

IBM WebSphere eXtreme Scale Version 7.0

Administration Guide

March 11, 2011



This edition applies to version 7, release 0, 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* v

Chapter 1. Getting started with WebSphere eXtreme Scale. 1

Chapter 2. Planning application deployment 7

Deployment configurations for eXtreme Scale 7
Hardware and software requirements 7
 Java Platform, Enterprise Edition considerations 8
Tuning performance 9
 Planning for network ports 9
 Operating systems and network tuning 10
 ORB properties and file descriptor settings 11
 JVM tuning for WebSphere eXtreme Scale 12
 Configuring failover detection 13
High-availability catalog service 15
 Catalog server quorums 18
Operational checklist 26

Chapter 3. Capacity planning 29

Data grids, partitions, and shards 29
Sizing memory and partition count calculation 30
Sizing CPU per partition for transactions 32
Sizing CPUs for parallel transactions 33

Chapter 4. Installing and deploying WebSphere eXtreme Scale 35

Migrating to WebSphere eXtreme Scale Version 7.0 36
Installing stand-alone WebSphere eXtreme Scale 37
 Using the Object Request Broker with stand-alone WebSphere eXtreme Scale processes 38
Integrating WebSphere eXtreme Scale with WebSphere Application Server 39
 Using the Installation Factory plug-in to create and install customized packages 41
 Creating and augmenting profiles for WebSphere eXtreme Scale 53
Installing WebSphere eXtreme Scale silently 64
 Installation parameters 65
Using the Update Installer to install maintenance packages 67
Configuring a custom Object Request Broker 68
Uninstalling WebSphere eXtreme Scale 71

Chapter 5. Customizing Guide. 73

Installing the WebSphere Customization Tools 73
Generating customization definitions 74
Uploading and running customized jobs 75

Chapter 6. Configuring WebSphere eXtreme Scale 77

Configuring a local eXtreme Scale configuration 77

ObjectGrid interface 78
BackingMap interface 81
Map entry locking 85
 Configuring a locking strategy 87
Distributed eXtreme Scale grid configuration 89
WebSphere eXtreme Scale client configuration 92
 Enabling the client invalidation mechanism. 97
 Configuring the request retry timeout 99
Troubleshooting XML configuration 101
Loaders 105
 Loader configuration 108
 Write-behind caching support 114
 Configuring JPA loaders 124
 Configuring a JPA time-based data updater 126
Cache integration 128
 JPA cache plug-in 128
 HTTP session management 147
 WebSphere eXtreme Scale dynamic cache provider 159
Configuring evictors 177
 TimeToLive (TTL) evictor 177
 Plug in a pluggable evictor 179
Configuring the HashIndex 182
Configuring peer-to-peer replication with JMS 183
 Distributing changes between peer Java virtual machines 184
 JMS event listener 187
Configuring with XML files 190
 Deployment policy descriptor XML file 190
 ObjectGrid descriptor XML file 196
 Entity metadata descriptor XML file 217
 Security descriptor XML file 225
Properties file reference 229
 Server properties file 230
 Client properties file 236
 ORB properties file 239
Integrating with Spring framework 242
 Spring extension beans and namespace support 243
 Spring descriptor XML file 247
Using WebSphere Real Time 253
 WebSphere Real Time in a stand-alone environment 254
 WebSphere Real Time in WebSphere Application Server 256

Chapter 7. Administering the WebSphere eXtreme Scale environment 259

Setting the availability of an ObjectGrid 259
Starting stand-alone WebSphere eXtreme Scale servers 262
 Starting a stand-alone catalog service 263
 Starting container processes 267
 startOgServer script 270
 Embedded server API 274

Configuring ports in stand-alone mode	279
Stopping stand-alone eXtreme Scale servers	280
stopOgServer script	281
Administering WebSphere eXtreme Scale with WebSphere Application Server	283
Starting the catalog service process in a WebSphere Application Server environment	283
Starting eXtreme Scale containers automatically in WebSphere Application Server applications	286
Configuring ports in a WebSphere Application Server environment	289
Chapter 8. Monitoring your deployment environment	291
Statistics overview	291
Monitoring with the statistics API	294
Statistics modules	297
Monitoring performance with WebSphere Application Server PMI	299
Enabling PMI	300
Retrieving PMI statistics	302
PMI modules	304
Using Managed Beans (MBeans) to administer your environment	312
Monitoring with managed beans (MBeans)	313
Accessing Managed Beans (MBeans) using the wsadmin tool	314
Accessing Managed Beans (MBeans) programmatically	314
Using the xsAdmin sample utility	319
Vendor tools	322
Monitoring with the IBM Tivoli Enterprise Monitoring Agent for WebSphere eXtreme Scale	323
Monitoring eXtreme Scale applications with CA Wily Introscope	329

Monitoring eXtreme Scale with Hyperic HQ	333
Logs and trace	335
Trace options	337

Chapter 9. Securing the deployment environment	341
Data grid security	342
Enabling local security	343
Application client authentication	344
Application client authorization	346
Transport layer security and secure sockets layer	349
Data grid authentication	351
Java Management Extensions (JMX) security	352
Security descriptor XML file	354
Security integration with WebSphere Application Server	357
Starting and stopping secure eXtreme Scale servers	358

Chapter 10. Troubleshooting	363
Logs and trace	363
Trace options	365
Troubleshooting loaders	367
Troubleshooting client connectivity problems	368
Messages	369
Release notes	369

Chapter 11. Glossary	371
Notices	393
Trademarks	395
Index	397

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 *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.
- The *Product Overview* contains a high-level view of WebSphere eXtreme Scale concepts, including use case scenarios, and tutorials.

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.

How this book is structured

The book contains information about the following major topics:

- **Chapter 1** includes information about getting started.
- **Chapter 2** includes information about planning application deployment.
- **Chapter 3** includes information about capacity planning.
- **Chapter 4** includes information about installing the product.
- **Chapter 5** includes information about customizing for the z/OS® platform.
- **Chapter 6** includes information about configuring the product.
- **Chapter 7** includes information about administering the environment.
- **Chapter 8** includes information about monitoring your deployment environment.
- **Chapter 9** includes information about securing the deployment environment.
- **Chapter 10** includes information about troubleshooting the environment.
- **Chapter 11** includes information the product glossary.

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. Getting started with WebSphere eXtreme Scale

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 eXtreme Scale data grid and client can be used. The getting started sample is in the *installRoot/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 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 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 data 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 installRoot/ObjectGrid/gettingstarted
```

Substitute *installRoot* with the path to the eXtreme Scale installation root directory or the root file path of the extracted eXtreme Scale trial *installRoot*.
3. Set or export the JAVA_HOME environmental variable to reference a valid JDK or JRE Version 1.5 or later installation directory.

```
UNIX Linux export JAVA_HOME=java_home_directory
```

```
Windows set JAVA_HOME=java_home_directory
```

4. 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.

5. 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 step 5 and 6 if you want to start more container server instances to support replication.

6. 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`

7. 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 `installRoot/ObjectGrid/gettingstarted/xml` directory to start a container server. The `objectgrid.xml` file is the ObjectGrid descriptor XML file and the `deployment.xml` file is the ObjectGrid deployment policy descriptor XML file. Both 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 start-up. 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 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 `installRoot/ObjectGrid/gettingstarted/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. Connecting 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 by this sample only requires catalog server endpoint in the format of `hostname:port`, such as `localhost:2809`. If the connection to the catalog server succeeds, the `connect` method returns a `ClientClusterContext` instance. The `ClientClusterContext` instance is required to obtain the `ObjectGrid` from `ObjectGridManager` API. 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);
```

2. Obtaining 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 `ObjectGrid` instance. The `ClientClusterContext` instance is obtained during connecting to catalog server. The name of `ObjectGrid` is "Grid" that is specified in the `objectgrid.xml` file. The

following code snippet demonstrates how to obtain ObjectGrid by calling getObjectGrid method of ObjectGridManager API.

```
ObjectGrid grid = ObjectGridManagerFactory.getObjectGridManager().getObjectGrid(ccc, "Grid");
```

3. Getting a Session instance

You can get a Session from the obtained ObjectGrid instance. A Session instance is required to get ObjectMap, and perform transaction demarcation. The following code snippet demonstrates how to get Session by calling getSession method of ObjectGrid API.

```
Session sess = grid.getSession();
```

4. Getting an ObjectMap instance

After getting a Session, you can get ObjectMap from a Session by calling getMap method of Session API. You have to pass the name of map as parameter to getMap method in order to get the ObjectMap. The following code snippet demonstrates how to obtain ObjectMap by calling getMap method of Session API.

```
ObjectMap map1 = sess.getMap("Map1");
```

5. Using the ObjectMap methods

After an ObjectMap is obtained, you can use ObjectMap API. Remember ObjectMap 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, ObjectMap operations run with auto-commit transactions.

