
        <xsd:attribute name="host" type="xsd:string"/>
        <xsd:attribute name="port" type="xsd:integer"/>
    </xsd:complexType>
</xsd:element>

<xsd:element name="catalogServerProperties">
    <xsd:complexType>
        <xsd:choice minOccurs="0" maxOccurs="unbounded">
            <xsd:element ref="catalogServerEndPoint"/>
        </xsd:choice>
        <xsd:attribute name="id" type="xsd:ID"/>
        <xsd:attribute name="enableQuorum" type="xsd:boolean"/>
        <xsd:attribute name="heartBeatFrequencyLevel" type="xsd:integer"/>
        <xsd:attribute name="domainName" type="xsd:string"/>
        <xsd:attribute name="domainEndpoints" type="xsd:string"/>
        <xsd:attribute name="foreignDomains" type="xsd:string"/>
        <xsd:attribute name="clusterSecurityURL" type="xsd:anyURI"/>
    </xsd:complexType>
</xsd:element>

<xsd:element name="catalogServerEndPoint">
    <xsd:complexType>
        <xsd:attribute name="name" type="xsd:string"/>
        <xsd:attribute name="host" type="xsd:string"/>
        <xsd:attribute name="clientPort" type="xsd:integer"/>
        <xsd:attribute name="peerPort" type="xsd:integer"/>
    </xsd:complexType>
</xsd:element>

<xsd:element name="container">
    <xsd:complexType>
        <xsd:attribute name="id" type="xsd:ID"/>
        <xsd:attribute name="objectgridxml" type="xsd:string"/>
        <xsd:attribute name="deploymentxml" type="xsd:string"/>
        <xsd:attribute name="server" type="xsd:string"/>
    </xsd:complexType>
</xsd:element>

<xsd:element name="JPALoader">
    <xsd:complexType>
        <xsd:attribute name="id" type="xsd:ID"/>
        <xsd:attribute name="entityClassName" type="xsd:string"/>
        <xsd:attribute name="preloadPartition" type="xsd:integer"/>
    </xsd:complexType>
</xsd:element>

<xsd:element name="JPATxCallback">
    <xsd:complexType>
        <xsd:attribute name="id" type="xsd:ID"/>
        <xsd:attribute name="persistenceUnitName" type="xsd:string"/>
        <xsd:attribute name="jpaPropertyFactory" type="xsd:string"/>
        <xsd:attribute name="exceptionMapper" type="xsd:string"/>
    </xsd:complexType>
</xsd:element>

<xsd:element name="JPAEntityLoader">
    <xsd:complexType>
        <xsd:attribute name="id" type="xsd:ID"/>
        <xsd:attribute name="entityClassName" type="xsd:string"/>
        <xsd:attribute name="preloadPartition" type="xsd:integer"/>
    </xsd:complexType>
</xsd:element>

<xsd:element name="LRUEvictor">
    <xsd:complexType>
        <xsd:attribute name="id" type="xsd:ID"/>
        <xsd:attribute name="maxSize" type="xsd:integer"/>
        <xsd:attribute name="sleepTime" type="xsd:integer"/>
        <xsd:attribute name="numberOfLRUQueues" type="xsd:integer"/>
        <xsd:attribute name="useMemoryUsageThresholdEviction" type="xsd:boolean"/>
    </xsd:complexType>
</xsd:element>

<xsd:element name="LFUEvictor">
    <xsd:complexType>
        <xsd:attribute name="id" type="xsd:ID"/>
        <xsd:attribute name="maxSize" type="xsd:integer"/>
        <xsd:attribute name="sleepTime" type="xsd:integer"/>
        <xsd:attribute name="numberOfHeaps" type="xsd:integer"/>
        <xsd:attribute name="useMemoryUsageThresholdEviction" type="xsd:boolean"/>
    </xsd:complexType>
</xsd:element>

<xsd:element name="HashIndex">

```

```

<xsd:complexType>
  <xsd:attribute name="id" type="xsd:ID"/>
  <xsd:attribute name="name" type="xsd:string"/>
  <xsd:attribute name="attributeName" type="xsd:string"/>
  <xsd:attribute name="rangeIndex" type="xsd:boolean"/>
  <xsd:attribute name="fieldAccessAttribute" type="xsd:boolean"/>
  <xsd:attribute name="POJOKeyIndex" type="xsd:boolean"/>
</xsd:complexType>
</xsd:element>

</xsd:schema>

```

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="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="TransactionCallback" className="{spring}jpaTxCallback"/>
  ...
</objectGrid>

```

The spring file for the "Grid" object contains the following information:

```

<bean id="jpaTxCallback" class="com.ibm.websphere.objectgrid.jpa.JPATxCallback" scope="shard">
  <property name="persistenceUnitName" value="employeeEMPU"/>
  <property name="JPAPropertyFactory" ref="jpaPropFactory"/>
</bean>

<bean id="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 the information about the Spring descriptor file in the *Administration Guide*.

Use the JPATxCallback example from the previous section. In the previous section, the JPATxCallback bean is configured as the following:

```
<bean id="jpaTxCallback" class="com.ibm.websphere.objectgrid.jpa.JPATxCallback" scope="shard">
  <property name="persistenceUnitName" value="employeeEMPU"/>
  <property name="JPAPropertyFactory" ref="jpaPropFactory"/>
</bean>

<bean id="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="jpaTxCallback" persistenceUnitName="employeeEMPU"
  jpaPropertyFactory="jpaPropFactory" />

<bean id="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."

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="ObjectGridEventListener" className="{spring}partitionListener" />
      <backingMap name="Test" pluginCollectionRef="test" />
    </objectGrid>
  </objectGrids>

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

```

        <bean id="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="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="ogregister" gridname="Grid"/>

  <objectgrid:server id="server" isCatalog="true" name="server">
    <objectgrid:catalog host="localhost" port="2809"/>
  </objectgrid:server>

  <objectgrid:container id="container"
objectgridxml="com/ibm/ws/objectgrid/test/springshard/objectgrid.xml"
deploymentxml="com/ibm/ws/objectgrid/test/springshard/deployment.xml"
server="server"/>

  <objectgrid:LRUEvictor id="testLRUEvictor" numberOfLRUQueues="31"/>

  <bean id="partitionListener"
class="com.ibm.websphere.objectgrid.springshard.ShardListener" scope="shard"/>
</beans>

```

There were six beans defined in this spring XML file:

- objectgrid:register*: This register the default bean factory for the ObjectGrid "Grid".
- 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.
- objectgrid:catalog*: This defines an ObjectGrid catalog service endpoint, which is set to "localhost:2809".
- 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();
    }
}
```

Configuring the REST data service

Use the following links to find information about administering the REST data service. See also the application programming interface information about the RestService Mbean.

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 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:

```
CW0BJ4004I: 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 22. Properties for the REST data service

Property	Description
catalogServiceEndpoints	<p>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" on page 351 for more information.</p> <p>catalogServiceEndpoints= server1:2809,server2:2809</p>
objectGridNames	<p>The required names of the ObjectGrids to expose to the REST service. At least one ObjectGrid name is required. Separate multiple ObjectGrid names using a comma:</p> <p>ECommerceGrid,InventoryGrid</p>
objectGridClientXML	<p>The optional name of the ObjectGrid client override XML file. The name specified here will be loaded from the classpath. The default is:</p> <p>/META-INF/objectGridClient.xml. See the WebSphere eXtreme Scale product documentation for details on how to configure an eXtreme Scale client.</p>
objectGridNames	<p>The required names of the ObjectGrids to expose to the REST service. At least one ObjectGrid name is required. Separate multiple ObjectGrid names using a comma:</p> <p>ECommerceGrid,InventoryGrid</p>
objectGridClientXML	<p>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</p> <p>. See the WebSphere eXtreme Scale product documentation for details on how to configure an eXtreme Scale client.</p>
ogClientPropertyFile	<p>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}</p>

Table 22. Properties for the REST data service (continued)

Property	Description
loginType	<p>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.</p> <p>If ObjectGrid client security is enabled and loginType is set to basic, the REST service will:</p> <ul style="list-style-type: none"> • 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 <p>If ObjectGrid client security is enabled and loginType is set to none the REST service will:</p> <ul style="list-style-type: none"> • 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	<p>The optional name of the file to redirect the trace output to. The default is logs/trace.log.</p>
traceSpec	<p>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</p>
verboseOutput	<p>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.</p>
maxResultsPerCollection	<p>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.</p>
wxRestAccessRightsFile	<p>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.</p>

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” on page 339.

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 the information on defining an entity schema in the *Programming Guide*. 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 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 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 23. 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
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 24. 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.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

Table 24. Compatible EDM type to Java type (continued)

EDM Type	Java Type
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

Table 24. Compatible EDM type to Java type (continued)

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 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 CascadeType.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>
```

Administering the REST data service

About this task

Use the following links to find information about administering the REST data service. See also the RestService Mbean information.

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

Restriction: Although eXtreme Scale supports Java Standard Edition 1.4 or later, the REST data service requires 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
- eXtreme Scale, Version 7.1 or later, including the trial.

About this task

The WebSphere eXtreme Scale REST data service includes a single WAR file `wxsrestservice.war`. 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 an eXtreme Scale grid and interact with it using an eXtreme Scale client or the REST data service. See REST data services sample and tutorial 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.

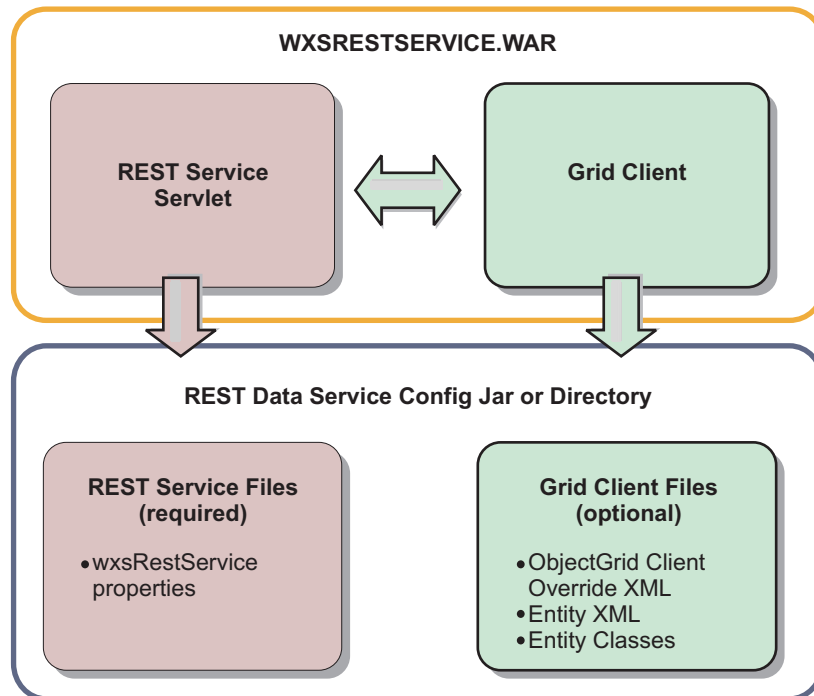


Figure 23. WebSphere eXtreme Scale REST Data Service Files

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” on page 327.

The following ObjectGrid client files are optional:

- `META-INF/objectGridClient.xml`: The ObjectGrid client override XML file is used to connect to the remote eXtreme Scale 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:

1. Any ObjectGrids that you want to expose to the REST data service.
2. 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"/>
```

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 later. These instructions also apply to deployments where WebSphere 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 the WebSphere 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 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 Chapter 7, “Configuring the deployment environment,” on page 115.
 - b. Verify that an eXtreme Scale client can connect to and access entities in the data grid. For an example, see Chapter 1, “Running the getting started sample application,” on page 1.

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” on page 339.
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. If WebSphere eXtreme Scale is integrated with the WebSphere Application Server installation, skip this step and proceed to step 5. Otherwise, continue: Add the WebSphere eXtreme Scale client runtime JAR, wsogclient.jar, and 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 fields:
 - Name: extremescale_client_v71
 - Classpath: wxs_home/lib/wsogclient.jar
 - 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. Navigate to **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 following shared libraries defined in steps 3 and 4:
 - 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" WebSphere eXtreme Scale REST data service application:
 - a. Choose application
 - If using WebSphere Application Server version 7.0: In the administrative console, click on **Applications > Application Types > WebSphere Applications**
 - If using WebSphere Application Server version 6.1: In the administrative console, click on **ApplicationsEnterprise 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.

Deploying the REST data service on WebSphere Application Server Community Edition

This topic describes how to 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 Sun 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.
- The eXtreme Scale Trial Version 7.1 with the REST data service is downloaded and extracted or the WebSphere eXtreme Scale 7.1.0.0 with cumulative fix 2 product is installed into a stand-alone directory.

Procedure

1. Configure and start an eXtreme Scale data grid.
 - a. For details on configuring an eXtreme Scale data grid for use with the REST data service, read about Chapter 7, "Configuring the deployment environment," on page 115.
 - b. Verify that an eXtreme Scale client can connect to and access entities in the data grid. For an example, see REST data services sample and tutorial.

2. Build the eXtreme Scale REST service configuration JAR or directory. See the packaging and deployment information in the “Installing the REST data service” on page 339 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**

1) 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`

2) Edit the `wasce_root/bin/setenv.sh` file

3) After the line that contains "WASCE_JAVA_HOME=", add the following:
`export JAVA_OPTS="-Djava.security.manager
-Djava.security.policy=geronimo.policy"`

Windows

1) 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`

2) Edit the `wasce_root\bin\setenv.bat` file

3) 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 25. Add Archive to Repository

Text box	Value
File	<code>wxs_home/lib/ogclient.jar</code>
Group	<code>com.ibm.websphere.xs</code>
Artifact	<code>ogclient</code>
Version	<code>7.1</code>
Type	<code>JAR</code>

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 New** link on the left side of the console window.
- d. On the **Install New Applications** page, enter the following values into the text boxes:

Table 26. Install New Applications

Text box	Value
Archive	<code>restservice_home/lib/wxsrestservice.war</code>
Plan	<code>restservice_home/gettingstarted/wasce/geronimo-web.xml</code>

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:
`CW0BJ4000I: 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:
 - 1) 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`.
 - 2) Click the **Repository** link on the left side of the console window, in the **Services** folder.
 - 3) In the **Add Archive to Repository** section, fill in the following into the input text boxes:

Table 27. 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

- 4) 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.
 - 1) 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`
 - 2) Open the WebSphere Application Server Community Edition administration console and log in.
 - 3) Click on the **Deploy New** link on the left side of the console window.
 - 4) On the **Install New Applications** page, enter the following values into the text boxes:

Table 28. Install New Applications

Text box	Value
Archive	<code>restservice_home/lib/wxsrestservice.war</code>
Plan	<code>restservice_home/gettingstarted/wasce/geronimo-web.xml</code>

- 5) Click the **Install** button.

The console page then indicates that the application has successfully installed and started.
- 6) 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.

Deploying the REST data service on Apache Tomcat

This topic describes how to configure the WebSphere eXtreme Scale REST data service on Apache Tomcat Version 5.5 or later.

About this task

- An IBM or Sun 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.
- The WebSphere eXtreme Scale Trial Version 7. with the REST data service is downloaded and extracted or the WebSphere eXtreme Scale Version 7.1.0.0 with cumulative fix 2 product is installed into a stand-alone directory.

Procedure

1. If using a Sun 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 Chapter 7, "Configuring the deployment environment," on page 115.
 - b. Verify that an eXtreme Scale client can connect to and access entities in the grid. For an example, see REST data services sample and tutorial.
3. Build the eXtreme Scale REST service configuration JAR or directory. See the packaging and deployment information in "Installing the REST data service" on page 339 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:**
 - a. `cd tomcat_root/bin`
 - b. 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
              
```
 - c. 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:**
 - a. `cd /bin`
 - b. Start the Apache Tomcat 6 configuration tool:
`tomcat6w.exe`
 - c. 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`
 - d. Click on the Start button on the Apache Tomcat 6 properties window to start the Tomcat server.

- e. 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
 - f. When the REST data service has started successfully, the following message is displayed in the system output log:
CW0BJ4000I: 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.

Chapter 8. Administering the deployment environment

Operating the product environment includes starting and stopping servers, managing the availability of the data grid, and recovering from data center failure scenarios.

About this task

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 the **startOgServer** and **stopOgServer** scripts or the embedded server API.

Before you begin

7.1.0.3+ 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 **startOgServer** or **stopOgServer** scripts. For more information about setting this environment variable, see “Starting secure servers in a stand-alone environment” on page 430.

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.

startOgServer

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” on page 205.

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. See Starting the catalog service in WebSphere Application Server for more information.

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 catalog service domains, see the information about catalog service domains in the *Product Overview*. 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” on page 356. 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
cs1:MyServer1.company.com:6601:6602,
cs2:MyServer2.company.com:6601:6602,
cs3:MyServer3.company.com:6601:6602
```

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 multi-master replication topologies” on page 227.

- **Start a secure catalog service.** See “Starting secure servers in a stand-alone environment” on page 430 for more information.
- **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” on page 364.

Starting container processes

You can start eXtreme Scale 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” on page 169.

Procedure

- **Start the container process 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, 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” on page 192. 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, 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” on page 199.

- **Start a container server programmatically.**

For more information about starting container servers programmatically, see “Using the embedded server API to start and stop servers” on page 364.