The following code snippet demonstrates how to use ObjectMap API with auto-commit transaction.

```
map1.insert(key1, value1);
```

The following code snippet demonstrates how to use 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 WebSphere Application Server deployment manager process and each WebSphere Application Server instance hosts an eXtreme Scale 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.

Chapter 2. Planning application deployment

Before you deploy applications on WebSphere eXtreme Scale, you must review the hardware and software requirements, networking and tuning settings, deployment configurations, 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.

Deployment configurations for eXtreme Scale

WebSphere eXtreme Scale can be deployed into two types of environments: local or distributed. The configuration that is necessary depends on the type of deployment environment.

Local environments

When you deploy in a local environment, eXtreme Scale runs within a single Java virtual machine, and is not replicated. A local environment requires an ObjectGrid XML file or ObjectGrid APIs.

Distributed environments

When you deploy in a distributed environment, eXtreme Scale runs across a set of Java virtual machines. With this configuration, you can use data replication and partitioning. A distributed environment can be further classified as a dynamic deployment topology in which you can add servers as needed. As with a local environment, an ObjectGrid XML file, or an equivalent programmatic configuration, is needed in a distributed environment. Additionally, you must provide a deployment policy XML file with configuration details for your eXtreme Scale in-memory data grid.

Hardware and software requirements

You are not required to use a specific level of hardware or operating system for WebSphere eXtreme Scale.

However, for the official detailed list of supported hardware and software options by operating system for WebSphere eXtreme Scale, see the System Requirements page.

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.

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

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.

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 + Interim Fix PK85622
- WebSphere Application Server Version 7.0.0.3 or higher + 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.

Related tasks

“Migrating to WebSphere eXtreme Scale Version 7.0” on page 36

To upgrade from Version 6.1.0.x to Version 7.0, merge any modified product script files with new product script files to maintain your changes.

Chapter 7, “Administering the WebSphere eXtreme Scale environment,” on page 259

Administering the product environment includes starting and stopping servers in stand-alone mode or in WebSphere Application Server. You can also use WebSphere eXtreme Scale as a session manager in a WebSphere Application Server environment.

“Stopping stand-alone eXtreme Scale servers” on page 280

You can use the stopOgServer script to stop server processes.

Chapter 4, “Installing and deploying WebSphere eXtreme Scale,” on page 35

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.

Java Platform, Enterprise Edition considerations

As you prepare to integrate WebSphere eXtreme Scale in a Java Platform, Enterprise Edition environment, consider certain items, such as 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 in the *Programming Guide* for more information.

Tuning performance

To improve performance, consider certain items such as operating system and network tuning, planning for network ports, Java Virtual Machine (JVM) tuning for WebSphere eXtreme Scale settings, configuring failover, and other tuning topics.

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 and other machines. You should plan and control your ports, especially in a firewall environment such as when you are using a catalog service and containers on multiple ports.

Catalog service grid

A catalog service grid requires the following.

1. peerPort for the high-availability (HA) manager to communicate between peer catalog servers over a TCP stack
2. clientPort for catalog servers to access catalog service data
3. JMXServicePort to specify which port the JMX service should use
4. ORB listener port for containers and clients to communicate with the catalog service through the ORB

Use the **startOgServer** command to specify the ports previously listed with the option in stand-alone, as shown in the following example:

```
-catalogServiceEndpoints cs1:host1:clientPort:peerPort, cs2:host2:clientPort:peerPort,
cs3:host3:clientPort:peerPort -listenerPort <orbPort> -JMXServicePort <jmxPort>
```

In a WebSphere Application Server environment, the ports in the previous list are specified in WebSphere cell custom properties with key catalog.services.cluster as:

- cell\node1\server1:host1:clientPort:peerPort:listenerPort
- cell\node2\server2:host2:clientPort:peerPort:listenerPort

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. In the firewall environment, since it is advantageous for you to plan and control ports, options are provided to start eXtreme Scale container servers with specified HAManager port and ORB listener port with an option in the startOgServer command, as shown in the following example:

```
-HaManagerPort <peerPort>
-listenerPort <orbPort>
```

Proper planning of port control is a benefit, but there is an inherent difficulty to plan and manage these ports when hundreds of Java Virtual Machines are started in a machine. Any port conflict will cause a failure of server startup.

When security is enabled, a Secure Socket Layer (SSL) port is a required addition to the ports listed previously. Using Dcom.ibm.CSI.SSLPort=<sslPort> as a -jvmArgs argument sets the SSL port to <sslPort>. Read more about security settings for eXtreme Scale to assist in planning for ports.

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:0000ffff
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
```

ORB properties and file descriptor settings

Tuning considerations include Object Request Broker (ORB) properties and file descriptor settings.

ORB properties

The ORB is used by WebSphere eXtreme Scale to communicate over a TCP stack. The necessary orb.properties file is in the java/jre/lib directory. For a heavy load of large objects, enable ORB fragmentation by specifying the following setting:

```
com.ibm.CORBA.FragmentSize=<right size>
```

Prevent ThreadPool growth by specifying the following setting:

```
com.ibm.CORBA.ThreadPool.IsGrowable=false
```

Set proper timeouts to avoid excess threads in an abnormal situation by specifying the following settings:

- com.ibm.CORBA.RequestTimeout
- com.ibm.CORBA.ConnectTimeout
- com.ibm.CORBA.FragmentTimeout

File descriptor

UNIX and Linux systems there is a limit to the number of open files allowed per process. The operating system specifies the number of open files permitted. If this value is set too low, a memory allocation error will occur 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`.

JVM tuning for WebSphere eXtreme Scale

Java virtual machine (JVM) tuning can yield a significant improvement in your deployment of WebSphere eXtreme Scale.

In most cases, WebSphere eXtreme Scale requires little or no special JVM settings. If you have a large amount of objects that are being stored in WebSphere eXtreme Scale, adjust the heap size to an appropriate level to avoid running out of memory.

Platforms

Performance testing occurred primarily on AIX (32 way), Linux (4 way), and Windows (8 way) boxes. With high-end AIX boxes, heavily multi-threaded scenarios can be pushed, and contention points can be identified and fixed.

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 out of the box with IBM-provided Java virtual machines. WebSphere eXtreme Scale requires a working ORB to operate. You can use WebSphere eXtreme Scale with third party ORBs, but if a problem in the ORB is identified, you must contact the ORB vendor for support. The IBM ORB implementation is compatible with third partyJava virtual machines and can be substituted if needed.

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. If you see a performance problem, then 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.

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 2 Platform, Standard Edition (J2SE). WebSphere eXtreme Scale Version 6.1 supports J2SE Version 1.4.2 and above. For improved developer productivity and performance, use J2SE 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 Version 1.4.2 or above, but if a problem on the JVM is identified, you must contact the JVM vendor for support. If possible, use the JVM from the WebSphere runtime on any platform that WebSphere Application Server supports.

For most of the scenarios in which WebSphere eXtreme Scale is used, Java Platform Standard Edition 6 of the JVM performs better than Edition 5 or 1.4. Java 2 Platform, Standard Edition Version 1.4 performs poorly especially for scenarios that use the gencon collector. Java Platform Standard Edition 5 performs well, but Java Platform, Standard Edition 6 performs better.

Large heaps

When the application needs to manage a large amount of data for each partition, then garbage collection might be a factor. A read mostly scenario performs well even with very 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 box occurs. This pause can be large on smaller boxes with large heaps.

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, and 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 253 for more information.

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 will 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.

Configuring 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.bat** | **startOgServer.sh** script file. Set this parameter to one of the following values:

Table 1. 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 disconnect 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 V6.1.0.13 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.

High-availability catalog service

A catalog service is the 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 data grid for high availability.

Components of the catalog service

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 server grid is propagated to the clients through the Common Object Request Broker Architecture (CORBA) service context.

Configure at least three catalog servers. 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 server grid 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 server grid and the container server grid are very different. The catalog server grid is for high availability of your system data. The container grid is for your data high availability, scalability, and workload management. Therefore, two different routing tables exist: the routing table for the catalog server grid and the routing table for the server grid shards.

The catalog responsibilities are divided into a series of services. The core group manager performs peer grouping for health monitoring, the placement service performs allocation, the administration service provides access to administration, and the location service manages locality.

Catalog grid deployment

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 placement of shards across the set of available containers. The placement service is responsible for maintaining a balance of resources across physical resources. The placement service is responsible for allocating individual shards to their host container. It runs as a one-of-N elected service in the grid, so there is always exactly one instance of the service 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 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 containers that are registering hosted applications with the placement service. The location service runs on all of the grid members to scale out this function.

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 grid.

Planning

After a catalog grid is started, the members of the grid bind together. Carefully plan your catalog grid topology, because you cannot modify your catalog configuration at run time. Spread out the grid as diversely as possible to prevent errors.

Starting a catalog server grid

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

You can configure eXtreme Scale containers that are embedded in a WebSphere Application Server environment to connect to a stand-alone catalog grid. Use the same property to connect the application server to the catalog grid. However, the property does not manage the life cycle of the catalog with the servers. Instead, the property allows the containers to locate the remote catalog grid. To set the property, see .

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

Quorum is the minimum number of catalog servers necessary to conduct placement operations for the data grid. The minimum number is the full set of catalog servers unless quorum has been overridden.

Important terms

The following is a list of terms related to quorum considerations for eXtreme Scale.

- **Brown out:** A brown out is the temporary loss of connectivity between one or more servers.
- **Black out:** A black out is the permanent loss of connectivity between one or more servers.
- **Data center:** A data center is a geographically located group of servers generally connected with a local area network (LAN).
- **Zone:** A zone is a configuration option used to group servers together that share some physical characteristic. The following are some examples of zones for a group of servers: a data center as described in the previous bullet, an area network, a building, a floor of a building, and so on.
- **Heartbeat:** Heartbeating is a mechanism used to determine whether or not a given Java virtual machine (JVM) is running.

Topology

This section explains how IBM WebSphere eXtreme Scale operates across a network that includes unreliable components. Examples of such a network would include a network spanning multiple data centers.

IP address space

WebSphere eXtreme Scale requires a network where any addressable element on the network can connect to any other addressable element on the network unimpeded. This means WebSphere eXtreme Scale requires a flat IP address naming space and it requires all firewalls to allow traffic to flow between the IP addresses and ports used by the Java virtual machines (JVM) hosting elements of WebSphere eXtreme Scale.

Connected LANs

Each LAN is assigned a zone identifier for WebSphere eXtreme Scale requirements. WebSphere eXtreme Scale aggressively heartbeats JVMs in a single zone and a missed heartbeat will result in a failover event if the catalog service has quorum. Read about [Configuring failover detection](#) for more information.

Catalog service data grid and container servers

A data grid is a collection of similar JVMs. A catalog service is a data grid composed of catalog servers, and is fixed in size. However, the number of container servers is dynamic. Container servers can be added and removed on demand. In a three-data-center configuration, WebSphere eXtreme Scale requires one catalog service JVM per data center.

The catalog service data grid uses a full quorum mechanism. This means that all members of the data grid must agree on any action.

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 will only include JVMs from the same zone. JVMs from different zones will never be in the same core group.

A core group will aggressively try to detect the failure of its member JVMs. The container JVMs of a core group must never span multiple LANs connected with links like in a wide area network. This means that a core group cannot have containers in the same zone running in different data centers.

Server life cycle

Catalog server startup

The catalog servers are started using the startOgServer command. The quorum mechanism is disabled by default. To enable quorum either pass the -quorum enabled flag on the startOgServer command or add the enableQuorum=true property in the property file. All of the catalog servers must be given the same quorum setting.

```
# bin/startOgServer cat0 -serverProps objectGridServer.properties
objectGridServer.properties file
catalogClusterEndPoints=cat0:cat0.domain.com:6600:6601,
cat1:cat1.domain.com:6600:6601
catalogServiceEndPoints= cat0.domain.com:2809, cat1.domain.com:2809
enableQuorum=true
```

Container server startup

The container servers are started using the startOgServer command. When running a data grid across data centers the servers must use the zone tag to identify the data center in which they reside. Setting the zone on the data grid servers allows WebSphere eXtreme Scale to monitor health of the servers scoped to the data center, minimizing cross-data-center traffic.

```
# bin/startOgServer gridA0 -serverProps objectGridServer.properties -
objectgridfile xml/objectgrid.xml -deploymentpolicyfile xml/
deploymentpolicy.xml
objectGridServer.properties file
catalogServiceEndPoints= cat0.domain.com:2809, cat1.domain.com:2809
zoneName=ZoneA
```

Data grid server shutdown

The grid servers are stopped using the stopOgServer command. When shutting down an entire data center for maintenance, pass the list of all the servers that belong to that zone. This will allow a clean transition of state from the zone in teardown to the surviving zone or zones.

```
# bin/stopOgServer gridA0,gridA1,gridA2 -catalogServiceEndPoints
cat0.domain.com:2809,cat1.domain.com:2809
```

Failure detection

WebSphere eXtreme Scale detects process death through abnormal socket closure events. The catalog service will be told immediately when a process terminates. A black out is detected through missed heartbeats. WebSphere eXtreme Scale protects

itself against brown out conditions across data centers by using a quorum implementation.

Heartbeating implementation

This section describes how liveness checking is implemented in WebSphere eXtreme Scale.

Core group member heartbeating

The catalog service places container JVMs into core groups of a limited size. A core group will try to detect the failure of its members using two methods. If a JVM's socket is closed, that JVM is regarded as dead. Each 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 period of time then the JVM is regarded as dead.

A single member of a core group is always elected to be the leader. The core group leader (CGL) is responsible to periodically tell the catalog service that the core group is alive and to report any membership changes to the catalog service. A membership change can be a JVM failing or a newly added JVM joining the core group.

If the core group leader cannot contact any member of the catalog service grid then it will continue to retry.

Catalog service grid heartbeating

The catalog service 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 heartbeating configuration.

Core group heartbeating

The catalog service needs to know when container servers fail. Each core group is responsible for determining container JVM failure and reporting this to the catalog service through the core group leader. 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.

If the catalog service marks a container JVM as failed and the container is later reported as alive, the container JVM will be told to shutdown the WebSphere eXtreme Scale container servers. A JVM in this state will not be visible in xsadmin queries. There will be messages in the logs of the container JVM indicating this has happened. These JVMs need to be manually restarted.

If a quorum loss event has occurred, heartbeating is suspended until quorum is reestablished.

Catalog service quorum behavior

Normally, the members of the catalog service have full connectivity. The catalog service grid is a static set of JVMs. WebSphere eXtreme Scale expects all members of the catalog service to be online always. The catalog service will only respond to container events while the catalog service has quorum.

If the catalog service loses quorum, it will wait for quorum to be reestablished. While the catalog service does not have quorum, it will ignore events from container servers. Container servers will retry any requests rejected by the catalog server during this time as WebSphere eXtreme Scale expects quorum to be reestablished.

The following message indicates that quorum has been lost. Look for this message in your catalog service logs.

```
CWOBJ1254W: The catalog service is waiting for quorum.
```

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

- Catalog service JVM member fails
- Network brown out
- Data center loss

Stopping a catalog server instance using `stopOgServer` does not cause loss of quorum because the system knows the server instance has stopped, which is different from a JVM failure or brown out.

Quorum loss from JVM failure

A catalog server that fails will cause quorum to be lost. In this case, quorum should be overridden as fast as possible. The failed catalog service cannot rejoin the grid until quorum has been overridden.

Quorum loss from network brown out

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

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

Catalog service JVM cycling

If a catalog server is stopped by using `stopOgServer`, then the quorum drops to one less server. This means the remaining servers still have quorum. Restarting the catalog server bumps quorum back to the previous number.

Consequences of lost quorum

If a container JVM was to fail while quorum is lost, recovery will not take place until the brown out recovers or in the case of a black out the customer does an override quorum command. WebSphere eXtreme Scale regards a quorum loss event and a container failure as a double failure, which is a rare event. This means that applications may lose write access to data that was stored on the failed JVM until quorum is restored at which time normal recovery will take place.

Similarly, if you attempt to start a container during a quorum loss event, the container will 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 brown out occurs then some clients may not have access to primary or replica copies of the data until the brown out clears.

New clients can be started, as there should be a catalog service JVM in each data center so at least one catalog service JVM can be reached by a client even during a brown out event.

Quorum recovery

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

Container behavior

This section describes how the container server JVMs behave while quorum is lost and recovered.

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 will honor that assignment until new instructions arrive from the catalog service. This means that if a primary shard in a container cannot communicate with a replica shard because of a brown out then it will continue to retry until it receives new instructions from the catalog service.

If a network brown out occurs and a primary shard loses communication with the replica then it will retry the connection until the catalog service provides new instructions.

Synchronous replica behavior

While the connection is broken the primary can accept new transactions as long as there are at least as many replicas online as the `minsync` property for the map set. If any new transactions are processed on the primary while the link to the synchronous replica is broken, the replica will be cleared and resynchronized with the current state of the primary when the link is reestablished.

Synchronous replication is strongly discouraged between data centers or over a WAN-style link.

Asynchronous replica behavior

While the connection is broken the primary can accept new transactions. The primary will buffer 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

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

Quorum commands with xsadmin

This section describes xsadmin commands useful for quorum situations.

Querying quorum status

The quorum status of a catalog server instance can be interrogated using the xsadmin command.