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.

startOgServer

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, instead of altering the startOgServer script, you can launch the server with arguments as follows:
-jvmArgs -cp C:\ . . . \DirectoryPOJ0s\POJ0s.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>

On the container, the **-catalogServiceEndPoints** option is used to reference the Object Request Broker (ORB) host and port on the catalog service. Each attribute is defined as follows:

serverName

Specifies a name to identify the process that you are launching.

hostName

Specifies the host name for the computer where the server is launched.

clientPort

Specifies the port that is used for peer catalog grid communication.

peerPort

This is the same as the haManagerPort. Specifies the port that is used for peer catalog grid 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
```

-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

-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>

Default: This is the same as the peerport. If this property is not set, the catalog service automatically generates an available port.

Specifies the port number the high availability manager uses.

-JMXServicePort <port>

Default: 1099

Specifies the port number for communication with Java Management Extensions (JMX). 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.

-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>

Default: localhost

Specifies the listener host for communication with Internet Inter-ORB Protocol (IIOP).

-listenerPort <port>

Default: 2809

Specifies the listener port for communication with IIOP. You must use a different port number for each JVM in your configuration.

-quorum true|false

Enables quorum on the catalog server. See "Catalog server quorums" on page 98 for more information.

-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>

Specifies a string that specifies the scope of the trace that is enabled when the server starts.

Example:

- `ObjectGrid=all=enabled`
- `ObjectGrid*=all=enabled`

-traceFile <trace file>

Specifies the path of a file in which to save trace information.

Example: `../logs/c4Trace.log`

-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`

-JMXServicePort <port>

Default: `1099`

Specifies the port number for communication with Java Management Extensions (JMX). You must use a different port number for each JVM in your configuration.

-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>

Default: localhost

Specifies the listener host for communication with Internet Inter-ORB Protocol (IIOP).

-listenerPort <port>

Default: 2809

Specifies the listener port for communication with IIOP. You must use a different port number for each JVM in your configuration.

-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 the path of a file in which to save trace information.

Example: ../logs/c4Trace.log

-traceSpec <trace specification>

Specifies a string that specifies the scope of the trace that is enabled when the server starts.

Example:

- ObjectGrid=all=enabled
- ObjectGrid*=all=enabled

-zone <zone name>

Specifies the zone to use for all of the containers within the server. See the information about zones in the *Product Overview* for more information about configuring zones.

Stopping stand-alone servers

You can use the stop0gServer 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 container servers.**

Run the **stopOgServer** script to stop the container server.

```
stopOgServer containerServer -catalogServiceEndPoints MyServer1.company.com:2809
```

Use the same script to stop multiple servers by separating a list of servers with commas:

```
stopOgServer cs0,cs1,cs2 -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 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.

- **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 “Trace options” on page 531.

- **Stop embedded servers programmatically.**

For more information about stopping embedded servers programmatically, see “Using the embedded server API to start and stop servers” on page 364.

stopOgServer script

The **stopOgServer** script stops catalog and container servers.

stopOgServer

Purpose

Use the **stopOgServer** script to stop a server. You must provide the name of the server and its catalog service endpoints.

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 servers: 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” on page 245 for more information about the security settings in this file.

-traceSpec <trace specification>

Specifies a string that specifies the scope of the trace that is enabled when the server starts.

Example:

- ObjectGrid=all=enabled
- ObjectGrid*=all=enabled

-traceFile <trace file>

Specifies the path of a file in which to save trace information.

Example: ../logs/c4Trace.log

-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

Stopping servers gracefully with the xsadmin tool

You can use the **xsadmin** tool with the **-teardown** parameter to stop a list or group of catalog and container servers. This command simplifies shutting down all or portions of a data grid, avoiding unnecessary placement and recovery actions by the catalog service that normally occur when processes are stopped or killed.

Procedure

- Stop a specific list of servers.

Provide a list of servers after the **-teardown** parameter:

```
xsadmin -teardown <server_name>[,<server_name>]
```

- Stop all the servers in a specific zone.

Use the **-fz** parameter and provide the name of the zone. The catalog server determines the servers that are running in the zone, and the **xsadmin** tool prompts you with a list of the servers in the selected zone before shutting down the servers:

```
xsadmin -teardown -fz <zone_name>
```

- Stop all the servers on a specific host.

Use the **-fh** parameter and provide the name of the host. For example to shut down all the servers on `myhost.mycompany.com`, enter `-fh myhost.mycompany.com`. The catalog server determines the servers that are running on the host, and the **xsadmin** tool prompts you with a list of the servers in the selected host before shutting down the servers:

```
xsadmin -teardown -fh <host_name>
```

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” on page 205
- “Configuring container servers in WebSphere Application Server” on page 221

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

- **Starting 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” on page 222 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” on page 205 for more information.

- **Starting 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.

- **Stopping 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 **xsadmin** tool or MBeans:

- **In the xsadmin tool:**
See “Stopping servers gracefully with the xsadmin tool” on page 363 for more information about stopping an entire data grid.
- **With Mbeans:**
Use the `tearDownServers` operation on the `PlacementServiceMBean` Mbean.

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.

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.

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.

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. 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.

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.

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. The following topic describes the available embedded server APIs.

Instantiating 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 new server with the following code.

```
Server server = ServerFactory.getInstance();
```

All properties set before the first invocation of `getInstance` are used to initialize the server.

Setting server properties

You can set the server properties until the `ServerFactory.getInstance` 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 prior to 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 ORB listener host name to use to bind to.
serverProperties.setListenerHost("host.local.domain");

// Set the ORB listener port to use to bind to.
serverProperties.setListenerPort(9010);

// Turn off all MBeans for this process.
serverProperties.setMBeansEnabled(false);

Server server = ServerFactory.getInstance();
```

Embedding 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 runtime 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();
```

Embedding the eXtreme Scale container

Issue the `Server.createContainer` method for any JVM to host multiple eXtreme Scale containers. The following code shows how to instantiate an eXtreme Scale container:

```
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: 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 prior to 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);

```

Embedding 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 in WebSphere 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” on page 364.

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.

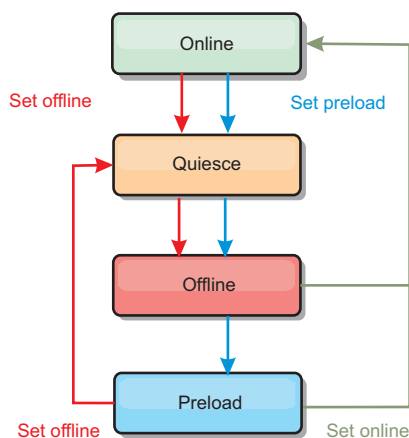


Figure 24. Availability states of an ObjectGrid instance

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” on page 153 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.

Managing data center failures

When the data center enters a failure scenario, consider overriding quorum so that container server events are not ignored. You can use the **xsadmin** tool 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” on page 224 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

The following message indicates that quorum has been lost. Look for this message in your catalog service logs.

```
CW0BJ1254W: 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.

Procedure

- Query quorum status with the **xsadmin** tool.

```
xsadmin -ch cathost -p 1099 -quorumstatus
```

Use this option to display the quorum status of a catalog service instance. One of the following outcomes is displayed:

- Quorum is disabled: The catalog servers are running in a quorum-disabled mode. Quorum disabled mode is a development or single data center mode. Do not use quorum disabled mode for multiple data center configurations.
 - Quorum is enabled and the catalog server has quorum: Quorum is enabled and the system is working normally.
 - Quorum is enabled but the catalog server is waiting for quorum: Quorum is enabled and quorum has been lost.
 - Quorum is enabled and the quorum is overridden: Quorum is enabled and quorum has been overridden.
 - Quorum status is outlawed: When a brown out occurs, splitting the catalog service into two partitions, A and B. The catalog server A has overridden quorum. The network partition resolves and the server in the B partition is outlawed, requiring a JVM restart. It also occurs if the catalog JVM in B restarts during the brown out and then the brown out clears.
- Override quorum with the **xsadmin** tool.

```
xsadmin -ch cathost -p 1099 -overridequorum
```

Running this command forces the surviving catalog servers to re-establish a quorum.

- Diagnose quorum with the **xsadmin** tool.

- **Display a list of the core groups:**

Use the **-coregroups** option to display a list of all the core groups for the catalog server.

```
xsadmin -ch cathost -p 1099 -coregroups
```

– **Teardown servers:**

Use the **-teardown** option to remove a server manually from the data grid. Removing a server from the grid is usually not necessary. Servers are automatically removed when they are detected as failed, but the command is provided for use under the guidance of IBM support. See “Stopping servers gracefully with the `xsadmin` tool” on page 363 for more information about using this command.

```
xsadmin -ch cathost -p 1099 -g Grid -teardown server1,server2,server3
```

– **Display the route table:**

Use the **-routetable** option to display 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.

```
xsadmin -ch cathost -p 1099 -g myGrid -routetable
```

– **Check the map sizes:**

Use the **-mapsizes** option to 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.

```
xsadmin -ch cathost -p 1099 -g myGrid -m myMapSet -mapsizes myMap
```

– **Set trace strings:**

Use the **-settracespec** option to set the trace settings for all JVMs that match the filter specified for the `xsadmin` command. This setting changes the trace settings only, until another command is used or the JVMs modified fail or stop.

```
xsadmin -ch cathost -p 1099 -g myGrid -fh host1 -settracespec  
ObjectGrid*=event=enabled
```

This string enables trace for all JVMs on the server with the specified host name, in this case `host1`.

– **Display unassigned shards:**

Use the **-unassigned** option to display the 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. For example, if you start JVMs on a single physical server while in production mode, then only primary shards can be placed. Replicas are not assigned until JVMs start on a second physical server. The placement service places replicas only on JVMs with different IP addresses than the JVMs that are hosting the primary shards. Having no JVMs in a zone can also cause shards to be unassigned.

```
xsadmin -ch cathost -p 1099 -g myGrid -unassigned
```

Forcing placement to occur

Consider forcing placement to occur when you are maintaining servers and must temporarily take some of your servers offline.

Before you begin

7.1.0.2+ You must be using Version 7.1.0.2 or later to run the `xsadmin -triggerPlacement` command.

About this task

In a typical configuration, you can use the **numInitialContainers** attribute in the deployment policy descriptor XML file to prevent shard placement on the data grid until the specified number of container servers are running. Using this setting can reduce unnecessary processing and rebalancing during a cold start. When this setting is configured, placement does not occur when the number of available container servers is lower than the **numInitialContainers** value.

You might want placement to continue running even when the available number of container servers drops below the **numInitialContainers** value. For example, you take some of your servers offline for maintenance. You want placement to occur on the remaining servers in the configuration. Instead of temporarily changing your configuration files to reduce the **numInitialContainers** value, you can use the **xsadmin -triggerPlacement** command.

Placement occurs during failover conditions regardless of the **numInitialContainers** value and the number of container servers that are running. The **numInitialContainers** value is used during cold startups or when you are starting additional container servers.

Procedure

1. Trigger placement to occur when you stop container servers. When you stop container servers, the number of running container servers can drop below the **numInitialContainers** value, which causes shard placement to stop. For example, in an environment that has four physical servers each running three container servers, you might set the **numInitialContainers** value to 12. If you must take two of the physical servers offline, the remaining number of running container servers is down to six servers.

To activate shard placement on the remaining container servers in the configuration, run the following command:

```
xsadmin -triggerPlacement myOG myMapSet
```

2. Trigger placement to occur when you start additional container servers, but the **numInitialContainers** value is still not met. Continuing with the previous example, you finish the maintenance on one of the physical servers, and start the three container servers. Starting these container servers brings the total number of container servers to nine servers. Because the value is still below the **numInitialContainers** value, you must run the following command again to run placement on the three new container servers:

```
xsadmin -triggerPlacement myOG myMapSet
```

3. When the number of container servers reaches the **numInitialContainers** value, no action is needed. When you finish maintenance on the second physical server and start the three container servers, placement occurs automatically because the **numInitialContainers** value is met.

What to do next

You can monitor the placement of your environment with the **xsadmin -placementStatus** command.

Administering programmatically 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.

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>
```

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 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.

```
package com.ibm.ws.objectgrid.test.catalogserver;

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);

        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);
    }
}
}
```

Figure 25. *CollectPlacementPlan.java*

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`:

```

package com.ibm.ws.objectgrid.test.catalogserver;

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);

        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"));
            }
        }
    }
}

```

Figure 26. *CollectContainerStatus.java*

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:

```

package com.ibm.ws.objectgrid.test.catalogserver;

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 CollectPlacementPlan {
    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);

        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);
    }
}

```

Figure 27. *CollectPlacementPlan.java*

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” on page 443.

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 “Monitoring with the **xsadmin** utility” on page 470 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.

Chapter 9. Securing the deployment environment

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.

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 `xsadmin` tool in WebSphere Application Server environment

Time required

This tutorial takes approximately 4 hours from start to finish.

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 `xsadmin` tool 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 **xsadmin** tool 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
- WebSphere eXtreme Scale Version 7.0 or Version 7.1 with interim fix PM20613 must be installed on the WebSphere Application Server nodes.
- 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” on page 427.

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.

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” on page 206 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.

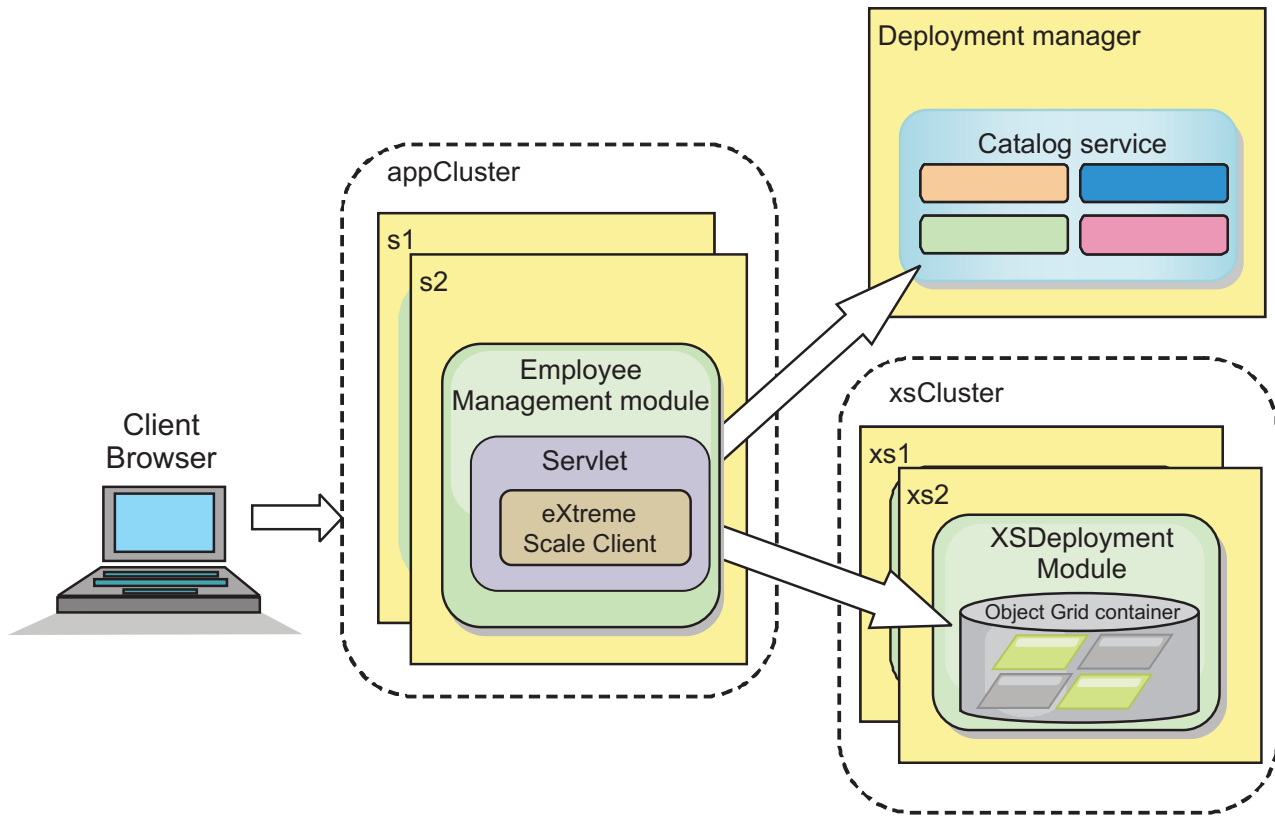


Figure 28. Tutorial topology

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 WASecurity.zip and security.zip files. You can download the sample application from the WebSphere eXtreme Scale wiki.
2. Extract the WASecurity.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” on page 222.

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 **User account repository**. 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, select 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.

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 EmployeeManagement 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.

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” on page 245 for more information about the client properties file.

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” on page 427 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 is 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.

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” on page 450 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`.

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” on page 427 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” on page 199 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 container 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.

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.

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.

```
CW0BJ0913I: 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:

```
CW0BJ0913I: 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.

```
CW0BJ0924I: 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 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.

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 which does 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.

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.

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=IBMJSE2
#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.

Lesson 3.3: Check the `orb.properties` file

With WebSphere eXtreme Scale Version 7.0 or earlier, you must verify that the Secure Sockets Layer (SSL) works correctly between WebSphere Application Server and WebSphere eXtreme Scale servers. You must edit the `orb.properties` file in the `JAVA_HOME/jre/lib` directory must contain the specific properties.

If you are using WebSphere eXtreme Scale Version 7.1 or later, changing the orb.properties file is not necessary. The attributes are automatically added at run time if they are not present in the orb.properties file.

orb.properties file:

The following lines in bold text are not in the default orb.properties file in the Java Development Kit that is shipped by WebSphere Application Server. Add these lines to your orb.properties file:

```
# IBM JDK properties
org.omg.CORBA.ORBClass=com.ibm.CORBA.iiop.ORB
org.omg.CORBA.ORBSingletonClass=com.ibm.rmi.corba.ORBSingleton
javax.rmi.CORBA.StubClass=com.ibm.rmi.javax.rmi.CORBA.StubDelegateImpl
javax.rmi.CORBA.PortableRemoteObjectClass=com.ibm.rmi.javax.rmi.PortableRemoteObject
javax.rmi.CORBA.UtilClass=com.ibm.ws.orb.WSUtilDelegateImpl

# WS Plugins
com.ibm.CORBA.ORBPluginClass.com.ibm.ws.orbimpl.transport.WSTransport
com.ibm.CORBA.ORBPluginClass.com.ibm.ws.orbimpl.WSORBPropertyManager
com.ibm.CORBA.ORBPluginClass.com.ibm.ISecurityUtilityImpl.SecurityPropertyManager
org.omg.PortableInterceptor.ORBInitializerClass.com.ibm.ws.objectgrid.corba.ObjectGridInitializer

# WS ORB & Plugins properties
com.ibm.ws.orb.transport.ConnectionInterceptorName=com.ibm.ISecurityLocalObjectBaseL13Impl.SecurityConnectionInterceptor
com.ibm.ws.orb.transport.WSSSLClientSocketFactoryName=com.ibm.ws.security.orbssl.WSSSLClientSocketFactoryImpl
com.ibm.CORBA.enableLocateRequest=true
com.ibm.CORBA.ORBCharEncoding=UTF8
com.ibm.CORBA.ForceTunnel=never
com.ibm.CORBA.TransportMode=Pluggable
com.ibm.CORBA.ServerName=ogserver
org.omg.PortableInterceptor.ORBInitializerClass.com.ibm.ISecurityLocalObjectBaseL13Impl.SecurityComponentFactory
```

Lesson checkpoint:

For your WebSphere eXtreme Scale Version 7.0 or earlier configuration, you edited the orb.properties file to contain the necessary properties.

Lesson 3.4: 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” on page 391 for more information about running and installing the sample application.

Lesson checkpoint:

You ran the sample application with transport security enabled.

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 the client authorization reference in the *Programming Guide*.

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 eXtreme Scale runtime 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.

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” on page 391 for more information about deploying applications.

Lesson checkpoint:

You enabled security on the `ObjectGrid`, which also enables authorization on the data grid.

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_homesecurity` 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 the following generic JVM arguments:
`-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>/EmployeeManagementWeb/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.

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. 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>/EmployeeManagementWeb/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.

Module 5: Use the `xsadmin` tool to monitor data grids and maps

You can use the `xsadmin` tool to show the primary data grids and map sizes of the Grid data grid. The `xsadmin` 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 `xsadmin` 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 `xsadmin` tool. Use the `-primaries` parameter as in the following examples:

Linux UNIX

```
xsadmin.sh -g Grid -m mapSet -dmgr -primaries
```

Windows

```
xsadmin.bat -g Grid -m mapSet -dmgr -primaries
```

Because the catalog server runs in the deployment manager, the `-dmgr` parameter is required. With this parameter, the WebSphere eXtreme Scale run time connects to the WebSphere Application Server MBean server. If your `BOOTSTRAP_PORT`, is set to a value other than 2809, specify the port number with the `-p` parameter.

Before you can view the output, you are prompted to log in with your WebSphere Application Server ID and password.

3. View the command output.

```
*** Showing all primaries for grid - Grid & mapset - mapSet
Partition Container Host Server
0 myCell102\myNode04\xs2_C-1 myhost.mycompany.com myCell102\myNode04\xs2
1 myCell102\myNode04\xs2_C-1 myhost.mycompany.com myCell102\myNode04\xs2
2 myCell102\myNode04\xs2_C-1 myhost.mycompany.com myCell102\myNode04\xs2
3 myCell102\myNode04\xs2_C-1 myhost.mycompany.com myCell102\myNode04\xs2
4 myCell102\myNode04\xs2_C-1 myhost.mycompany.com myCell102\myNode04\xs2
```

If you see the following exception on WebSphere eXtreme Scale Version 7.1, verify that the PM20613 interim fix is installed:

```
Could not load class: com.ibm.ws.security.auth.ContextManagerImpl
```

Lesson checkpoint

You used the **xsadmin** tool in WebSphere Application Server by using the **-dmgr** parameter.

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” on page 381. 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 **xsadmin** tool 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.

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” on page 381.

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 **xsadmin** tool 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.
- WebSphere eXtreme Scale Version 7.0 or Version 7.1. 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” on page 427.

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.

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 WebSphere eXtreme Scale wiki.

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” on page 206

for more information about creating a catalog service domain that consists of remote servers.

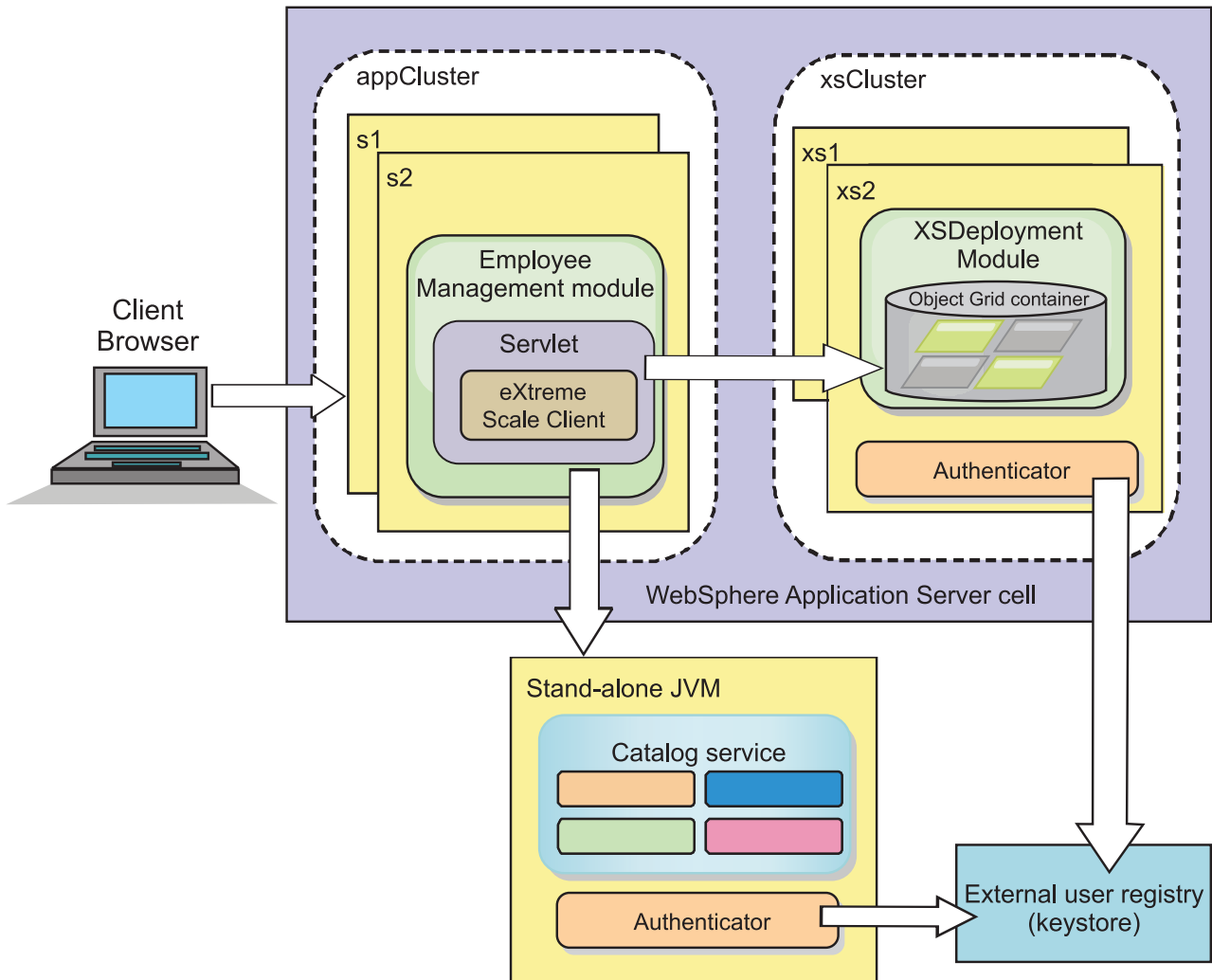


Figure 29. Tutorial topology

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.

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” on page 26 for more information.

Configure WebSphere Application Server security.

1. In the WebSphere Application Server administrative console, click **Security > Global Security**.
2. Select **Federated repositories** as the **User account repository**. Click **Set as current**.
3. Click **Configure..** to go to the **Federated repositories** panel.
4. Enter the **Primary administrative user name**, for example, admin. Click **Apply**.
5. When prompted, enter the administrative user password and click **OK**. Save your changes.
6. On the **Global Security** page, verify that **Federated repositories** setting is set to the current user account registry.
7. 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.
8. 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 Version 7.1 or later**

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.

Note: If you are using Version 7.1 without fix 1 applied, set the JMX Port to 16099.

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.

- **Define a catalog service domain in WebSphere eXtreme Scale Version 7.0**

You can create the catalog.services.cluster custom property in a cell, node, or server scope. For this example, you can create the catalog.server.cluster custom property as a cell custom property.

1. In the WebSphere Application Server administrative console, click **System administration > Cell > Custom properties > New**.
2. Specify the name as catalog.services.cluster and the value in the appropriate form, using the defined attributes. For example, the value might be:
cs1:[your_node_hostname]:16601:16602:16809. See Version 7.0: Starting the catalog service process in a WebSphere Application Server environment

Lesson checkpoint:

You enabled security in WebSphere Application Server, and created the server topology for WebSphere eXtreme Scale.

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

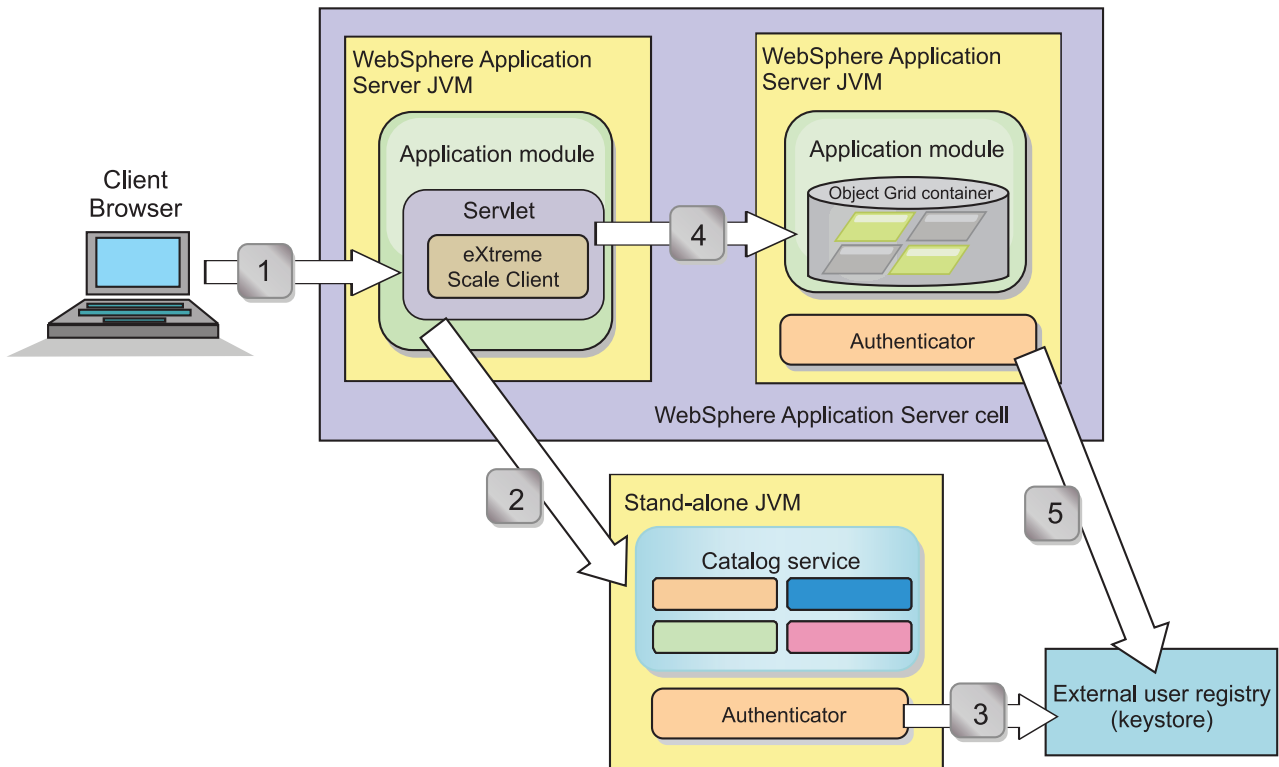


Figure 30. Authentication flow

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.

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. **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`
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.

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” on page 450 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” on page 199 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. In 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.

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 **-clusterFile** 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.

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`.
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.

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.xls.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” on page 414.
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:

```
CW0BJ0913I: 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.

```
CW0BJ0924I: The client property file {0} has been loaded.
```

If you are using WebSphere eXtreme Scale Version 7.0, the English-only `CW0BJ9000I` 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 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.

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    0 [KeyStoreLoginModule] initialize: Successfully loaded key store
SystemOut    0 [KeyStoreLoginModule] login: entry
SystemOut    0 [KeyStoreLoginModule] login: user entered user name: manager
SystemOut    0   Print out the certificates:
...
```

Lesson checkpoint:

You installed and ran the sample application.

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.

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.

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=IBMJSE2
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.

Lesson 3.3: Check the orb.properties file

With WebSphere eXtreme Scale Version 7.0 or earlier, you must verify that the Secure Sockets Layer (SSL) works correctly between WebSphere Application Server and WebSphere eXtreme Scale servers. You must edit the orb.properties file in the JAVA_HOME/jre/lib directory must contain the specific properties.

If you are using WebSphere eXtreme Scale Version 7.1 or later, changing the orb.properties file is not necessary. The attributes are automatically added at run time if they are not present in the orb.properties file.

orb.properties file:

The following lines in bold text are not in the default orb.properties file in the Java Development Kit that is shipped by WebSphere Application Server. Add these lines to your orb.properties file:

```
# IBM JDK properties
org.omg.CORBA.ORBClass=com.ibm.CORBA.iiop.ORB
org.omg.CORBA.ORBSingletonClass=com.ibm.rmi.corba.ORBSingleton
javax.rmi.CORBA.StubClass=com.ibm.rmi.javax.rmi.CORBA.StubDelegateImpl
javax.rmi.CORBA.PortableRemoteObjectClass=com.ibm.rmi.javax.rmi.PortableRemoteObject
javax.rmi.CORBA.UtilClass=com.ibm.ws.orb.WSUtilDelegateImpl

# WS Plugins
com.ibm.CORBA.ORBPluginClass.com.ibm.ws.orbimpl.transport.WSTransport
com.ibm.CORBA.ORBPluginClass.com.ibm.ws.orbimpl.WSORBPropertyManager
com.ibm.CORBA.ORBPluginClass.com.ibm.ISecurityUtilityImpl.SecurityPropertyManager
org.omg.PortableInterceptor.ORBInitializerClass.com.ibm.ws.objectgrid.corba.ObjectGridInitializer

# WS ORB & Plugins properties
com.ibm.ws.orb.transport.ConnectionInterceptorName=com.ibm.ISecurityLocalObjectBaseL13Impl.SecurityConnectionInterceptor
com.ibm.ws.orb.transport.WSSSLClientSocketFactoryName=com.ibm.ws.security.orbssl.WSSSLClientSocketFactoryImpl
com.ibm.CORBA.enableLocateRequest=true
com.ibm.CORBA.ORBCharEncoding=UTF8
com.ibm.CORBA.ForceTunnel=never
```

```
com.ibm.CORBA.TransportMode=Pluggable
com.ibm.CORBA.ServerName=ogserver
org.omg.PortableInterceptor.ORBInitializerClass.com.ibm.ISecurityLocalObjectBaseL13Impl.SecurityComponentFactory
```

Lesson checkpoint:

For your WebSphere eXtreme Scale Version 7.0 or earlier configuration, you edited the `orb.properties` file to contain the necessary properties.

Lesson 3.4: 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” on page 416 for more information about running and installing the sample application.

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 the client authorization reference in the *Programming Guide*.

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 eXtreme Scale runtime 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.

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 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” on page 416 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.

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.auth.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>/EmployeeManagementWeb/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 first name Joe, and the last name 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 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 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.

Module 5: Use the xsadmin tool to monitor data grids and maps

You can use the `xsadmin` tool to show the primary data grids and map sizes of the Grid data grid. The `xsadmin` 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 **xsadmin** 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 **xsadmin** 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 **xsadmin** tool. Use the **-primaries** parameter as in the following examples:

Linux UNIX

```
./xsadmin.sh -g Grid -m mapSet -primaries -username manager -password manager1 -p 16099
```

Windows

```
xsadmin.bat -g Grid -m mapSet -primaries -username manager -password manager1 -p 16099
```

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 **xsadmin** tool. Use the **-mapsizes** parameter as in the following examples:

Linux UNIX

```
./xsadmin.sh -g Grid -m mapSet -mapsizes -username manager -password manager1 -p 16099
```

Windows

```
xsadmin.bat -g Grid -m mapSet -mapsizes -username manager -password manager1 -p 16099
```

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 **-mapsizes** option gets the map size from each container server, which requires the WebSphere Application Server security.

5. View the command results.