```
xsadmin -ch cathost -p 1099 -quorumstatus
```

There are five possible outcomes.

- Quorum is disabled: The catalog servers are running in a quorum-disabled mode. This is a development or single only data center mode. It is not recommended for multi-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.

Overriding quorum

The xsadmin command can be used to override quorum. Any surviving catalog server instance can be used. All survivors are notified when one is told to override quorum. The syntax for this is as follows.

```
xsadmin -ch cathost -p 1099 -overridequorum
```

Diagnostic commands

- Quorum status: As described in the previous section.
- Coregroup list: This displays a list of all core groups. The members and leaders of the core groups are displayed.

```
xsadmin -ch cathost -p 1099 -coregroups
```

- **Teardown servers:** This command removes a server manually from the grid. This is not needed normally since servers are automatically removed when they are detected as failed, but the command is provided for use under IBM support help.

```
xsadmin -ch cathost -p 1099 -g Grid -teardown server1,server2,server3
```

- **Display route table:** This command shows the current route table by simulating a new client connection to the 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
```

- **Display unassigned shards:** If some shards cannot be placed on the grid then this can be used to list them. This only happens when the placement service has a constraint preventing placement. For example, if you start JVMs on a single physical box while in production mode then only primary shards can be placed. Replicas will be unassigned until JVMs start on a second box. The placement service only places replicas on JVMs with different IP addresses than the JVMs 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
```

- **Set trace settings:** This command sets the trace settings for all JVMs matching the filter specified for the xsadmin command. This setting only changes the trace settings 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 enables trace for all JVMs on the box with the host name specified, in this case host1.

- **Checking map sizes:** The map sizes command is useful for verifying that key distribution is uniform over the shards in the key. If some containers have significantly 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
```

Overriding quorum

This should only be used when a data center failure has occurred. Quorum loss due to a catalog service JVM failure or a network brownout should recovery automatically once the catalog service JVM is restarted or the network brownout clears.

Administrators are the only ones with knowledge of a data-center failure. WebSphere eXtreme Scale treats a brownout and a blackout similarly. You are required to inform the eXtreme Scale environment of any outages using the xsadmin command to override quorum which causes the catalog service to register that quorum is achieved with the current membership. Having achieved quorum allows your deployment to fully recover. Issuing an override quorum command confirms that JVMs in the failed data-center have failed beyond recovery.

The following list considers some scenarios for overriding quorum. Assume that you have deployed three catalog servers: A, B, and C.

- **Brown out:** Say you have a brown out where C is isolated temporarily. The catalog service will lose quorum and wait for the brown out to clear at which point C rejoins the catalog service grid and quorum is reestablished. Your application sees no problems during this time.

- Temporary failure: Here, C fails and the catalog service loses quorum, so you must override quorum. Once quorum is reestablished, C can be restarted. C will rejoin the catalog service grid when it is restarted. The application is unaffected during a temporary failure.
- Data center failure: You verify that the data center actually failed and that it has been isolated on the network. Then you issue the `xsadmin override quorum` command. The two surviving data centers fully recover by replacing shards that were hosted in the failed data center. The catalog service is now running with a full quorum of A and B. The application may see delays or exceptions during the interval between the start of the black out and when quorum is overridden. Once quorum is overridden the grid recovers and normal operation is resumed.
- Data center recovery: The surviving data centers are already running with quorum overridden. When the data center containing C is restarted, all JVMs in the data center must be restarted. Then C will rejoin the existing catalog service grid and quorum will revert to the normal situation with no user intervention.
- Data center failure and brown out: The datacenter containing C fails. Quorum is overridden and recovered on the remaining data centers. If a brown out between A and B occurs, the normal brown out recovery rules apply. Once the brown out clears, quorum is reestablished and necessary recovery from the quorum loss occurs.

Transport security considerations

Since data centers are normally deployed in different geographical locations, users might want to enable transport security between the data centers for security reasons.

Read about transport layer security in the *Administration Guide*.

Related tasks

“Using the xsAdmin sample utility” on page 319

With the xsAdmin sample 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.

Operational checklist

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

Table 2. 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 file” on page 239</p>

Table 2. 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 270</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>Chapter 9, “Securing the deployment environment,” on page 341</p>
<p>Choose how you are going to monitor your environment.</p> <ul style="list-style-type: none"> • xsAdmin <ul style="list-style-type: none"> – The JMX ports of the catalog servers need to be visible to the XSAAdmin tool. The container ports also need to be accessible for some commands that gather information from the containers. • You can choose between the following vendor monitoring tools: <ul style="list-style-type: none"> – Tivoli® Enterprise Monitoring Agent – CA Wily Introscope – Hyperic HQ 	<ul style="list-style-type: none"> • “Using the xsAdmin sample utility” on page 319 • “Java Management Extensions (JMX) security” on page 352 • “Monitoring with the IBM Tivoli Enterprise Monitoring Agent for WebSphere eXtreme Scale” on page 323 • “Monitoring eXtreme Scale with Hyperic HQ” on page 333 • “Monitoring eXtreme Scale applications with CA Wily Introscope” on page 329

Chapter 3. 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.

Data grids, partitions, and shards

An eXtreme Scale distributed data grid is divided into partitions. A partition holds an exclusive subset of the data. A partition is made up of one or more shards: a primary shard and replica shards. You do not need to have replica shards in a partition, but replica shards provide high availability. Whether your deployment is an independent in-memory data grid or an in-memory database processing space, data access in eXtreme Scale relies heavily on sharding concepts.

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

Two placement strategies exist: `FIXED_PARTITIONS` (default) and `PER_CONTAINER`. The following discussion focuses on the usage of the `FIXED_PARTITIONS` strategy.

Number of shards

If your environment included ten partitions that hold one million objects with no replicas, then ten shards would exist that each store 100,000 objects. If you add a replica to this scenario, then an extra shard exists in each partition. In this case, 20 shards exist - ten primary shards and ten replica shards. Again, each one of these shards store 100,000 objects. Each partition consists of a primary shard and one or more (N) replica shards. Determining the optimal shard count is critical. If you configure a small number of shards, data is not distributed evenly among the shards, resulting in out of memory errors and processor overloading issues. You must have at least ten shards for each JVM as you scale. When you are initially deploying the data grid, you would potentially use a large number of partitions.

Number of shards per JVM

Scenario: small number of shards for each JVM

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

- 7 Java virtual machines with 2 shards
- 2 Java virtual machines with 3 shards

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

Scenario: increased number of shards per JVM

In this scenario, consider a much larger number of shards. In this scenario, there are 101 shards with 9 Java virtual machines hosting 10GB of data. In this case, each shard holds 99 MB of data. The Java virtual machines have the following distribution of shards:

- 7 Java virtual machines with 11 shards
- 2 Java virtual machines with 12 shards

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

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

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, which 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, 10 shards for each JVM should be used. As described before, each JVM has one primary shard and (N-1) shards for the replicas, or in this case, 9 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 7 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 billion 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, you would start with 250 Java virtual machines and grow toward 500 Java virtual machines over 5 years, which allows you to 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). You should 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 3.

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. You should repeat the sizing process considering future growth again, and reset the partition count. This practice requires a full grid restart, or an outage of your 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.

Chapter 4. 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. See WebSphere eXtreme Scale architecture and topology overview in the *Product Overview* for more information.
- Verify your environment meets the prerequisites to install eXtreme Scale. See “Hardware and software requirements” on page 7 for more information.