```
*****Displaying Results for Grid - Grid, MapSet - mapSet*****

*** Listing Maps for xddev15Cell02\xddev15Node04\xs1 ***
Map Name Partition Map Size Used Bytes (B) Shard Type
Map1      0          0          0          Primary
Map1      1          0          0          Primary
Map1      2          1        272          Primary
Map1      3          0          0          Primary
Map1      4          0          0          Primary
Server Total: 1 (272B)
```

- 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
```

- Optional: If you are using the `xsadmin` command on a WebSphere eXtreme Scale stand-alone installation, then you must add the following options to indicate the truststore, truststore type, and truststore password.

```
-trustStore
-trustStoreType
-trustStorePassword
```

You can also set up a properties file to contain these parameters. See “`xsadmin` utility reference” on page 474 for more information.

Lesson checkpoint

You used the `xsadmin` tool to monitor data grids and maps in your configuration.

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.

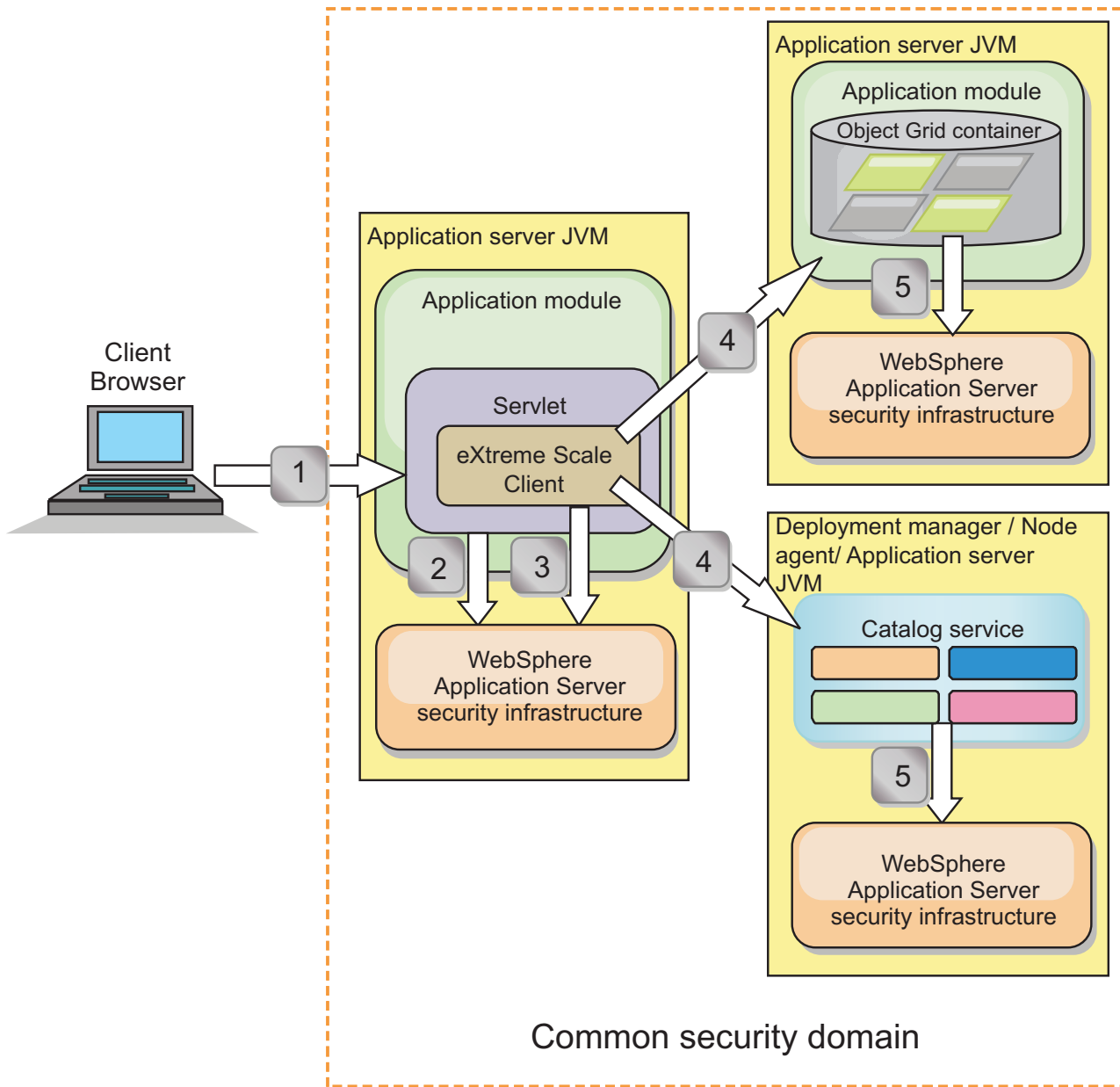


Figure 31. Authentication flow for servers within the same security domain

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. 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 page 381.

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 the information about local security in the *Programming Guide*. 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” on page 381 for more information.

Enabling local security

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.

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` command.

Before you begin

7.1.0.3+ If you are using an external client security provider for authentication or authorization, define the `CLIENT_AUTH_LIB` environment variable. Open a command-line or terminal window and run the command that is appropriate for your operating system:

- Windows

```
set CLIENT_AUTH_LIB=<path_to_security_JAR_or_classes>
```

- **UNIX**

```
set CLIENT_AUTH_LIB=<path_to_security_JAR_or_classes>  
export CLIENT_AUTH_LIB
```

When the **startOgServer** and **stopOgServer** commands run, this variable is appended to the classpath.

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” on page 199 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`.

- 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`.

Stopping secure servers

Stopping a secure catalog servers or container servers requires one security configuration file.

Procedure

Stop a secure catalog server or container server.

Stopping a secure server requires the following security configuration file:

- **Client property file:** The client property file can be used to configure the client security properties. The client security properties are required for a client to connect to a secure server. Refer to the “Client properties file” on page 245 for more details.

Specify the location of these configuration files by providing the following argument to the **stopOgServer** script:

-clientSecurityFile

Specifies the path to the client properties file that defines security properties for the client. The file name that you specify for this property is in plain file path format, such as `../security/objectGridClient.properties`.

Example

```
stopOgServer.bat|sh cs1 -catalogServiceEndPoints
cs1:MyServer1.company.com:6601:6602,
cs2:MyServer2.company.com:6601:6602,
cs3:MyServer3.company.com:6601:6602
-clientSecurityFile ../security/objectGridClient.properties
```

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

- Start a secure catalog service in WebSphere Application Server.

A catalog server contains two different levels of security information:

 - `-Dobjectgrid.cluster.security.xml.url`: Specifies the location of the `objectGridSecurity.xml` file, which describes the security properties 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 should be in an URL format, such as `file:///tmp/og/objectGridSecurity.xml`.
 - `-Dobjectgrid.server.props`: Specifies the server property file that 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`.
 1. In the WebSphere Application Server administrative console, click **System administration**. Click the process on which the catalog server is deployed, such as the deployment manager.
 2. Click **Java and process management > Process definition > Java Virtual Machine**.
 3. Type the properties in the **Generic JVM arguments** field. An example of the values you might add follows:


```
-Dobjectgrid.cluster.security.xml.url=file:///tmp/og/objectGridSecurity.xml
-Dobjectgrid.server.props=/tmp/og/catalog.server.props
```
 4. Click **OK** and save your changes.
- Start a secure container server in WebSphere Application Server.

A container server, when connecting to the catalog server, inherits the security configuration that is in the `objectGridSecurity.xml` file, such as the authenticator configuration or login session timeout settings. You must also define server-specific security properties for specific container servers in the `-Dobjectgrid.server.props` property.

The file name specified for this property is just in plain file path format, such as `c:/tmp/og/server.props`.

Follow the same steps as above to add the security property to the generic JVM arguments.

1. Open the Java virtual machine page for the server. In the WebSphere Application Server administrative console, click **Servers > Application servers > server_name > Java and process management > Process definition > Java Virtual Machine**
2. Type the property in the **Generic JVM arguments** field. An example of the values you might add follows:
`-Dobjectgrid.server.props=/opt/wxs/security/server2.props`
3. Click **OK** and save your changes.

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 to 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 JCKES type keystore to store the key pair, a private key and public key, and a secret key. The keystore has to be the JCKES 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.

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” on page 199 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 JCKES type keystore to store the key pair, a private key and public key, and a secret key. The keystore has to be the JCKES 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.

Application client authentication

Application client authentication consists of enabling client-server security and credential authentication, and configuring an authenticator and a system credential generator.

Enabling client-server security

You must enable security on both the client and server to successfully authenticate with the ObjectGrid.

Enable client security

WebSphere eXtreme Scale provides a client property sample file, the `sampleClient.properties` file, in the `was_root/optionalLibraries/ObjectGrid/properties` directory for a WebSphere Extended Deployment installation, or the `/ObjectGrid/properties` directory in a mixed-server installation. You can modify this template file with appropriate values. Set the `securityEnabled` property in the `objectgridClient.properties` file to `true`. The `securityEnabled` property indicates if security is enabled. When a client connects to a server, the value on the client and server side must be set both `true` or both `false`. For example, if the connected server security is enabled, the property value must be set to `true` on the client side for the client to connect to the server.

The `com.ibm.websphere.objectgrid.security.config.ClientSecurityConfiguration` interface represents the `security.ogclient.props` file. You can use the `com.ibm.websphere.objectgrid.security.config.ClientSecurityConfigurationFactory`

public API to create an instance of this interface with default values, or you can create an instance by passing the ObjectGrid client security property file. The security.ogclient.props file contains other properties. See the ClientSecurityConfiguration API Documentation and ClientSecurityConfigurationFactory API Documentation for more details.

Enabling server security

To enable the security on the server side, you can set the **securityEnabled** property in the security.xml file to true. Use a security descriptor XML file to specify the data grid security configuration to isolate the grid-wide security configuration from the non-security configuration.

Enabling credential authentication

After the eXtreme Scale client retrieves the Credential object using the CredentialGenerator object, the 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 Credential object. This Subject object is then used for authorizing the request.

Set the **credentialAuthentication** property on the client and server properties files to enable the credential authentication. For more information, see “Client properties file” on page 245 and “Server properties file” on page 199.

The following table provides which authentication mechanism to use under different settings.

Table 29. Credential authentication under client and server settings

Client credential authentication	Server credential authentication	Result
No	Never	Disabled
No	Supported	Disabled
No	Required	Error case
Supported	Never	Disabled
Supported	Supported	Enabled
Supported	Required	Enabled
Required	Never	Error case
Required	Supported	Enabled
Required	Required	Enabled

Configuring an authenticator

The eXtreme Scale server uses the Authenticator plug-in to authenticate the Credential object. An implementation of the Authenticator interface gets the Credential object and then authenticates it to a user registry, for example, a Lightweight Directory Access Protocol (LDAP) server, and so on. eXtreme Scale does not provide a registry configuration. Connecting to a user registry and authenticating to it must be implemented in this plug-in.

For example, one 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 can use Java Authentication and Authorization Service (JAAS) login modules. A Subject object is returned as a result of authentication.

You can configure the authenticator in the security descriptor XML file, as shown in the following example:

```
<?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>
```

Use the **-clusterSecurityFile** option when starting a secure server to set the security XML file. See the Java SE security tutorial in the *Product Overview* for more information.

Configuring a system credential generator

The system credential generator is used to represent a factory for the system credential. A system credential is similar to an administrator credential. You can configure the SystemCredentialGenerator element in the catalog security XML, as shown in the following example:

```
<systemCredentialGenerator className ="com.ibm.websphere.objectgrid.security.plugins.
  builtins.UserPasswordCredentialGenerator">
  <property name="properties" type="java.lang.String" value="manager manager1"
    description="username password" />
</systemCredentialGenerator>
```

For demonstration purposes, the user name and password are stored in clear text. Do not store the user name and password in clear text in a production environment.

WebSphere eXtreme Scale provides a default system credential generator, which uses the server credentials. If you do not explicitly specify the system credential generator, this default system credential generator is used.

Application client authorization

Application client authorization consists of ObjectGrid permission classes, authorization mechanisms, a permission checking period, and access by creator only authorization.

For eXtreme Scale, authorization is based on the Subject object and permissions. The product supports two kinds of authorization mechanisms: Java Authentication and Authorization Service (JAAS) and custom authorization.

ObjectGrid permission classes

Authorization is based on permissions. There are four different types of permission classes as follows.

- The MapPermission class represents permissions to access the data in ObjectGrid maps.
- The ObjectGridPermission class represents permissions to access ObjectGrid.
- The ServerMapPermission class represents permissions to access ObjectGrid maps on the server side from a client.
- The AgentPermission class represents permissions to start an agent on the server side.

For more information on APIs and associated permissions, see the topic on client authorization programming in the *Programming Guide*.

Permission checking period

eXtreme Scale supports caching the map permission checking results for performance reasons. Without this mechanism, when a method that is listed in the list of methods that for your particular permission class is called, the runtime calls the configured authorization mechanism to authorize access. With this permission checking period set, the authorization mechanism is called periodically based on the permission checking period. For a list of methods for each permission class, see the topic on client authorization programming in the *Programming Guide*.

The permission authorization information is based on the Subject object. When a client tries to access the methods, the eXtreme Scale runtime looks up the cache based on the Subject object. If the object cannot be found in the cache, the runtime checks the permissions granted for this Subject object, and then stores the permissions in a cache.

The permission checking period must be defined before the ObjectGrid is initialized. The permission checking period can be configured in two ways:

You can use the ObjectGrid XML file to define an ObjectGrid and set the permission check period. In the following example, the permission check period is set to 45 seconds:

```
<objectGrids>
<objectGrid name="secureClusterObjectGrid" securityEnabled="true"
authorizationMechanism="AUTHORIZATION_MECHANISM_JAAS"
permissionCheckPeriod="45">
  <bean id="TransactionCallback"
className="com.ibm.websphere.samples.objectgrid.HeapTransactionCallback" />
...
</objectGrids>
```

If you want to create an ObjectGrid with APIs, call the following method to set the permission checking period. This method can be called only before the ObjectGrid instance is initialized. This method applies only to the local eXtreme Scale programming model when you instantiate the ObjectGrid instance directly.

```
/**
 * This method takes a single parameter indicating how often you
 * want to check the permission used to allow a client access. If the
 * parameter is 0 then every single get/put/update/remove/evict call
 * asks the authorization mechanism, either JAAS authorization or custom
 * authorization, to check if the current subject has permission. This might be
 * prohibitively expensive from a performance point of view depending on
 * the authorization implementation, but if you need to have ever call check the
 * authorization mechanism, then set the parameter to 0.
 * Alternatively, if the parameter is > 0 then it indicates the number
 * of seconds to cache a set of permissions before returning to
 * the authorization mechanism to refresh them. This value provides much
 * better performance, but if the back-end
 * permissions are changed during this time then the ObjectGrid can
 * allow or prevent access even though the back-end security
 * provider was modified.
```

```

*
* @param period the permission check period in seconds.
*/
void setPermissionCheckPeriod(int period);

```

Access by creator only authorization

Access by creator only authorization ensures that only the user (represented by the Principal objects associated with it) who inserts the entry into the ObjectGrid map can access (read, update, invalidate and remove) that entry.

The existing ObjectGrid map authorization model is based on the access type but not data entries. In other words, a user has a particular type of access, such as read, write, insert, delete, or invalidate, to either all the data in the map or none of the data. However, eXtreme Scale does not authorize users for individual data entry. This feature offers a new way to authorize users to data entries.

In a scenario where different users access different sets of data, this model can be useful. When the user loads data from the persistent store into the ObjectGrid maps, the access can be authorized by the persistent store. In this case, there is no need to do another authorization in the ObjectGrid map layer. You need only ensure that the person who loads the data into the map can access it by enabling the access by creator only feature.

Creator only mode attribute values:

disabled

The access by creator only feature is disabled.

complement

The access by creator only feature is enabled to complement the map authorization. In other words, both map authorization and access by creator only feature takes effect. Therefore, you can further limit the operations to the data. For example, the creator cannot invalidate the data.

supersede

The access by creator only feature is enabled to supersede the map authorization. In other words, the access by creator only feature supersedes the map authorization; no map authorization occurs.