About this task

Supported environments

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

Install stand-alone eXtreme Scale

You can install stand-alone eXtreme Scale in an environment that does not contain WebSphere Application Server or WebSphere Application Server Network Deployment. With the stand-alone option, you define a new installation location to install the eXtreme Scale server.

Integrate the product with WebSphere Application Server or WebSphere Application Server Network Deployment

You can install and integrate eXtreme Scale with an existing installation of WebSphere Application Server or WebSphere Application Server Network Deployment. You can select to install both the eXtreme Scale client and server, or you can install the client only.

Create and augment profiles

Create and augment profiles to use eXtreme Scale features. 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.

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. To do so, you have to 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.

Apply maintenance

Use the IBM Update Installer Version 7.0.0.4 or later to apply maintenance to your environment.

Related concepts

“Hardware and software requirements” on page 7

You are not required to use a specific level of hardware or operating system for WebSphere eXtreme Scale.

Related reference

“Installation parameters” on page 65

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

Migrating to WebSphere eXtreme Scale Version 7.0

To upgrade from Version 6.1.0.x to Version 7.0, merge any modified product script files with new product script files to maintain your changes.

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 7 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 7.0 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 7.0. See Chapter 4, “Installing and deploying WebSphere eXtreme Scale,” on page 35 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 7, “Administering the WebSphere eXtreme Scale environment,” on page 259 for more information.

Related concepts

“Hardware and software requirements” on page 7

You are not required to use a specific level of hardware or operating system for WebSphere eXtreme Scale.

Installing stand-alone WebSphere eXtreme Scale

You can install stand-alone WebSphere eXtreme Scale 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 eXtreme Scale or the ObjectGrid component exists in the directory that you specify to install Version 7.0, the product is not installed. For example, you may have a previously existing <wxs_install_root>/ObjectGrid folder. You can either choose a different installation directory or cancel the installation. Next, uninstall the previous installation and run the wizard again.

- For improved performance and serviceability, download and install an IBM developer kit from developerWorks.

Restriction: Windows If you are using hardware from an independent vendor, select one of the following options to download and install a developer kit.

- Download a Sun Java Development Kit (JDK).
- Download a JDK or Java Runtime Environment (JRE) from another independent software vendor.

About this task

When you install the product as stand-alone, you install the eXtreme Scale client and server independently. Server and client processes, therefore, access all required resources locally. You can also embed eXtreme Scale into existing Java Platform, Standard Edition (J2SE) applications by using scripts and Java archive (JAR) files.

WebSphere eXtreme Scale relies on ObjectGrid processes and related APIs. The following table lists the JAR files that are included in the installation.

Table 3. Runtime files in the /ObjectGrid/lib installation directory

File name	Environment	Description
cglib.jar	Local, client, and server	The cglib.jar file is read by the cglib utility function when you are using the copy-on-write copy mode and when you are using the EntityManager API to track the changes of an entity. The file is automatically included in the server runtime environment when you use the supplied scripts. Add this file to your client or local runtime environment. The cglib.jar file is a CGLIB library and its version is cglib-nodep-2.1.3.jar.
objectgrid.jar	Local, client, and server	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.
ogagent.jar	Local, client, and server	The ogagent.jar file contains the runtime classes that are required to run the Java instrumentation agent that is used with the EntityManager API.

Table 3. Runtime files in the /ObjectGrid/lib installation directory (continued)

File name	Environment	Description
ogclient.jar	Local and client	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.
wsogclient.jar	Local and client	The wsogclient.jar file is included when you install the product in an environment that contains WebSphere Application Server Version 6.0.2 and later. This file contains only the local and client runtime environments.
wxsdynacache.jar	Server only	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. Attention: The file is in the ObjectGrid/dynacache/lib directory.

Procedure

- Use the wizard to complete the installation. Run the following script to start the wizard:
 - `Linux` `UNIX` `dvd_root/install`
 - `Windows` `dvd_root\install.bat`
- Follow the prompts in the wizard, and click **Finish** to complete the installation.

Restriction: 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.

What to do next

Read about configuring eXtreme Scale to set up your client application processes and server processes.

Related reference

“Installation parameters” on page 65

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

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 file” on page 239

on page 239 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” on page 68.

Related reference

“ORB properties file” on page 239

The orb.properties file is used to pass the properties that are used by the Object Request Broker (ORB) to modify the transport behavior of the data grid.

 Object Request Broker custom properties

Integrating WebSphere eXtreme Scale with WebSphere Application Server

You can install WebSphere eXtreme Scale 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 eXtreme Scale.
- Stop all processes that are running in your WebSphere Application Server or WebSphere Application Server Network Deployment. See Command-line utilities for more information.

Important: When you install eXtreme Scale, 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 eXtreme Scale 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 eXtreme Scale grids and access the grids using a client connection.

The following table lists the Java archive (JAR) files that are included in the installation.

Table 4. Runtime files in the /lib installation directory

File name	Environment	Description
cglib.jar	Local, client, and server	The cglib.jar file is read by the cglib utility function when you are using the copy-on-write copy mode and when you are using EntityManager API to track the changes of an entity. The cglib.jar file is a CGLIB library and its version is cglib-nodep-2.1.3.jar.
ogagent.jar	Local, client, and server	The ogagent.jar file contains the runtime classes that are required to run the Java instrumentation agent that is used with the EntityManager API.
wsobjectgrid.jar	Local, client, and server	The wsobjectgrid.jar file contains the eXtreme Scale local, client, and server run times.
wsogclient.jar	Local and client	The wsogclient.jar file is included when you install the product in an environment that contains WebSphere Application Server Version 6.0.2 and later. This file contains only the local and client run times.
wxdynacache.jar	Server only	The wxsdynacache.jar file contains the necessary classes to use with the dynamic cache provider.

Procedure

1. Use the wizard to complete the installation. Run the following script to start the wizard:

- `Linux` `UNIX` `dvd_root/install`
- `Windows` `dvd_root\install.bat`

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.

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 “Administering WebSphere eXtreme Scale with WebSphere Application Server” on page 283 for more information.

Related reference

“Installation parameters” on page 65

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

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 48 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:

- **UNIX** **Linux** `ifgui.sh`
- **Windows** `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.

Installing a CIP:

Simplify the product installation process by installing a customized installation package (CIP), which 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” 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 53 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 “Administering WebSphere eXtreme Scale with WebSphere Application Server” on page 283 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 42 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_home/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 the CIP identifier prepended to its filename. 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, 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 choose to 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 45 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 42 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 “Administering WebSphere eXtreme Scale with WebSphere Application Server” on page 283 for more information.

Related reference

“Installation parameters” on page 65

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

wxssetup.response.txt file:

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

Change the response file for your requirements

```
#####
#
# IBM WebSphere eXtreme Scale V7.0.0 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.xls\license.xls. 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"

#####
# 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 only feature option for
# the eXtreme Scale standalone installation is 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"

#
# 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

```

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 “Integrating WebSphere eXtreme Scale with WebSphere Application Server” on page 39 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:

```

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.

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. Alternatively, repeat any of the preceding tasks to create or augment additional profiles.

Related reference

“Installation parameters” on page 65

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

“manageprofiles command” on page 56

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.

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

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:

```
install_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 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.

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.

The Action Selection page is displayed only if at least one profile and the augment templates exist.

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 `install_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 “Administering WebSphere eXtreme Scale with WebSphere Application Server” on page 283 for more information.

Related reference

“manageprofiles command” on page 56

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.

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:

```
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.

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 if you installed only 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 `install_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 `install_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 “Administering WebSphere eXtreme Scale with WebSphere Application Server” on page 283 for more information.

Related reference

“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.

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 “Integrating WebSphere eXtreme Scale with WebSphere Application Server” on page 39 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
```

Related tasks

“Creating and augmenting profiles for WebSphere eXtreme Scale” on page 53
After you install the product, create unique types of profiles and augment existing profiles for WebSphere eXtreme Scale.

“Using the graphical user interface to create profiles” on page 54

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.

“Using the graphical user interface to augment profiles” on page 55

After you install the product, you can augment an existing profile to make it compatible with WebSphere eXtreme Scale.

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 plugin, 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 he 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 further details, read the detailed information on creating profiles for non-root users, which includes the steps to complete the preceding task examples, in the WebSphere Application Server Network Deployment Information Center.

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 he creates. Follow the WebSphere Application Server Network Deployment non-root user augmentation process to complete these tasks.

However, when a non-root user augments a profile that is created by the installer, the following files do not need to be created by the non-root user before augmentation, because the files were established during the profile creation process:

- `app_server_root/logs/manageprofiles.xml`
- `app_server_root/properties/fsdb.xml`
- `app_server_root/properties/profileRegistry.xml`

When a non-root user augments a profile that he 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. To do so, you have to 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 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.

Before you begin

Stop all processes that are running in your WebSphere Application Server or WebSphere Application Server Network Deployment environment. Read about command-line utilities for more information about the `stopManager`, `stopNode`, and `stopServer` commands.

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 49 including a description of each option.

Procedure

1. Optional: If you choose to install eXtreme Scale using a response file, first customize the 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. Copy the response file from the product DVD to your disk drive.
- 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:
 - The license agreement
 - The installation directory

Tip: When you install eXtreme Scale 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.

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

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:

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

Related reference

“Installation parameters”

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

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 typically 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
```


Related tasks

“Installing stand-alone WebSphere eXtreme Scale” on page 37

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

“Uninstalling WebSphere eXtreme Scale” on page 71

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

“Integrating WebSphere eXtreme Scale with WebSphere Application Server” on page 39

You can install WebSphere eXtreme Scale 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.

“Creating and augmenting profiles for WebSphere eXtreme Scale” on page 53

After you install the product, create unique types of profiles and augment existing profiles for WebSphere eXtreme Scale.

“Installing WebSphere eXtreme Scale silently” on page 64

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.

Chapter 4, “Installing and deploying WebSphere eXtreme Scale,” on page 35

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.

“Silently installing a CIP or an IIP” on page 48

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.

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 eXtreme Scale servers” on page 280 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, refer to 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.

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 need to use a "custom" ORB, the ORB supplied with the IBM SDK is a good choice, although you will need to perform some configuration, as described here. ORBs from other vendors can be used, also with configuration.

Before you begin

Decide whether you will use the ORB provided with WebSphere eXtreme Scale or WebSphere Application Server, the ORB provided with the IBM SDK, or a third party ORB.

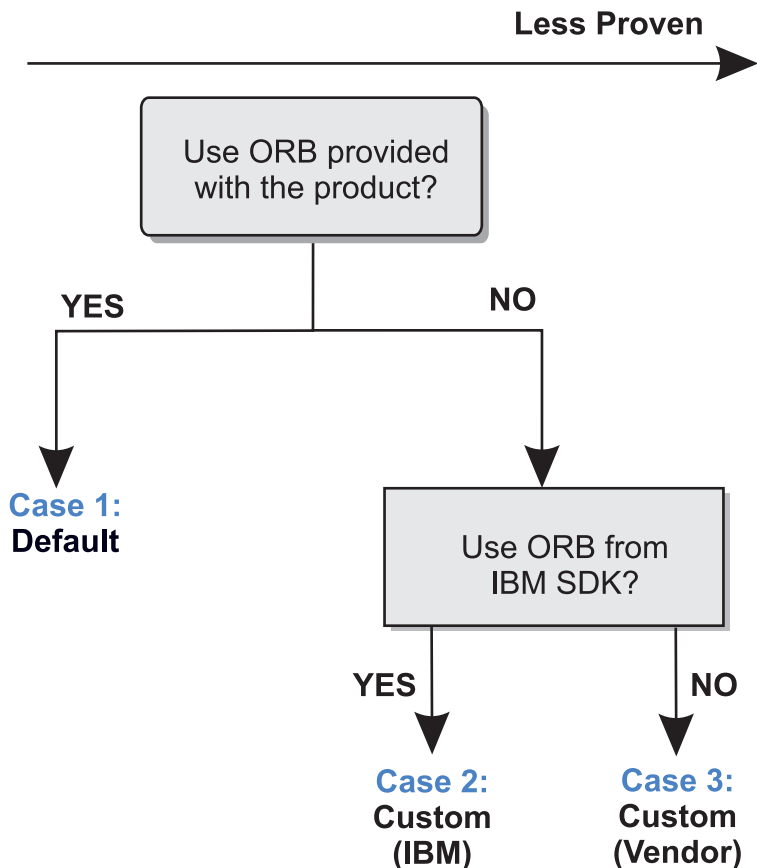


Figure 1. Choosing an ORB

You can make separate decisions for the WebSphere eXtreme Scale server processes and WebSphere eXtreme Scale client processes. While eXtreme Scale supports developer kits from most vendors, it is recommended you use the ORB that is supplied with eXtreme Scale for both your server and client processes. eXtreme Scale does not support the ORB that is supplied with 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 (Third party vendor)

Using a third party 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`. You also might need to edit the following properties in the `orb.properties` file:

```
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 JAR files into an empty directory, hereafter referred to as the *custom_ORB_directory*.
 - `ibmorb.jar`
 - `ibmorbapi.jar`

Tip: If using a custom ORB from a third party vendor (**Case 3**), these additional JAR files might be required:

- `ibmext.jar`
- `ibmcfw.jar`, if using the NIO ORB

2. Specify the *custom_ORB_directory* as an endorsed directory in the scripts that start the Java command.

Tip: If your Java commands already refer to an endorsed directory, another option is to place the *custom_ORB_directory* under the existing endorsed directory – then you would not need to update the scripts. If you decide to update the scripts anyway, be sure to prepend the *custom_ORB_directory* 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 refer to 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 file” on page 239 the information on the ORB properties file in the *Administration Guide*.
 4. Ensure 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.

Related reference

“ORB properties file” on page 239

The `orb.properties` file is used to pass the properties that are used by the Object Request Broker (ORB) to modify the transport behavior of the data grid.

 Object Request Broker custom properties

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. Otherwise, the uninstallation fails.
 - 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. Run the following script to uninstall the product:
 - `UNIX` `Linux` `install_root/uninstall_wxs/uninstall.`
 - `Windows` `install_root\uninstall_wxs\uninstall.exe`

Results

You removed eXtreme Scale from your environment.

Related reference

“Installation parameters” on page 65

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

Chapter 5. Customizing Guide

Using the WebSphere Customization Tools, you can generate and run customized jobs to customize WebSphere eXtreme Scale for z/OS.

Before you begin

- Verify 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 27 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 3 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 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 74
- “Uploading and running customized jobs” on page 75

Installing the WebSphere Customization Tools

Install the WebSphere Customization Tools Version 7.0.0.3 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/V7R0/WCT` directory on your z/OS system.

2. Download and install the WebSphere Customization Tools Version 7.0.0.3 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_7.0.0.0\ 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_7.0.0.0\ 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 73 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 the purpose of 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 new location of the customization definition that you want to create. On the **Customization Locations** tab, click **Add**. If you create a new 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 74 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 6. Configuring WebSphere eXtreme Scale

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.

Related tasks

“Configuring a local eXtreme Scale configuration”

A local in-memory eXtreme Scale configuration can be created by using an ObjectGrid descriptor XML file or eXtreme Scale APIs.

Configuring a local eXtreme Scale configuration

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.

Related tasks

Chapter 6, “Configuring WebSphere eXtreme Scale,” on page 77

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.

ObjectGrid interface

The following methods allow you to interact with an ObjectGrid instance.

Introduction

Create and initialize

See the ObjectGridManager interface topic for the required steps for creating an ObjectGrid instance. Two distinct methods exist to create an ObjectGrid instance: programmatically or with XML configuration files. See the API documentation for more information.

Get or set and factory methods

Any set methods must be called before you initialize the ObjectGrid instance. If you call a set method after the initialize method is called, a `java.lang.IllegalStateException` exception results. Each of the `getSession` methods of the ObjectGrid interface also implicitly call the `initialize` method. Therefore, you must call the set methods before calling any of the `getSession` methods. The only exception to this rule is with the setting, adding, and removing of `ObjectGridEventListener` objects. These objects are allowed to be processed after the initialization processing has completed.

The ObjectGrid interface contains the following major methods.

Table 5. ObjectGrid interface methods

Method	Description
<code>BackingMap defineMap(String name);</code>	<code>defineMap</code> : is a factory method to define a uniquely named <code>BackingMap</code> . For more information about backing maps, see <code>BackingMap</code> interface.
<code>BackingMap getMap(String name);</code>	<code>getMap</code> : Returns a <code>BackingMap</code> previously defined by calling <code>defineMap</code> . By using this method, you can configure the <code>BackingMap</code> , if it is not already configured through XML configuration.