You can configure the access by creator only mode in two ways:

By XML file:

You can use the ObjectGrid XML file to define an ObjectGrid and set the access by creator only mode to either disabled, complement, or supersede, as shown in the following example:

```

<objectGrids>
  <objectGrid name="secureClusterObjectGrid" securityEnabled="true"
    accessByCreatorOnlyMode="supersede"
    <bean id="TransactionCallback"
      classname="com.ibm.websphere.samples.objectgrid.HeapTransactionCallback" />
    ...
  </objectGrid>
</objectGrids>

```

Programmatically:

If you want to create an ObjectGrid programmatically, you can call the following method to set the access by creator only mode. Calling this method applies only to the local eXtreme Scale programming model when you directly instantiate the ObjectGrid instance:


```

/**
 * Set the "access by creator only" mode.
 * Enabling "access by creator only" mode ensures that only the user (represented
 * by the Principals associated with it), who inserts the record into the map,
 * can access (read, update, invalidate, and remove) the record.
 * The "access by creator only" mode can be disabled, or can complement the
 * ObjectGrid authorization model, or it can supersede the ObjectGrid
 * authorization model. The default value is disabled:
 * {@link SecurityConstants#ACCESS_BY_CREATOR_ONLY_DISABLED}.
 * @see SecurityConstants#ACCESS_BY_CREATOR_ONLY_DISABLED
 * @see SecurityConstants#ACCESS_BY_CREATOR_ONLY_COMPLEMENT
 * @see SecurityConstants#ACCESS_BY_CREATOR_ONLY_SUPERSEDE
 *
 * @param accessByCreatorOnlyMode the access by creator mode.
 *
 * @since WAS XD 6.1 FIX3
 */
void setAccessByCreatorOnlyMode(int accessByCreatorOnlyMode);

```

To further illustrate, consider a scenario in which an ObjectGrid map account is in a banking grid, and Manager1 and Employee1 are the two users. The eXtreme Scale authorization policy grants all access permissions to Manager1, but only read access permission to Employee1. The JAAS policy for the ObjectGrid map authorization is shown the following example:

```

grant codebase "http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction"
    Principal com.acme.PrincipalImpl "Manager1" {
        permission com.ibm.websphere.objectgrid.security.MapPermission
            "banking.account", "all"
    };
grant codebase "http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction"
    Principal com.acme.PrincipalImpl "Employee1" {
        permission com.ibm.websphere.objectgrid.security.MapPermission
            "banking.account", "read, insert"
    };

```

Consider how the access by creator only feature affects authorization:

- **disabled** If the access by creator only feature is disabled, the map authorization is no different. The user "Manager1" can access all the data in the "account" map. The user "Employee1" can read and insert all the data in the map but cannot update, invalidate, remove any data in the map.
- **complement** If the access by creator only feature is enabled with "complement" option, both the map authorization and access by creator only authorization will take effect. The user "Manager1" can access the data in the "account" map, but only if the user alone loaded them into the map. The user "Employee1" can read the data in the "account" map, but only if that user alone loaded them into the map. (However, this user cannot update, invalidate, or remove any data in the map.)
- **supersede** If the access by creator only feature is enabled with "supersede" option, the map authorization will not be enforced. The access by creator only authorization will be the only authorization policy. The user "Manager1" has the same privilege as in the "complement" mode: this user can access the data in the "account" map only if the same user loaded the data into the map. However, the user "Employee1" now has full access to the data in the "account" map if this user loaded them into the map. In other words, the authorization policy defined in the Java Authentication and Authorization Service (JAAS) policy will then not be enforced.

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.

Enable TLS/SSL in both directions

TLS/SSL is sometimes enabled in one direction. For example, the server public certificate is imported in the client truststore, but not 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 new connection. The eXtreme Scale server then acts as a client. Therefore, you must import the client public certificate into the server truststore.

Enable transport security for Sun JDK

WebSphere eXtreme Scale requires IBM Java Secure Sockets Extension (IBMJSSE) or the IBM Java Secure Sockets Extension 2 (IBMJSSE2). The IBMJSSE and IBMJSSE2 providers contain a reference implementation supporting SSL and Transport Layer Security (TLS) protocols and an application programming interface (API) framework.

The Sun JDK does not ship the IBM JSSE and IBM JSSE2 providers, therefore transport security cannot be enabled with a Sun JDK. In order to make this work, a Sun JDK shipped with WebSphere Application Server is required. The WebSphere Application Server shipped Sun JDK contains the IBM JSSE and IBM JSSE2 providers.

See “Configuring a custom Object Request Broker” on page 241 for information about using a non-IBM JDK for WebSphere eXtreme Scale. If `-Djava.endorsed.dirs` is configured, it points to both the `objectgridRoot/lib/endorsed` and the `JRE/lib/endorsed` directories. The directory `objectgridRoot/lib/endorsed` is required so the IBM ORB is used, and the directory `JRE/lib/endorsed` is required to load the IBM JSSE and IBM JSSE2 providers.

Work with step 4 of the security tutorial in the *Product Overview* for information about how to configure your required SSL properties, to create keystores and truststores, and to start secure servers in WebSphere eXtreme Scale.

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” on page 427 for more information.

Table 30. 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” on page 245.
2. To set the transportType property in the container and catalog server security configuration, see “Server properties file” on page 199.

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 not 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 new 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” on page 202.

Java Management Extensions (JMX) security

You can secure managed beans (MBean) invocations in a distributed environment.

For more information on the MBeans available, see “Administering programmatically with Managed Beans (MBeans)” on page 374.

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 utilizing 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 utilize the ObjectGrid Authenticator plug-in to authenticate the remote clients. See the security tutorial in the *Product Overview* 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", "xxxxx"));

// Create the JMXConnectorServer
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 given 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 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, but with restriction 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 utilizing TLS/SSL. If the `transportType` of catalog server or container server is set to `SSL_Required` or `SSL_Supported`, then you have to 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 using `-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, 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 enable JMX secure transport for the container server, you must set the JMX service port. Setting the JMX service port is required when you are using Transport Layer Security/Secure Sockets Layer (TLS/SSL) and you want to display container server information from the catalog server. For example, the port is required when you are using the **-mapsizes** option in the **xsadmin** tool. 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.

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.

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.

Restriction: Do not use a Development Kit or Runtime Environment at Version 1.6 and above when you are also using SSL Transport Layer security with a stand-alone configuration of WebSphere eXtreme Scale Version 7.1 or 7.0. Version 1.6 and later does not support the WebSphere eXtreme Scale Version 7.1 application programming interfaces. Use Version 1.5 or earlier for configurations requiring SSL Transport security for stand-alone eXtreme Scale installations. This restriction is only applicable when you are using SSL security in a stand-alone eXtreme Scale configuration. Version 1.6 and later is supported for non-SSL transport configurations.

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.

start0gServer**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” on page 445 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” on page 245.

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 property 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” on page 327.

- 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” on page 327. 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 31. 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" on page 440.

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” on page 433. (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 “Application client authorization” on page 437. (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.

See the “Security descriptor XML file” on page 450 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:schema>
```



```
</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>
```

Chapter 10. Monitoring the deployment environment

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.

The following figure shows the general setup of statistics for WebSphere eXtreme Scale.

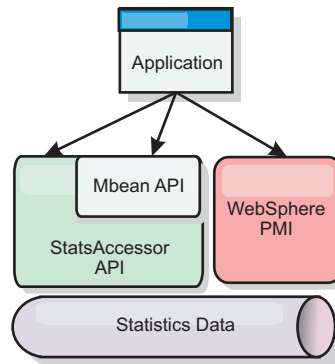


Figure 32. Statistics overview

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" on page 481 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.

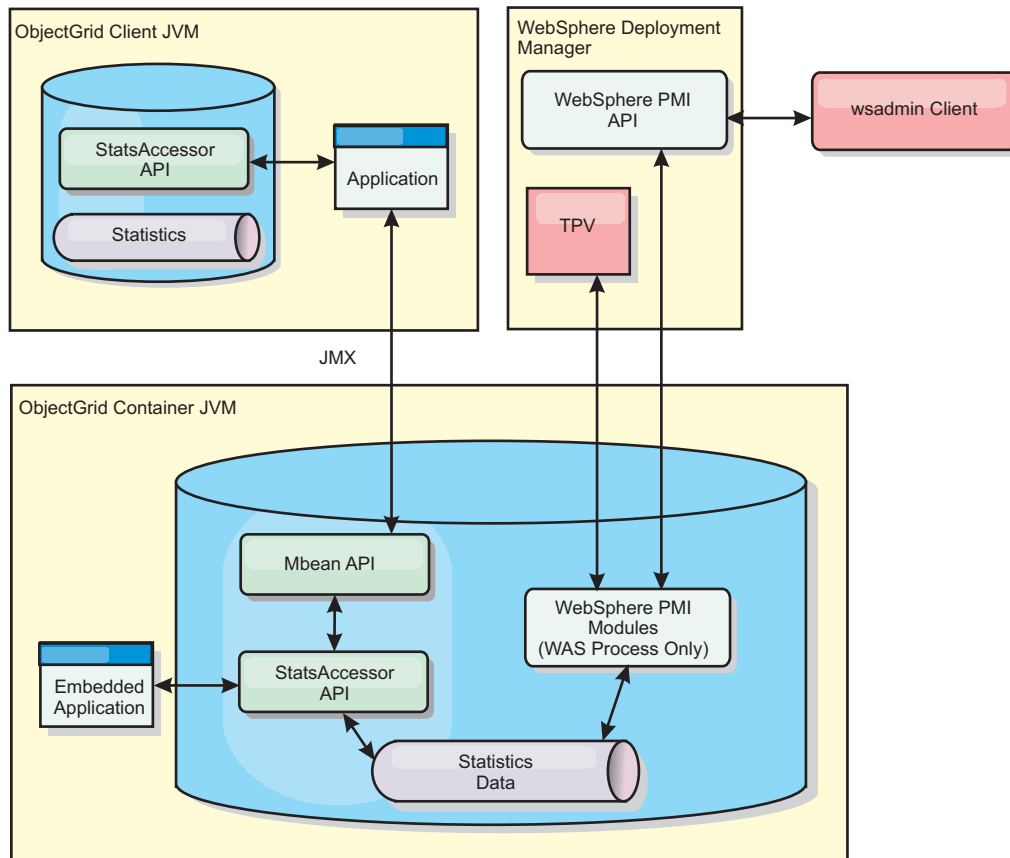


Figure 33. MBean overview

See “Monitoring with the **xsadmin** utility” on page 470 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” on page 503
- “Monitoring eXtreme Scale applications with CA Wily Introscope” on page 500

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.

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

- **System requirements**
 - Use a AIX, Linux, or Windows system to run the web console.

- Install a stand-alone WebSphere eXtreme Scale server on the system that is hosting the console server. See “Installing stand-alone WebSphere eXtreme Scale or WebSphere eXtreme Scale Client” on page 18 for more information.
- The console server system must be able to connect to your catalog service. The catalog service also must be able to connect back to the web console server.

- **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

Procedure

1. Start the console server. The **startConsoleServer.bat|sh** script for starting the console server is in the *wxs_install_root/ObjectContext/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.
 - 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 email 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.

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” on page 457 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” on page 351 for more information.

Procedure


1. If your catalog servers have Secure Sockets Layer (SSL) enabled, you must configure a keystore, truststore, and client properties file. You enable SSL for a catalog server by setting the `transportType` attribute to `SSL-Required` in the “Server properties file” on page 199.
 - 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 example, you might edit the `wxs_install_root/ObjectGridProperties/sampleclient.properties` 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}PDM20jErLvg=  
#trustStoreType=JKS  
#trustStore=etc/test/security/server.public  
#trustStorePassword={xor}Lyo9M2Y8
```

Important: Windows 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.

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.



- 1) Click the add icon () to register an existing catalog server.
 - 2) Provide information, such as the host name and listener port. See “Planning for network ports” on page 232 for more information about port configuration and defaults.
 - 3) Click **OK**.
 - 4) 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.



- 1) Click the add icon () to register a catalog service domain. Enter a name for the catalog service domain.

- 2) 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.

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.

Fix 1+

eXtreme Scale client properties path

Specifies the path to the client properties file that you edited to include SSL properties in a previous step. For example, you might indicate the c:\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.

- 3) Click **OK**.
- 4) 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.

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” on page 457 for more information.
2. Connect your catalog servers to the web console server. See “Connecting the web console to catalog servers” on page 458 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” on page 466 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.

- “Data grid domain overview”
- “Data grid overview” on page 462
- “Data grid details” on page 462
- “Server overview” on page 463
- “Custom reports: Catalog service domain statistics” on page 463
 - “Custom reports: Container server statistics” on page 463
 - “Custom reports: Data grid statistics” on page 465
 - “Custom reports: Map statistics” on page 465

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** chat 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.

Grid 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 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.

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” on page 457 for more information.
2. Connect your catalog servers to the web console server. See “Connecting the web console to catalog servers” on page 458 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.

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();

}

```

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. You can use several methods to retrieve the information from the statistics modules.

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 `com.ibm.websphere.objectgrid.stats` package in the API documentation.

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`.

Monitoring with the `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.

`xsadmin` Before you begin

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” on page 351 for more information.

About this task

The `xsadmin` sample utility uses an implementation of Managed Beans (MBeans). This sample monitoring application that enables out-of-the box 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 out 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. On the command line, set the *JAVA_HOME* environment variable.

- **UNIX** export JAVA_HOME=javaHome
- **Windows** set JAVA_HOME=javaHome

2. Go to the bin directory.

```
cd wxs_home/bin
```

3. Run the **xsadmin** utility.

- **To display the online help, run the following command:**

```
UNIX  
xsadmin.sh
```

```
Windows  
xsadmin.bat
```

Take note on the required arguments section of the help message, because you must pass in only one of the listed options for the utility to work. If no **-g** or **-m** option is specified, the **xsadmin** utility prints out information for every grid in the topology.

- **To enable statistics for all of the servers, run the following command:**

```
UNIX  
xsadmin.sh -g ObjectGridA -setstatsspec ALL=enabled
```

```
Windows  
xsadmin.bat -g ObjectGridA -setstatsspec ALL=enabled
```

- **To display all online containers for a grid, run the following command:**

```
UNIX  
xsadmin.sh -g ObjectGridA -m MapSetA -containers
```

```
Windows  
xsadmin.bat -g ObjectGridA -m MapSetA -containers
```

All container information is displayed. An example of the output follows:
Connecting to Catalog service at localhost:1099

```
*** Show all online containers for grid - ObjectGridA & mapset - MapSetA
```

```
Host: 192.168.0.186  
Container: server1_C-0, Server:server1, Zone:DefaultZone  
Partition Shard Type  
          0 Primary
```

```
Num containers matching = 1  
Total known containers = 1  
Total known hosts = 1
```

Attention: To obtain this information when Transport Layer Security/Secure Sockets Layer (TLS/SSL) is enabled, you must start the catalog and container servers with the JMX service port set. To set the JMX service port, you can either use the **-JMXServicePort** option on the **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 standalone version of WebSphere eXtreme Scale cannot be used to connect to a catalog service hosted by a WebSphere Application Server

process. Use the **xsadmin** that is script included in the *was_root/bin* directory, which is available when the installing WebSphere eXtreme Scale on WebSphere Application Server or WebSphere Application Server Network Deployment.

- a. Navigate to the WebSphere Application Server bin directory:

```
cd was_root/bin
```

- b. Launch the **xsadmin** utility using the following command:

UNIX

```
xsadmin.sh -g ObjectGridA -m MapSetA -mapsizes -fm MapA -dmgr
```

Windows

```
xsadmin.bat -g ObjectGridA -m MapSetA -mapsizes -fm MapA -dmgr
```

The size of the specified map is displayed.

Connecting to Catalog service at localhost:9809

```
****Displaying Results for Grid - ObjectGridA, MapSet - MapSetA****
```

```
*** Listing Maps for server1 ***
```

```
Map Name: MapA Partition #: 0 Map Size: 0 Shard Type: Primary  
Server Total: 0
```

- **7.1.0.2+** To display the configured and runtime placement of your configuration, run one of the following commands:

```
xsadmin -placementStatus  
xsadmin -placementStatus -g myOG -m myMapSet  
xsadmin -placementStatus -m myMapSet  
xsadmin -placementStatus -g myOG
```

You can scope the command to display placement information for the entire configuration, a single data grid, a single map set, or a combination of a data grid and map set. An example of the output follows:

```
*****Printing Placement Status for Grid - Grid, MapSet - mapSet*****
```

```
<objectGrid name="Grid" mapSetName="mapSet">  
<configuration>  
  <attribute name="placementStrategy" value="FIXED_PARTITIONS"/>  
  <attribute name="numInitialContainers" value="3"/>  
  <attribute name="minSyncReplicas" value="0"/>  
  <attribute name="developmentMode" value="true"/>  
</configuration>  
<runtime>  
  <attribute name="numContainers" value="3"/>  
  <attribute name="numMachines" value="1"/>  
  <attribute name="numOutstandingWorkItems" value="0"/>  
</runtime>  
</objectGrid>
```

Creating a configuration profile for the xsadmin utility

You can save your frequently specified parameters for the **xsadmin** utility in a properties file. As a result, the **xsadmin** utility calls are shorter.

Before you begin

Fix 1+ You can define a properties file for the **xsadmin** utility with Version 7.1 Fix 1 or later.

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” on page 356.

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 xsadmin arguments

Fix 1+ You can define a properties file for the **xsadmin** utility with Version 7.1 Fix 1 or later. By creating a properties file, you can save some of the frequently used arguments, such as the user name. The properties that you can add to a properties file are in the following table. If you specify both a property in a properties file and the equivalent command-line argument, the command-line argument value overrides the properties file value.

For more information about defining a properties file for the **xsadmin** utility, see “Creating a configuration profile for the **xsadmin** utility” on page 473.

Table 32. Arguments for the **xsadmin** utility

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

Table 32. Arguments for the `xsadmin` utility (continued)

Command Line Argument	Equivalent Property Name in Properties File	Description and valid values
-containers	n/a	For each data grid and map set, displays a list of container servers. Allows the following filters: -fnp
-continuous	n/a	Specify this flag if you want continuous map size results to monitor the data grid. When you run this command with the -mapsizes argument, the map size is displayed every 20 seconds.
-coregroups	n/a	Displays all core groups for the catalog server. This argument is used for advanced diagnostics.
-dismissLink <catalog_service_domain>	n/a	Removes a link between 2 catalog service domains. Provide the name of the foreign catalog service domain to which you previously connected with the -establishLink argument.
-dmgr	n/a	Indicates if you are connecting to a WebSphere Application Server hosted catalog service. Default: false
-empties	n/a	Specify this flag if you want to show empty containers in the output.
-establishLink <foreign_domain_name> <host1:port1,host2:port2...>	n/a	Connects the catalog service domain to a foreign catalog service domain. Use the following format: -establishLink <foreign_domain_name> <host1:port1,host2:port2...> . <i>foreign_domain_name</i> is the name of the foreign catalog service domain, and <i>host1:port1,host2:port2...</i> is a comma-separated list of catalog server host names and Object Request Broker (ORB) ports that are running in this catalog service domain.
-fc	n/a	Filters for only this container. Use with the following arguments: -mapsizes, -teardown,-revisions,-getTraceSpec,-setTraceSpec,-getStatsSpec,-setStatsSpec
-fh	n/a	Filters for only this host. Use with the following arguments: -mapsizes, -teardown,-revisions,-getTraceSpec,-setTraceSpec,-getStatsSpec,-setStatsSpec,-routetable
-fm	n/a	Filters only for this map. Use with the following arguments: -clear, -mapsizes
-fnp	n/a	Filters servers that have no primary shards. Use with the following arguments: -containers
-fp	n/a	Filters for only this partition. Use with the following arguments: -mapsizes, -teardown,-revisions,-getTraceSpec,-setTraceSpec,-getStatsSpec,-setStatsSpec,-routetable
-fs	n/a	Filters for only this server. Use with the following arguments: -mapsizes, -teardown,-revisions,-getTraceSpec,-setTraceSpec,-getStatsSpec,-setStatsSpec

Table 32. Arguments for the `xsadmin` utility (continued)

Command Line Argument	Equivalent Property Name in Properties File	Description and valid values
-fst	n/a	Filters for only this shard type. Specify P for primary shards only, A for asynchronous replica shards only, and S for synchronous replica shards only. Use with the following arguments: -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 <code>xsadmin</code> utility, which includes a list of arguments.
-hosts	n/a	Displays all of the hosts in the configuration.
-jmxUrl	XSADMIN_JMX_URL	Specifies the address of a JMX API connector server in the following format: <code>service:jmx:protocol:sap</code> . The <code>protocol</code> and <code>sap</code> variable definitions follow: <i>protocol</i> Specifies the transport protocol to be used to connect to the connector server. <i>sap</i> Specifies the address at which the connector server is found. For more information about the format of the JMX service URL, see Class <code>JMXServiceURL</code> (Java 2 Platform SE 5.0).
-l	n/a	Displays all known data grids and map sets.
-m	n/a	Specifies the name of the map set.
-mapsizes	n/a	Displays the size of each map on the catalog server to verify that key distribution is uniform over the shards. Allows the following filters: -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 <code>xsadmin</code> utility. Do not specify the password in your properties file if you want your password to remain secure.
-p	n/a	Indicates the JMX port for the catalog server host. Default: 1099 or 9809 for a WebSphere Application Server host, 1099 for stand-alone configurations.

Table 32. Arguments for the `xsadmin` utility (continued)

Command Line Argument	Equivalent Property Name in Properties File	Description and valid values
7.1.0.2+ -placementStatus	n/a	Displays the configured placement and runtime placement of your configuration. You can scope the output to a combination of data grids and map sets, or for the entire configuration: <ul style="list-style-type: none"> Entire configuration: -placementStatus For a specific data grid: -placementStatus -g <i>my_grid</i> For a specific map set: -placementStatus -m <i>my_mapset</i> For a specific data grid and map set: -placementStatus -g <i>my_grid</i> -m <i>my_mapset</i>
-primaries	n/a	Displays a list of the primary shards.
-profile	n/a	Specifies a fully qualified path to the properties file for the <code>xsadmin</code> utility.
-quorumstatus	n/a	Displays the status of quorum for the catalog service.
-releaseShard <container_server_name> <objectgrid_name> <map_set_name> <partition_name>	n/a	Used in conjunction with the <code>-reserveShard</code> argument. The <code>-releaseShard</code> argument must be invoked after a shard has been reserved and placed. . The <code>-releaseShard</code> argument invokes the <code>ContainerMBean.release()</code> method.
-reserved	n/a	Used with the <code>-containers</code> argument to display only shards that have been reserved with the <code>-reserveShard</code> argument.
-reserveShard <container_server_name> <objectgrid_name> <map_set_name> <partition_name>	n/a	Moves a primary shard to the specified container server. The <code>ContainerMBean.reserve()</code> method is invoked by this argument.
7.1.0.3+ -resumeBalancing <objectgrid_name> <map_set_name>	n/a	Attempts to balance requests and allow future rebalancing attempts to 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 his argument to determine if an asynchronous replica or linked domain is caught up. This argument invokes the <code>ObjectGridMBean.getKnownRevisions()</code> method. Allows the following filters: -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 <code>TargetNotAvailable</code> exceptions. Allows the following filters: -fz -fh -fp

Table 32. Arguments for the xsadmin utility (continued)

Command Line Argument	Equivalent Property Name in Properties File	Description and valid values
-settracespec <trace_string>	n/a	Enables trace on servers during run time. See the following example: -setTraceSpec "ObjectGridReplication=all=enabled" See "Collecting trace" on page 530 and "Trace options" on page 531 for more information about the trace strings that you can specify. Allows the following filters: -fst -fc -fz -fs -fh -fp
7.1.0.3+ -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. See Class StatsSpec for more information about the statistics modules you can monitor. Allows the following filters: -fm -fst -fc -fz -fs -fh -fp
7.1.0.3+ -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.
-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.

Table 32. Arguments for the `xsadmin` utility (continued)

Command Line Argument	Equivalent Property Name in Properties File	Description and valid values
-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 <code>xsadmin</code> 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 <code>xsadmin</code> utility” for more information.
-xml	n/a	Prints the unfiltered output from the <code>PlacementServiceMBean.listObjectGridPlacement()</code> method. The other <code>xsadmin</code> arguments filter the output of this method and organize the data into a more consumable format.

Verbose option for the `xsadmin` utility

You can use the `xsadmin` verbose option to troubleshoot problems. Run the `xsadmin -v` command to list all configured parameters. The verbose option displays all values in all scopes, including command line arguments, properties file arguments, and environment-specified arguments. The Effective arguments section includes the settings that are being used in the environment if you have specified the same property using multiple scopes.

`xsadmin` Verbose option example

Fix 1+ You use the verbose option for the `xsadmin` utility with Version 7.1 Fix 1 or later.

`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

```

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” on page 483 for more information.

What to do next

For more information about the wsadmin tool, see “Accessing Managed Beans (MBeans) using the wsadmin tool” on page 375.

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 scripting, you also must be able to log in and use the wsadmin tool. For more information about the wsadmin tool, see the wsadmin tool topic in the WebSphere Application Server information center.

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” on page 481 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.

- Administrative console steps
- Scripting steps

For more information about the statistics that can be retrieved, see “PMI modules” on page 485.

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.

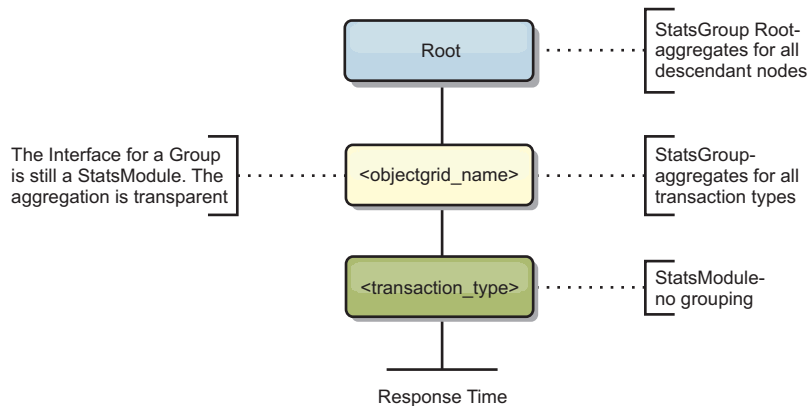


Figure 34. ObjectGridModule module structure

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.

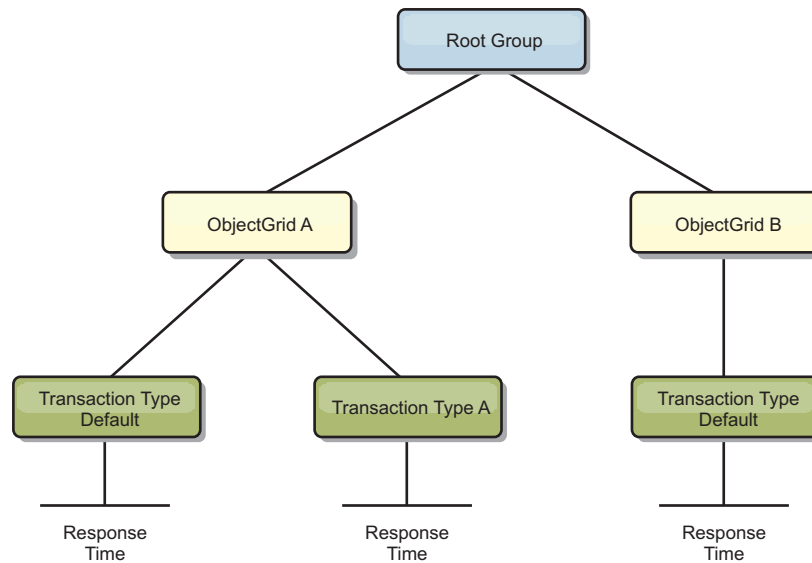


Figure 35. ObjectGridModule module structure example

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:

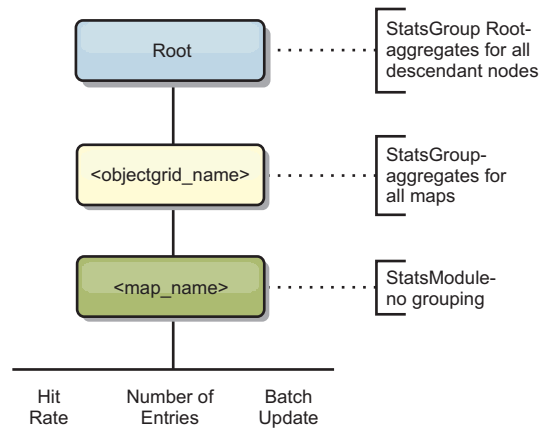
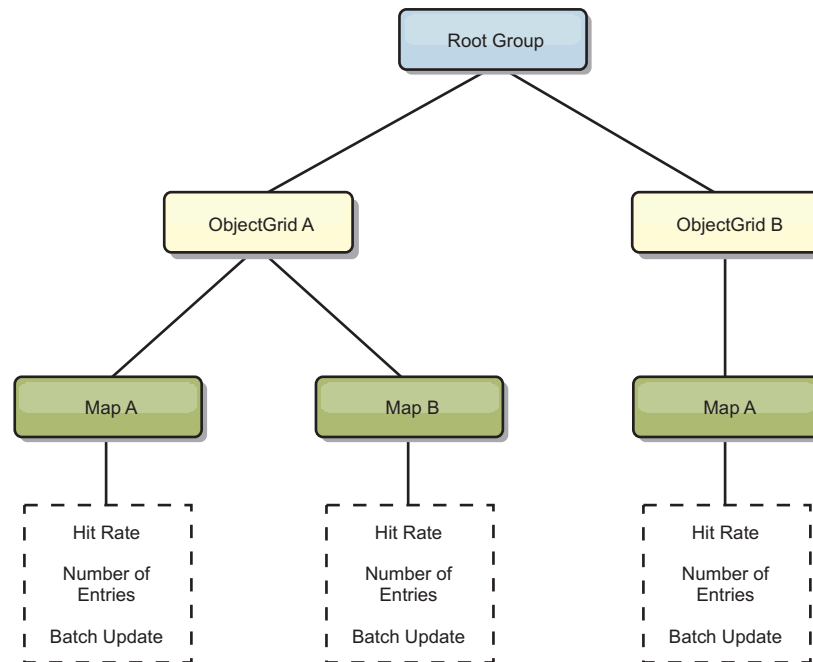


Figure 36. mapModule structure

The following diagram shows an example of the mapModule structure:

Figure 37. 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 ObjectGrids as its child elements, ObjectGrids have maps as their child elements, which finally have HashIndexes as their child elements and leaf nodes of the tree. Every HashIndex instance has the three listed statistics. The hashIndexModule structure is shown in the following diagram:

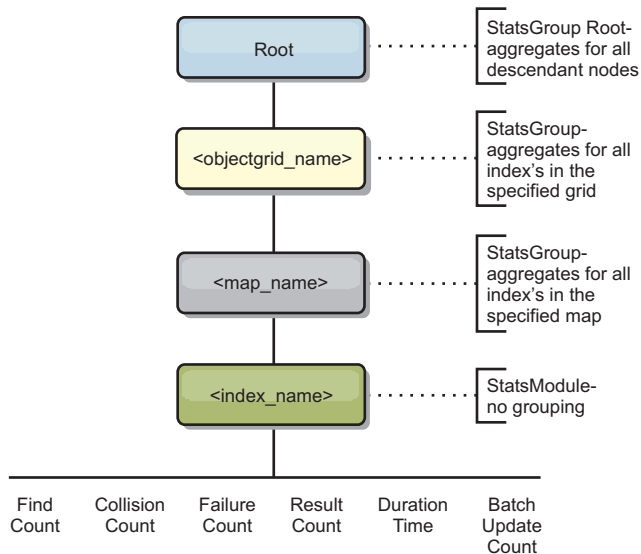


Figure 38. hashIndexModule module structure

The following diagram shows an example of the hashIndexModule structure:

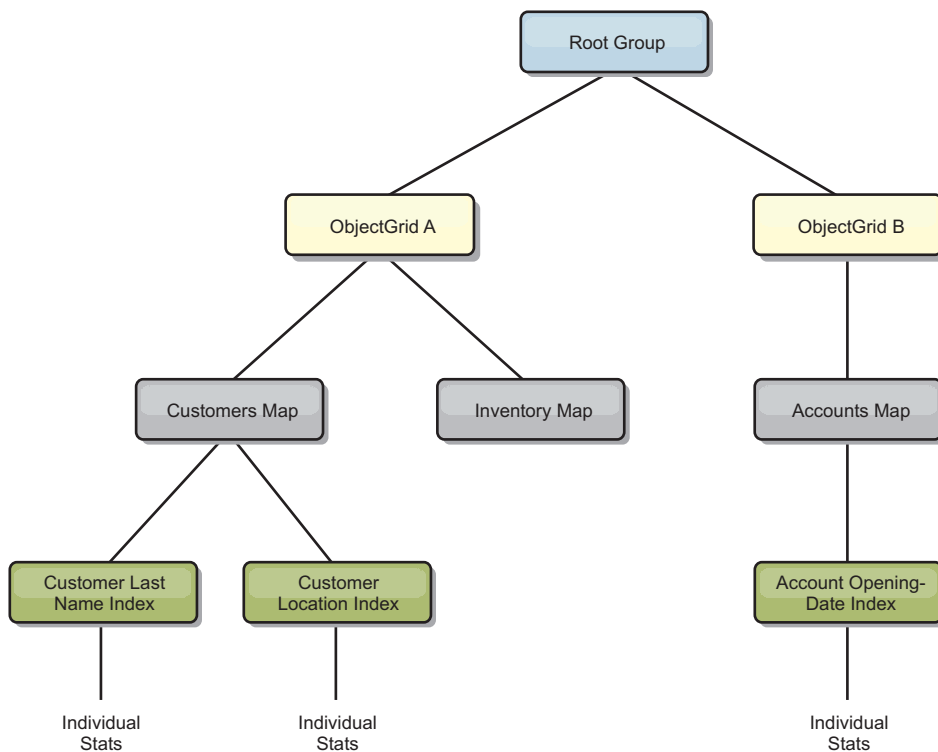


Figure 39. hashIndexModule module structure example

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.

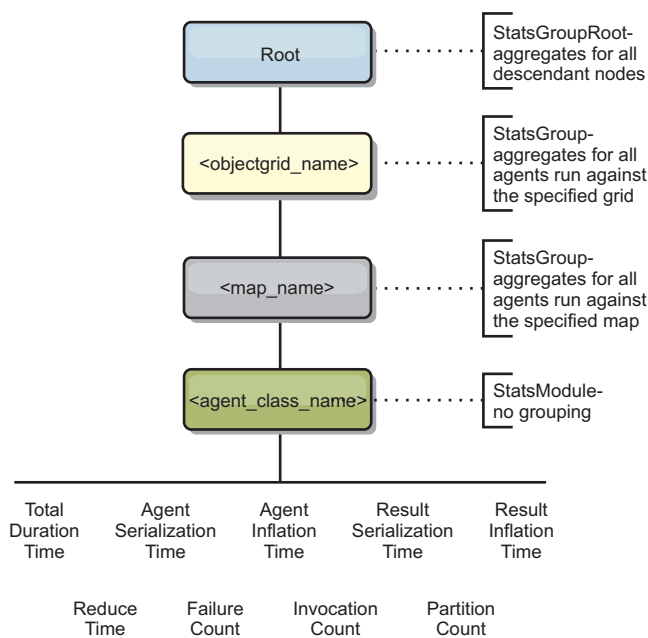


Figure 40. agentManagerModule structure

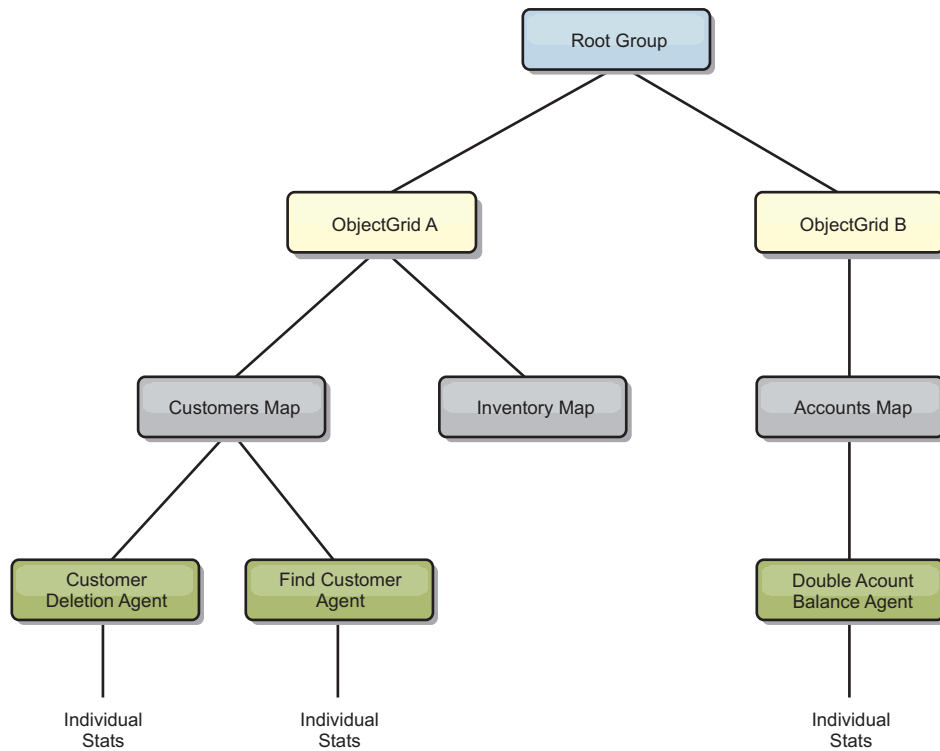


Figure 41. agentManagerModule structure example

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.

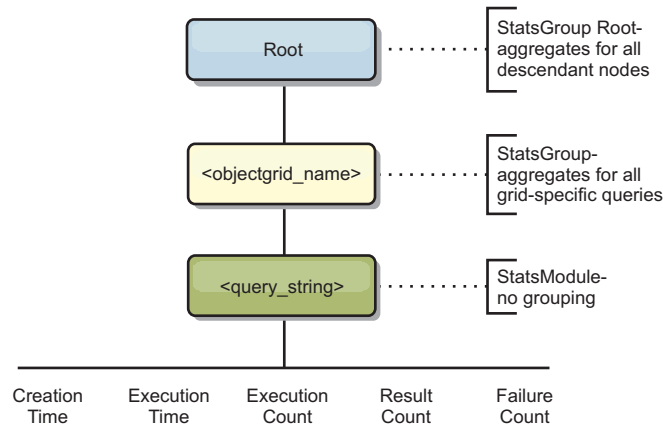


Figure 42. queryModule structure

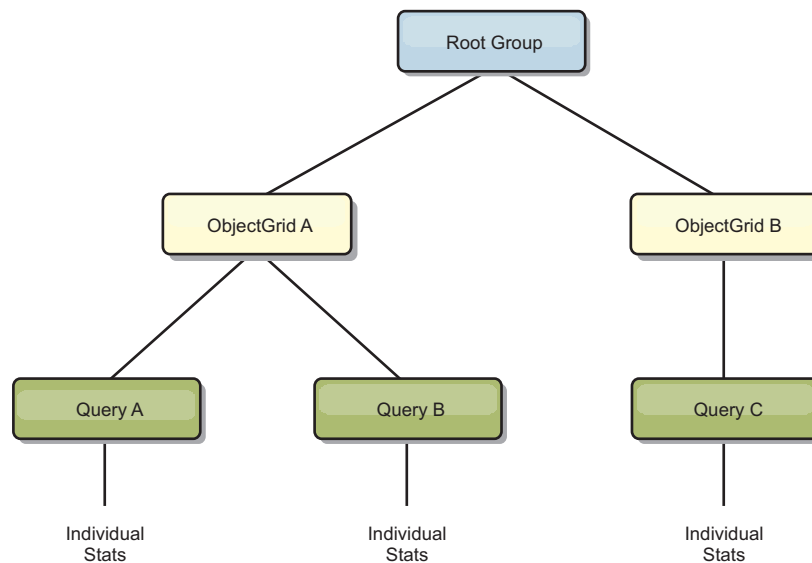


Figure 43. QueryStats.jpg queryModule structure example

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>

```

Monitoring with managed beans (MBeans)

You can use 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 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 in the API documentation.

For more information about the server properties file, see “Server properties file” on page 199.

- **With a managed bean:**

You can enable statistics using the `StatsSpec` attribute on the ObjectGrid MBean. For more information, see the `StatsSpec` API in the API documentation.

- **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” on page 375.

- **Access MBean statistics programmatically.**

For more information, see “Accessing Managed Beans (MBeans) programmatically” on page 375.

Example

For an example of how to use managed beans, see “Monitoring with the `xsadmin` utility” on page 470.

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 IBM Tivoli Enterprise Monitoring Agent for WebSphere eXtreme Scale

The IBM Tivoli Enterprise Monitoring Agent 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 both stand-alone eXtreme Scale and WebSphere Application Server deployments.

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 with managed beans (MBeans)” on page 493 and “Monitoring with the `xsadmin` utility” on page 470
- 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 WebSphere eXtreme Scale agent, which you can download for free from the IBM Open Process Automation Library (OPAL) site.

Complete the following steps to install and configure the Tivoli Monitoring Agent:

Procedure

1. Install the Tivoli Monitoring Agent for WebSphere eXtreme Scale.
Download the Tivoli installation image and extract its files to a temporary directory.
2. Install eXtreme Scale application support files.
Install eXtreme Scale application support on each of the following deployments.
 - Tivoli Enterprise Portal Server (TEPS)
 - 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 the eXtreme Scale agent 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 hostname and administrative credentials. 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 required by WebSphere eXtreme Scale agent 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 **Location for 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 WebSphere eXtreme Scale agent.

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 default**, 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:
 - a. Update the Java parameters, ensure that the **Java Home** value is correct. JVM arguments can be left empty. Click **Next**.
 - b. Select the type of **MBean server connection type**, Use JSR-160-Complaint Server for stand-alone eXtreme Scale servers. Click **Next**.

- c. 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:

- a. 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.
 - b. If multiple eXtreme Scale container servers exist on your computer, configure the agent to monitor each one server.
 - c. 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.)
 - d. After you configure the container servers, click **Next**, which brings you to the **WebSphere eXtreme Scale Catalog Servers** pane.
 - e. 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:
 - a. Update the Java parameters, ensure that the **Java Home** value is correct. JVM arguments can be left empty. Click **Next**.
 - b. Select the **MBean server connection type**. Select the WebSphere Application Server version that is appropriate for your environment. Click **Next**.
 - c. Ensure that the WebSphere Application Server information in the panel is correct. Click **Next**.
 - d. 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**.
 - e. 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://localhost:
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 # ./itcmd 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 eXtreme Scale agent 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 Scale container 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 `start0gServer.sh | .bat` 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.

- 1) `cd <ITM_install>/bin`
- 2) `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.

- 1) `cd <ITM_install>/bin`
- 2) `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.

PBD file settings for monitoring the catalog service

You can use one or more of the following settings in your PBD file to monitor the catalog service.

```
TraceOneMethodOfClass: com.ibm.ws.objectgrid.hamanager.HAControllerImpl
  changeDefinedCompleted
BlamePointTracerDifferentMethods "OGcatalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.hamanager.HAControllerImpl
  viewChangeCompleted
BlamePointTracerDifferentMethods "OGcatalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.hamanager.HAControllerImpl
  viewAboutToChange
BlamePointTracerDifferentMethods "OGcatalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent
  heartbeat
BlamePointTracerDifferentMethods "OGcatalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent
  heartbeatCluster
BlamePointTracerDifferentMethods "OGcatalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent
  heartbeatCurrentLeader
BlamePointTracerDifferentMethods "OGcatalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent
  heartbeatDeadServer
BlamePointTracerDifferentMethods "OGcatalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent
  heartbeatNewLeader
BlamePointTracerDifferentMethods "OGcatalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent
  heartbeatNewServer
BlamePointTracerDifferentMethods "OGcatalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.catalog.placement.
  PlacementServiceImpl
importRouteInfo BlamePointTracerDifferentMethods
  "OGcatalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.catalog.placement.
```

```

PlacementServiceImpl heartbeat
BlamePointTracerDifferentMethods "OGcatalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.catalog.placement.
PlacementServiceImpl joinPlacementGroup
BlamePointTracerDifferentMethods "OGcatalog|{classname}|{method}"
TraceOneMethodOfClass:
com.ibm.ws.objectgrid.catalog.placement.PlacementServiceImpl
classifyServer
BlamePointTracerDifferentMethods "OGcatalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.catalog.placement.
BalanceGridEventListener shardActivated
BlamePointTracerDifferentMethods "OGcatalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.catalog.placement.
BalanceGridEventListener shardDeactivate
BlamePointTracerDifferentMethods "OGcatalog|{classname}|{method}"

```

Classes for monitoring the catalog service

HAControllerImpl

The HAControllerImpl class handles core group life cycle and feedback events. You can monitor this class to get an indication of the core group structure and changes.

ServerAgent

The ServerAgent class is responsible for communicating core group events with the catalog service. You can monitor the various heartbeat calls to spot major events.

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 currently acting as the leader.

PBD file settings for monitoring the containers

You can use one or more of the following settings in your PBD file to monitor the containers.

```

TraceOneMethodOfClass: com.ibm.ws.objectgrid.ShardImpl processMessage
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.plugins.
CommittedLogSequenceListenerProxy applyCommitted
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.plugins.
CommittedLogSequenceListenerProxy sendApplyCommitted
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.map.BaseMap evictMapEntries
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.checkpoint.
CheckpointMapImpl$CheckpointIterator activateListener
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.hamanager.HAControllerImpl
changeDefinedCompleted
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.hamanager.HAControllerImpl
viewChangeCompleted
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.hamanager.HAControllerImpl
viewAboutToChange
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent
batchProcess

```



```

BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent
  heartbeat
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent
  heartbeatCluster
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent
  heartbeatCurrentLeader
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent
  heartbeatDeadServer
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent
  heartbeatNewLeader
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent
  heartbeatNewServer
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"

```

Classes for monitoring the containers

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.

CheckpointIterator

The `CheckpointIterator` class has the `activateListener` method call which puts primaries into peer mode. When the primaries are put into peer mode, the replica is up to date with the primary after the method completes. When a replica is regenerating from a full primary, this operation can take an extended period of time. The system is not fully recovered until this operation completes, so you can use this class to monitor the progress of the operation.

CommittedLogSequenceListenerProxy

The `CommittedLogSequenceListenerProxy` class has two methods of interest. The `applyCommitted` method runs for every transaction and the `sendApplyCommitted` runs as the replica is pulling information. The ratio of how often these two methods run can give you some indication of how well the replica is able to keep up with the primary.

PBD file settings for monitoring the clients

You can use one or more of the following settings in your PBD file to monitor the clients.

```

TraceOneMethodOfClass: com.ibm.ws.objectgrid.client.ORBClientCoreMessageHandler
  sendMessage
BlamePointTracerDifferentMethods "OGclient|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.corba.cluster.ClusterStore
  bootstrap
BlamePointTracerDifferentMethods "OGclient|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.corba.cluster.ClusterStore
  epochChangeBootstrap
BlamePointTracerDifferentMethods "OGclient|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.map.BaseMap evictMapEntries
BlamePointTracerDifferentMethods "OGclient|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.cluster.orb.routing.
  SelectionServiceImpl routeFailed
BlamePointTracerDifferentMethods "OGclient|{classname}|{method}"

```



```

TraceOneMethodOfClass: com.ibm.ws.objectgrid.cluster.orb.routing.
  SelectionServiceImpl routeFailed
BlamePointTracerDifferentMethods "OGclient|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.SessionImpl getMap
BlamePointTracerDifferentMethods "OGclient|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.ObjectGridImpl getSession
BlamePointTracerDifferentMethods "OGclient|{classname}|{method}"
TurnOn: ObjectMap
SetFlag: ObjectMap
IdentifyClassAs: com.ibm.ws.objectgrid.ObjectMapImpl ObjectMap
TraceComplexMethodsifFlagged: ObjectMap BlamePointTracerDifferentMethods
"OGclient|{classname}|{method}"

```

Classes for monitoring the clients

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.

ObjectGridImpl

The ObjectGridImpl class has the getSession method that you can monitor to see the number of requests to this 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 “**startOgServer** script” on page 356.
2. Put the `extremescale-plugin.xml` file and the `wshyperic.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” on page 199, which enable statistics as soon as the containers start. Various options for enabling statistics are described in “Monitoring with managed beans (MBeans)” on page 493 and “Monitoring with the **xsadmin** utility” on page 470
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” on page 530 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

Chapter 11. Tuning and performance

You can tune settings in your environment to increase the overall performance of your WebSphere eXtreme Scale installation.

Operational checklist

Use the operational checklist to prepare your environment for deploying WebSphere eXtreme Scale.

Table 33. Operational checklist

Checklist item	For more information
<p>If you are using AIX, tune the following operating system settings:</p> <p>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:</p> <pre># no -o tcp_keepintvl</pre> <p>To change the current setting, run the following command:</p> <pre># no -o tcp_keepintvl=10</pre> <p>The TCP_KEEPINTVL setting is in half seconds.</p> <p>TCP_KEEPIINIT The TCP_KEEPIINIT 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:</p> <pre># no -o tcp_keeppinit</pre> <p>To change the current setting, run the following command:</p> <pre># no -o tcp_keeppinit=40</pre> <p>The TCP_KEEPIINIT setting is in half seconds.</p>	<ul style="list-style-type: none">For AIX tuning information, see <i>Tuning AIX systems</i>.
<p>Update the <code>orb.properties</code> file to modify the transport behavior of the grid. The <code>orb.properties</code> file is in the <code>java/jre/lib</code> directory.</p>	<p>"ORB properties" on page 237</p>

Table 33. Operational checklist (continued)

Checklist item	For more information
<p>Use parameters in the startOgServer script. In particular, use the following parameters:</p> <ul style="list-style-type: none"> • Set heap settings with the -jvmArgs parameter. • Set application class path and properties with the -jvmArgs parameter. • Set -jvmArgs parameters for configuring agent monitoring. <p>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:</p> <p>Listener port You can use the -listenerPort argument to specify the port that is used for communication between processes.</p> <p>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.</p> <p>JMX service port You can use the -JMXServicePort argument to specify the port that the JMX service should use.</p> <p>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.</p> <p>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</p>	<p>“startOgServer script” on page 356</p>
<p>Verify that security settings are configured correctly:</p> <ul style="list-style-type: none"> • Transport (SSL) • Application (Authentication and Authorization) <p>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.</p>	<p>“Security integration with external providers” on page 445</p>
<p>Choose how you are going to monitor your environment.</p> <ul style="list-style-type: none"> • xsAdmin tool: <ul style="list-style-type: none"> – The JMX ports of the catalog servers need to be visible to the xsAdmin tool. The container server ports also need to be accessible for some commands that gather information from the containers. • 7.1+ Monitoring console: With the monitoring console, you can chart current and historical statistics. • Vendor monitoring tools: <ul style="list-style-type: none"> – Tivoli Enterprise Monitoring Agent – CA Wily Introscope – Hyperic HQ 	<ul style="list-style-type: none"> • “Monitoring with the xsadmin utility” on page 470 • “Java Management Extensions (JMX) security” on page 443 • 7.1+ “Monitoring with the web console” on page 457 • “Monitoring with the IBM Tivoli Enterprise Monitoring Agent for WebSphere eXtreme Scale” on page 494 • “Monitoring eXtreme Scale with Hyperic HQ” on page 503 • “Monitoring eXtreme Scale applications with CA Wily Introscope” on page 500

Operating systems and network tuning

Network tuning can reduce Transmission Control Protocol (TCP) stack delay by changing connection settings and can improve throughput by changing TCP buffers.

Operating systems

A Windows system needs the least tuning while a Solaris system needs the most tuning. The following information pertains to each system specified, and might improve WebSphere eXtreme Scale performance. You should 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

```
nnd -set /dev/tcp tcp_time_wait_interval 60000
fnnd -set /dev/tcp tcp_keepalive_interval 15000
nnd -set /dev/tcp tcp_fin_wait_2_flush_interval 67500
nnd -set /dev/tcp tcp_conn_req_max_q 16384
nnd -set /dev/tcp tcp_conn_req_max_q0 16384
nnd -set /dev/tcp tcp_xmit_hiwat 400000
nnd -set /dev/tcp tcp_recv_hiwat 400000
nnd -set /dev/tcp tcp_cwnd_max 2097152
nnd -set /dev/tcp tcp_ip_abort_interval 20000
nnd -set /dev/tcp tcp_rexmit_interval_initial 4000
nnd -set /dev/tcp tcp_rexmit_interval_max 10000
nnd -set /dev/tcp tcp_rexmit_interval_min 3000
nnd -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

```
nnd -set /dev/tcp tcp_ip_abort_cinterval 20000
```

Planning for network ports

WebSphere eXtreme Scale is a distributed cache that requires opening ports to communicate with the Object Request Broker (ORB) and Transmission Control Protocol (TCP) stack among Java virtual machines. Plan and control your ports, especially in an environment that has a firewall, and when you are using a catalog service and containers on multiple ports.