Table 5. ObjectGrid interface methods (continued)

Method	Description
BackingMap createMap(String name);	createMap: Creates a BackingMap, but does not cache it for use by this ObjectGrid. Use this method with the setMaps(List) method of the ObjectGrid interface, which caches BackingMaps for use with this ObjectGrid. Use these methods when you are configuring an ObjectGrid with the Spring Framework.
void setMaps(List mapList);	setMaps: Clears any BackingMaps that have been previously defined on this ObjectGrid and replaces them with the list of BackingMaps that is provided.
public Session getSession() throws ObjectGridException, TransactionCallbackException;	getSession: Returns a Session, which provides begin, commit, rollback functionality for a Unit of Work. For more information about Session objects, see Session interface.
Session getSession(CredentialGenerator cg);	getSession(CredentialGenerator cg): Get a session with a CredentialGenerator object. This method can only be called by the ObjectGrid client in a client server environment.
Session getSession(Subject subject);	getSession(Subject subject): Allows the use of a specific Subject object rather than the one configured on the ObjectGrid to get a Session.
void initialize() throws ObjectGridException;	initialize: ObjectGrid is initialized and available for general use. This method is called implicitly when the getSession method is called, if the ObjectGrid is not in an initialized state.
void destroy();	destroy: The framework is disassembled and cannot be used after this method is called.
void setTxTimeout(int timeout);	setTxTimeout: Use this method to set the amount of time, in seconds, that a transaction that is started by a session that this ObjectGrid instance created is allowed for completion. If a transaction does not complete within the specified amount of time, the Session that started the transaction is marked as being "timed out". Marking a Session as timed out causes the next ObjectMap method that is invoked by the timed out Session to result in a exception. The Session is marked as rollback only, which causes the transaction to be rolled back even if the application calls the commit method instead of the rollback method after the TransactionTimeoutException exception is caught by the application. A timeout value of 0 indicates that the transaction is allowed unlimited amount of time to complete. The transaction does not time out if a time out value of 0 is used. If this method is not called, then any Session that is returned by the getSession method of this interface has a transaction timeout value set to 0 by default. An application can override the transaction timeout setting on a per Session basis by using the setTransactionTimeout method of the com.ibm.websphere.objectgrid.Session interface. You can also configure transaction timeout with the objectGrid.xml file in the distributed case.
int getTxTimeout();	getTxTimeout: Returns the transaction timeout value in seconds. This method returns the same value that is passed as the timeout parameter on the setTxTimeout method. If the setTxTimeout method was not called, then the method returns 0 to indicate that the transaction is allowed an unlimited amount of time to complete.
public int getObjectGridType();	Returns the type of ObjectGrid. The return value is equivalent to one of the constants declared on this interface: LOCAL, SERVER, or CLIENT.
public void registerEntities(Class[] entities);	Register one or more entities based on the class metadata. Entity registration is required prior to ObjectGrid initialization to bind an Entity with a BackingMap and any defined indices. This method may be called multiple times.
public void registerEntities(URL entityXML);	Registers one ore more entities from an entity XML file. Entity registration is required prior to ObjectGrid initialization to bind an Entity with a BackingMap and any defined indices. This method may be called multiple times.
void setQueryConfig(QueryConfig queryConfig);	Set the QueryConfig object for this ObjectGrid. A QueryConfig object provides query configurations for executing object queries over the maps in this ObjectGrid.
//Keywords	
void associateKeyword(Serializable parent, Serializable child);	Deprecated: Use Index or query function to get Objects with specific attributes. The associateKeyword method provides a flexible invalidation mechanism based on keywords. This method links the two keywords together in a directional relationship. If parent is invalidated, then the child is also invalidated. Invalidating the child has no impact on the parent.
//Security	
void setSecurityEnabled()	setSecurityEnabled: Enables security. Security is disabled by default.

Table 5. ObjectGrid interface methods (continued)

Method	Description
void setPermissionCheckPeriod(long period);	setPermissionCheckPeriod: This method takes a single parameter that indicates how often to check the permission that is used to allow a client access. If the parameter is 0, all methods ask the authorization mechanism, either JAAS authorization or custom authorization, to check if the current subject has permission. This strategy might cause performance issues depending on the authorization implementation. However, this type of authorization is available if it is required. Alternatively, if the parameter is less than 0, it indicates the number of milliseconds to cache a set of permissions before returning to the authorization mechanism to refresh them. This parameter provides much better performance, but if the backend permissions are changed during this time the ObjectGrid might allow or prevent access even though the backend security provider has been modified.
void setAuthorizationMechanism(int authMechanism);	setAuthorizationMechanism: Set the authorization mechanism. The default is SecurityConstants.JAAS_AUTHORIZATION.
setMapAuthorization(MapAuthorization ma);	Deprecated: Use the setObjectGridAuthorization (ObjectGridAuthorization) method instead to plug in custom authorizations. setMapAuthorization: Sets the MapAuthorization plug-in for this ObjectGrid instance. This plug-in can be used to authorize ObjectMap or JavaMap accesses to the principals that are contained in the Subject object. A typical implementation of this plug-in is to retrieve the principals from the Subject object, and then check if the specified permissions are granted to the principals.
setSubjectSource(SubjectSource ss);	setSubjectSource: Sets the SubjectSource plugin. This plug-in can be used to get a Subject object that represents the ObjectGrid client. This subject is used for ObjectGrid authorization. The SubjectSource.getSubject method is called by the ObjectGrid runtime when the ObjectGrid.getSession method is used to get a session and the security is enabled. This plug-in is useful for an already authenticated client: it can retrieve the authenticated Subject object and then pass to the ObjectGrid instance. Another authentication is not necessary.
setSubjectValidation(SubjectValidation sv);	setSubjectValidation: Sets the SubjectValidation plugin for this ObjectGrid instance. This plug-in can be used to validate that a javax.security.auth.Subject subject that is passed to the ObjectGrid is a valid subject that has not been tampered with. An implementation of this plug-in needs support from the Subject object creator, because only the creator knows if the Subject object has been tampered with. However, a subject creator might not know if the Subject has been tampered with. In this case, this plug-in should not be used.
void setObjectGridAuthorization (ObjectGridAuthorization ogAuthorization);	Sets the ObjectGridAuthorization for this ObjectGrid instance. Passing null to this method removes a previously set ObjectGridAuthorization object from an earlier invocation of this method and indicates that this <code>ObjectGrid</code> is not associated with a ObjectGridAuthorization object. This method should only be used when ObjectGrid security is enabled. If the ObjectGrid security is disabled, the provided ObjectGridAuthorization object will not be used. A ObjectGridAuthorization plugin can be used to authorize access to the ObjectGrid and maps. As of XD 6.1, the setMapAuthorization is deprecated and setObjectGridAuthorization is recommended for use. However, if both MapAuthorization plugin and ObjectGridAuthorization plugin are used, ObjectGrid will use the provided MapAuthorization to authorize map accesses, even though it is deprecated.

ObjectGrid interface: plug-ins

The ObjectGrid interface has several optional plug-in points for more extensible interactions.

```
void addEventListener(ObjectGridEventListener cb);
void setEventListeners(List cbList);
void removeEventListener(ObjectGridEventListener cb);
void setTransactionCallback(TransactionCallback callback);
int reserveSlot(String);
// Security related plug-ins
void setSubjectValidation(SubjectValidation subjectValidation);
void setSubjectSource(SubjectSource source);
void setMapAuthorization(MapAuthorization mapAuthorization);
```

- **ObjectGridEventListener:** An ObjectGridEventListener interface is used to receive notifications when significant events occur on the ObjectGrid. These events include ObjectGrid initialization, beginning of a transaction, ending a transaction, and destroying an ObjectGrid. To listen for these events, create a

class that implements the `ObjectGridEventListener` interface and add it to the `ObjectGrid`. These listeners are associated with each `Session`. See `Listeners and Session interface` for more information.

- `TransactionCallback`: A `TransactionCallback` listener interface allows transactional events such as `begin`, `commit` and `rollback` signals to send to this interface. Typically, a `TransactionCallback` listener interface is used with a `Loader`. For more information, see `TransactionCallback plug-in and Loaders`. These events can then be used to coordinate transactions with an external resource or within multiple loaders.
- `reserveSlot`: Allows plug-ins on this `ObjectGrid` to reserve slots for use in object instances that have slots like `TxID`.
- `SubjectValidation`. If security is enabled, this plug-in can be used to validate a `javax.security.auth.Subject` class that is passed to the `ObjectGrid`.
- `MapAuthorization`. If security is enabled, this plug-in can be used to authorize `ObjectMap` accesses to the principals that are represented by the `Subject` object.
- `SubjectSource`: If security is enabled, this plug-in can be used to get a `Subject` object that represents the `ObjectGrid` client. This subject is then used for `ObjectGrid` authorization.