Important: When you are specifying port numbers, avoid setting ports that are in the ephemeral range for your operating system. If you use a port that is in the ephemeral range, port conflicts could 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 for catalog servers to access catalog service data. In WebSphere Application Server, this port is set through the catalog service domain configuration.

listenerPort

Defines the ORB listener port for containers and clients to communicate with the catalog service through the ORB. In WebSphere Application Server, the listenerPort is inherited by the BOOTSTRAP_ADDRESS port configuration.

Default: 2809

Container servers

The WebSphere eXtreme Scale container servers also require several ports to operate. By default, the eXtreme Scale container server generates its HA manager port and ORB listener port automatically with dynamic ports. 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 peer port. (Required for WebSphere Application Server environments only.)

listenerPort

Defines the ORB listener port for containers and clients to communicate with the catalog service through the ORB.

Default: 2809

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 several other commands. Use the **listenerPort** property in the client properties file to specify the port in which the client listens for callbacks from the server.

haManagerPort

Specifies the peer port. (Required for WebSphere Application Server environments only.)

jvmArgs (optional)

Specifies a list of Java virtual machine (JVM) arguments. When security is enabled, you must use the following argument to configure the Secure Socket Layer (SSL) port: `-jvmArgs Dcom.ibm.CSI.SSLPort=<sslPort>`

listenerPort

Defines the ORB listener port for containers and clients to communicate with the catalog service through the ORB.

Default: 2809

Ports in WebSphere Application Server

- The **listenerPort** value is inherited from the **BOOTSTRAP_ADDRESS** value for each WebSphere Application Server application server.
- The **haManagerPort** and **peerPort** values inherited from the **DCS_UNICAST_ADDRESS** value for each WebSphere Application Server application server.

You can define a catalog service domain in the administrative console as described in “Creating catalog service domains in WebSphere Application Server” on page 206.

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: **Servers > Server Types > WebSphere Application Servers > *server_name* > Ports > *port_name***

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 -1 (unlimited) with the command: `ulimit -n -1`.

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
```

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.

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 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 using WebSphere eXtreme Scale security, set the `com.ibm.CORBA.NoLocalInterceptors` property value to `false`. The security infrastructure uses interceptors for authentication.

JVM tuning for WebSphere eXtreme Scale

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.

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.

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.

All modern JVMs today 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.

Attention: If you are using a Sun 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.

See “Using WebSphere Real Time” on page 519 for more information.

For more information about configuring garbage collection, see Tuning the IBM virtual machine for Java.

JVM performance

WebSphere eXtreme Scale can run on different versions of Java Platform, Standard Edition. WebSphere eXtreme Scale supports Java SE Version 1.4.2 and above. For improved developer productivity and performance, use Java SE 5 or later 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 Version 1.4.2 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.

For most of the scenarios in which WebSphere eXtreme Scale is used, Java SE Version 6 of the JVM performs better than Edition 5 or 1.4. Java Platform, Standard Edition Version 1.4 performs poorly especially for scenarios that use the **gencon** collector. 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.

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 party Java 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
```

Configuring 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.

About this task

Configuring failover varies depending 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.

You can configure heartbeat intervals on the command line by using the **-heartbeat** parameter in the **startOgServer** script file. Set this parameter to one of the following values:

Table 34. Heartbeat intervals

Value	Action	Description
0	Typical (default)	Failovers are typically detected within 30 seconds.
-1	Aggressive	Failovers are typically detected within 5 seconds.
1	Relaxed	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 6.0.2 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.x and 7.x:**

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.

Using 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.

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` parameter in the **ogStartServer** command. For example, 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` parameter to 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, optavgpause and subpool are specifically designed to solve differing application requirements and environments. For more information on these policies, see “JVM tuning for WebSphere eXtreme Scale” on page 515. 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 "Using WebSphere Real Time" on page 519. 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 peak 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 will either be null in the case of 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 will be not be compressible, consider setting this value to false.

Tuning the cache sizing agent for accurate memory consumption estimates

Beginning with Version 7.1, WebSphere eXtreme Scale supports sizing the memory consumption of BackingMaps 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.

```
CWOBJ45411: 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

Beginning with the 7.1 release, WebSphere eXtreme Scale can accurately estimate the Java heap memory usage of a given `BackingMap` in bytes. Leverage 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 Serializable,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” on page 469.

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 programmatically with Managed Beans (MBeans)” on page 374.

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” on page 485.

7.1+ WebSphere eXtreme Scale console

With the console, you can view the memory consumption statistics. See “Monitoring with the web console” on page 457.

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” on page 524 can enhance the accuracy of the size in bytes measurements provided by WebSphere eXtreme Scale.

Chapter 12. 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 xsAdmin sample utility to display textual information about your environment.

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 **startOgServer** 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 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 if 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:

CW0BJ1001I: ObjectGrid Server catalogServer01 is ready to process requests.

For more information about a specific message in the log files, see “Messages” on page 540.

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 “Trace options” on page 531 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 **startOgServer** 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” on page 356.

- **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” on page 199.

- **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
```


See WebSphere Application Server: Enabling trace on client and stand-alone applications for more information.

- **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 the information about interacting with the ObjectGrid using the ObjectGridManager interface in the *Programming Guide*.

- **Collect trace on container servers with the xsadmin utility.**

To collect trace with the **xsadmin** utility, use the **setTraceSpec** option. Use the **xsadmin** 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:

```
xsadmin.bat -setTraceSpec "ObjectGridReplication=all=enabled"
```

You can also disable trace by setting the trace specification to ***=all=disabled**. For more information about the **setTraceSpec** option, see "Monitoring with the **xsadmin** utility" on page 470.

Results

Trace files are written to the specified location.

Trace options

You can enable trace to provide information about your environment to IBM support.

About trace

WebSphere eXtreme Scale trace is divided into several different components. You can specify the level of trace to use. 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

ObjectGrid

General core cache engine.

ObjectGridCatalogServer

General catalog service.

ObjectGridChannel

Static deployment topology communications.

7.1+ ObjectGridClientInfo

DB2 client information.

7.1+ 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.

7.1+ ObjectGridMonitor

Historical monitoring infrastructure.

ObjectGridPlacement

Catalog server shard placement service.

ObjectGridQuery

ObjectGrid query.

ObjectGridReplication

Replication service.

ObjectGridRouting

Client/server routing details.

ObjectGridSecurity

Security trace.

ObjectGridStats

ObjectGrid statistics.

ObjectGridStreamQuery

The Stream Query API.

ObjectGridWriteBehind

ObjectGrid write behind.

Projector

The engine within the EntityManager API.

QueryEngine

The query engine for the Object Query API and EntityManager Query API.

QueryEnginePlan

Query plan diagnostics.

Troubleshooting installation

Use this information to troubleshoot issues with your installation.

Procedure

Problem: When you run the install 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 install script from the network drive successfully, for example, y:\mymachine\downloads\WXS\install.bat.

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.

Troubleshooting loaders

Use this information to troubleshoot issues with your database loaders.

Procedure

- **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" />
```

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 an 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.

```
CW0BJ2403E: 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:

CW0BJ2403E: 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 CW0BJ2403E appears in the log. For example, the following message is displayed when a quotation is missing on one of the XML attributes

CW0BJ2403E: 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 CW0BJ9002E:
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="collection1">
      <bean id="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 1.4.2 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.

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” on page 441 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 will initiate a connect to send the agent results. This makes the container server a client from connect point of view. Therefore, if TLS/SSL is configured, make sure the client's public certificate is imported in the server's trust store.

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/>.

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.

See the information center for an index of product messages.

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” on page 541

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.

- **7.1+** To see a list of the release notes for Version 7.1, go to the Support Web page.
- To see a list of the release notes for Version 7.0, go to the Support Web page.
- To see a list of the release notes for Version 6.1, go to the Release notes wiki 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 page.

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

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

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

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, New York 10594 USA

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

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

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

Trademarks

The following terms are trademarks of IBM Corporation in the United States, other countries, or both:

- AIX
- CICS[®]
- Cloudscape
- DB2
- Domino[®]
- IBM
- Lotus[®]
- RACF[®]
- Redbooks[®]
- Tivoli
- WebSphere
- z/OS

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

LINUX is a trademark of Linus Torvalds in the U.S., other countries, or both.

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

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

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

Index

A

- administering 351
 - WebSphere Application Server 205
- Administration API 364
- API 367
- architecture 78, 103

B

- back-end 142
- benefits 134, 137
- best practices 519
- build definition file
 - creating
 - CIP 31
 - IIP 34

C

- cache
 - local 79
- caching 133
- caching support 134, 137
- caching supportloaderloader
 - transaction 134, 137
- capacity planning 9, 105
- catalog server
 - configuring 198
 - starting 351
 - stopping 351
- catalog service
 - best practices for 226
 - catalog service domain 363
 - configuring in WebSphere Application Server 205
 - overview 95
 - starting
 - in a WebSphere Application Server Environment 351
 - in an environment that is not running WebSphere Application Server 351
 - starting in WebSphere Application Server 363
- catalog service domain 96
 - administrative tasks 207
 - creating in WebSphere Application Server 206
- client 244
- client authorization
 - creator only access 437
 - custom 437
 - JAAS 437
 - permissions
 - checking period 437
- client invalidation 252
- client properties file 245
- client-server security
 - secure sockets layer (SSL) 441
 - TCP/IP 441

- client-server security (*continued*)
 - transport layer security (TLS) 441
- clients
 - configuring 247
 - command line 56
 - configuration 453
 - configuring 115, 116, 241
 - Configuring 244
 - configuring after install 46
 - container processes
 - starting 354
 - container server 103
 - configuring 198
 - configuring in WebSphere Application Server 221
 - starting 351
 - stopping 351
 - container servers
 - configuring in WebSphere Application Server 222
 - containers 96
 - CPU sizing 11, 107, 108
 - custom properties 237, 511
 - customization definitions
 - generating 59, 65
 - customized jobs
 - running 60, 66
 - uploading 60, 66
 - customizing 57, 63

D

- data grid security
 - JSSE 434
 - token manager 434
- DB2 505
- deployment policy
 - configuring 174
 - deploymentPolicy.xsd file 197
 - descriptor XML 192
 - XML configuration 197
- distributing changes
 - peer JVMs 147
- dynamic cache
 - configuring 311

E

- entities
 - relationships 256
- entity
 - configuring 256
- entity metadata
 - emd.xsd file 263
 - XML configuration 258, 263
- event listener 150
- evictors
 - configuring 116
 - plug-in 119
 - TTL evictor 117

- extension files 58, 63
- eXtreme Scale overview 73

F

- failed updates 142
- failover
 - configuring 190, 517
- First steps console 46

G

- grid authorization 433

H

- HTTP session manager 288
 - configuring 288
 - configuring with XML 300
 - parameters for configuring 308
 - with WebSphere Virtual Enterprise 306
- HTTP sessions
 - splicer.properties file 297
- Hyperic HQ 503

I

- IBM Installation Factory
 - build definition file 30
- IBM Support Assistant 539
- IBM Tivoli Monitoring 494
- IBM Update Installer for WebSphere
 - uninstalling
 - CIP 34
- IBM Update Installer for WebSphere Software 70
- index
 - configuration 122
 - HashIndex 122
- Installation Factory
 - CIP
 - maintenance 33
- Installation Factory plug-in
 - build definition file
 - modify 36
 - installing
 - CIP 32
 - IIP 35
 - installing 241
 - customized installation package 37
 - IBM Installation Factory
 - CIP 30
 - IIP 30
 - maintenance 70
 - Network Deployment 26
 - silently 37, 55, 56
 - stand-alone 18
 - WebSphere Application Server 26

Introscope 500
invalidation 150

J

Java Authentication and Authorization Service
JAAS 430
Java EE 76
Java message service 146
Java Persistence API 268
Java Persistence API (JPA) 266
cache plug-in
configuration 269
introduction 272
cache topology
embedded 269, 272, 276, 283
embedded partitioned 269, 272, 276, 283
Hibernate 276
OpenJPA 283
remote 269, 272, 276, 283
Hibernate plug-in
configuration 276
OpenJPA plug-in
configuration 283
Java SE 75
Java virtual machine 515
JDK 75
JMS 150
JMX securityaccess control
authentication 443
JAAS support 443
secure transport 443
jobs 57, 63
JVM 515

L

loader 142
preloading 128
loader transaction 142
loaders
JPA 266
locking
configuring programmatically 126
configuring with XML 126
no 126
optimistic 126
pessimistic 126
log element 147
log sequence 147

M

maintaining 373
managed bean 493
manageprofiles command 45, 48
MBean
accessing with wsadmin 375, 492
administering with 375
overview 493
MBeans
accessing programmatically 375
accessing with security enabled 443
messages 540

migrate 69
migrating 67
monitoring
agent 494
with CA Wily Introscope 500
with DB2 505
with Hyperic HQ 503
with Introscope 500
with PMI 481
with the statistics API 467
with vendor tools 494
monitoring console
statistic descriptions 461
viewing statistics with 460
multi-master data grid replication 85

N

network 509
Network Deployment 48
network ports 232, 510

O

Object Request Broker 241
Object Request Broker (ORB)
configuring 237
configuring stand-alone 241
orb.properties file 237, 511
ObjectGrid
XML configuration 170
ObjectGrid descriptor XML 153
objectGrid.xsd file 170
objectGridSecurity.xsd file 453
operating systems 509
operational checklist 112, 507

P

parallel transactions 11, 108
parameters 56
partition 103
peer 146
per partition 11, 107
performance 507
Performance Monitoring Infrastructure
enabling 481
modules 485
monitoring 455
monitoring with 481
retrieving statistics from 483
Performance Monitoring Infrastructure (PMI) 115
placement
forcing 373
planning 73, 112, 232, 507, 509, 510
application deployment 73
PMI i
MBean 115
Portal 304
ports
configuring 232
profile
augmenting 45, 47
creating 45, 46

Profile Management Tool 57, 58, 59, 63, 65
Profile Management Tool plug-in 45, 46, 47
profiles
augment 48
create 48
non-root user 54
properties
server 199

R

real time 519
release notes 540
replication 146, 150
response file 55
response time 519
Rest data service
securing 446
REST data service
administering 339
configuring 327
properties file 327

S

security 453
authentication
creating an authenticator 435
LDAP 435
Tivoli access manager 435
WebSphere Application Server 435
credential 435
integration 445
integration with WebSphere Application Server 427
introduction 445
local 430
plug-ins 430
single sign-on (SSO) 435
XML configuration 450
Security 381
server properties 199
session management 293
session manager 288, 306
shard 103
silent installation 38
silently 61
SIP
session 300
session management 300
Spring 315
descriptor XML 315
extension beans 323
namespace 323
objectgrid.xsd file 321
XML configuration 321
stand-alone 241, 351, 446
start server
programmatically 364
starting
catalog server 356
container server 356
starting servers 351, 446

- startOgServer
 - options 356
- statistics
 - monitoring 455
 - with the statistics API 467
- statistics API 115
- stop server
 - programatically 364
- stopOgServer 362
- stopping servers 361
- support 134, 137, 540
- Support 539

T

- time-based data updater 268
- timeoutrequest retry 254
- Tivoli 494
- topology 78, 103
- trace
 - options for configuring 531
- transaction
 - callback 128
 - ID 128
- troubleshooting 529
 - messages 540
 - release notes 540
- tuning 232, 507, 509, 510, 515

U

- uninstalling 61
- upgrading 67

W

- wasprofile command 45, 47
- web console
 - connecting to catalog servers 458
 - creating custom reports in 466
 - monitoring with 457
 - overview 457
 - starting 457
- WebSphere Application Server 47, 48, 70
- WebSphere Customization Tools 57, 59, 63, 65
 - installing 58, 63
- WebSphere Portal 304
- Wily Introscope 500
- write-behind 133, 134, 137, 142
 - failed updates
 - handling 143
- wsadmin 207
- wxssetup.response.txt file 38

X

- XML 115
- xsadmin
 - configuration profile 473
- xsadmin utility
 - monitoring with 470, 474
 - verbose output for 479

Z

- zones 181
 - monitoring with xsadmin utility 188



Printed in USA