For more information about plug-ins, see the introduction to plug-ins in the *Programming Guide*.

BackingMap interface

Each `ObjectGrid` instance contains a collection of `BackingMap` objects. Use the `defineMap` method or the `createMap` method of the `ObjectGrid` interface to name and add each `BackingMap` to an `ObjectGrid` instance. These methods return a `BackingMap` instance that is then used to define the behavior of an individual `Map`.

Session interface

The `Session` interface is used to begin a transaction and to obtain the `ObjectMap` or `JavaMap` that is required for performing transactional interaction between an application and a `BackingMap` object. However, the transaction changes are not applied to the `BackingMap` object until the transaction is committed. A `BackingMap` can be considered as an in-memory cache of committed data for an individual map. For more information, see the information about using `Sessions` to access data in the *Programming Guide*.

The `BackingMap` interface provides methods for setting `BackingMap` attributes. Some of the set methods allow extensibility of a `BackingMap` through several custom designed plug-ins. See the following list of the set methods for setting attributes and providing custom designed plug-in support:

```
// For setting BackingMap attributes.
public void setReadOnly(boolean readOnlyEnabled);
public void setNullValuesSupported(boolean nullValuesSupported);
public void setLockStrategy( LockStrategy lockStrategy );
public void setCopyMode(CopyMode mode, Class valueInterface);
public void setCopyKey(boolean b);
public void setNumberOfBuckets(int numBuckets);
public void setNumberOfLockBuckets(int numBuckets);
public void setLockTimeout(int seconds);
public void setTimeToLive(int seconds);
public void setTtlEvictorType(TTLType type);
public void setEvictionTriggers(String evictionTriggers);
```



```

// For setting an optional custom plug-in provided by application.
public abstract void setObjectTransformer(ObjectTransformer t);
public abstract void setOptimisticCallback(OptimisticCallback checker);
public abstract void setLoader(Loader loader);
public abstract void setPreloadMode(boolean async);
public abstract void setEvictor(Evictor e);
public void setMapEventListeners( List /*MapEventListener*/ eventListenerList );
public void addMapEventListener(MapEventListener eventListener );
public void removeMapEventListener(MapEventListener eventListener );
public void addMapIndexPlugin(MapIndexPlugin index);
public void setMapIndexPlugins(List /* MapIndexPlugin */ indexList );
public void createDynamicIndex(String name, boolean isRangeIndex,
String attributeName, DynamicIndexCallback cb);
public void createDynamicIndex(MapIndexPlugin index, DynamicIndexCallback cb);
public void removeDynamicIndex(String name);

```

A corresponding get method exists for each of the set methods listed.

BackingMap attributes

Each BackingMap has the following attributes that can be set to modify or control the BackingMap behavior:

- **ReadOnly:** This attribute indicates if the Map is a read-only Map or a read and write Map. If this attribute is never set for the Map, then the Map is defaulted to be a read and write Map. When a BackingMap is set to be read only, ObjectGrid optimizes performance for read only when possible.
- **NullValuesSupported:** This attribute indicates if a null value can be put into the Map. If this attribute is never set, the Map does not support null values. If null values are supported by the Map, a get operation that returns null can mean that either the value is null or the map does not contain the key specified by the get operation.
- **LockStrategy:** This attribute determines if a lock manager is used by this BackingMap. If a lock manager is used, then the LockStrategy attribute is used to indicate whether an optimistic locking or pessimistic locking approach is used for locking the map entries. If this attribute is not set, then the optimistic LockStrategy is used. See the Locking topic for details on the supported lock strategies.
- **CopyMode:** This attribute determines if a copy of a value object is made by the BackingMap when a value is read from the map or is put into the BackingMap during the commit cycle of a transaction. Various copy modes are supported to allow the application to make the trade-off between performance and data integrity. If this attribute is not set, then the COPY_ON_READ_AND_COMMIT copy mode is used. This copy mode does not have the best performance, but it has the greatest protection against data integrity problems. If the BackingMap is associated with an EntityManager API entity, then the CopyMode setting has no effect unless the value is set to COPY_TO_BYTES. If any other CopyMode is set, it is always set to NO_COPY. To override the CopyMode for an entity map, use the ObjectMap.setCopyMode method. For more information about the copy modes, see the information about CopyMode method best practices in the *Programming Guide*.
- **CopyKey:** This attribute determines if the BackingMap makes a copy of a key object when an entry is first created in the map. The default action is to not make a copy of key objects because keys are normally unchangeable objects.
- **NumberOfBuckets:** This attribute indicates the number of hash buckets to be used by the BackingMap. The BackingMap implementation uses a hash map for its implementation. If a lot of entries exist in the BackingMap, then more buckets means better performance. The number of keys that have the same bucket

becomes lower as the number of buckets grows. More buckets also mean more concurrency. This attribute is useful for fine tuning performance. An undefined default value is used if the application does not set the `NumberOfBuckets` attribute.

- `NumberOfLockBuckets`: This attribute indicates the number of lock buckets that are be used by the lock manager for this `BackingMap`. When the `LockStrategy` is set to `OPTIMISTIC` or `PESSIMISTIC`, a lock manager is created for the `BackingMap`. The lock manager uses a hash map to keep track of entries that are locked by one or more transactions. If a lot of entries exist in the hash map, more lock buckets lead to better performance because the number of keys that collide on the same bucket is lower as the number of buckets grows. More lock buckets also means more concurrency. When the `LockStrategy` attribute is set to `NONE`, no lock manager is used by this `BackingMap`. In this case, setting `numberOfLockBuckets` has no effect. If this attribute is not set, an undefined value of is used.
- `LockTimeout`: This attribute is used when the `BackingMap` is using a lock manager. The `BackingMap` uses a lock manager when the `LockStrategy` attribute is set to either `OPTIMISTIC` or `PESSIMISTIC`. The attribute value is in seconds and determines how long the lock manager waits for a lock to be granted. If this attribute is not set, then 15 seconds is used as the `LockTimeout` value. See `Pessimistic locking` for details regarding the lock wait timeout exceptions that can occur.
- `TtlEvictorType`: Every `BackingMap` has its own built in time to live evictor that uses a time-based algorithm to determine which map entries to evict. By default, the built in time to live evictor is not active. You can activate the time to live evictor by calling the `setTtlEvictorType` method with one of three values: `CREATION_TIME`, `LAST_ACCESS_TIME`, or `NONE`. A value of `CREATION_TIME` indicates that the evictor adds the `TimeToLive` attribute to the time that the map entry was created in the `BackingMap` to determine when the evictor should evict the map entry from the `BackingMap`. A value of `LAST_ACCESS_TIME` indicates that the evictor adds the `TimeToLive` attribute to the time that the map entry was last accessed by some transaction that the application is running to determine when evictor should evict the map entry. The map entry is evicted only if a map entry is never accessed by any transaction for a period of time that is specified by the `TimeToLive` attribute. A value of `NONE` indicates the evictor should remain inactive and never evict any of the map entries. If this attribute is never set, then `NONE` is used as the default and the time to live evictor is not active. See `Evictors` for details regarding the built-in time to live evictor.
- `TimeToLive`: This attribute is used to specify the number of seconds that the built in time to live evictor needs to add to the creation or last access time for each entry as described for the `TtlEvictorType` attribute. If this attribute is never set, then the special value of zero is used to indicate the time to live is infinity. If this attribute is set to infinity, map entries are never evicted by the evictor.

The following example demonstrates how to define the someMap `BackingMap` in the someGrid `ObjectGrid` instance and set various attributes of the `BackingMap` by using the set methods of the `BackingMap` interface:

```
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.LockStrategy;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;

...

ObjectGrid og =
```

```

ObjectGridManagerFactory.getObjectGridManager().createObjectGrid("someGrid");
BackingMap bm = objectGrid.getMap("someMap");
bm.setReadOnly( true ); // override default of read/write
bm.setNullValuesSupported(false); // override default of allowing Null values
bm.setLockStrategy( LockStrategy.PESSIMISTIC ); // override default of OPTIMISTIC
bm.setLockTimeout( 60 ); // override default of 15 seconds.
bm.setNumberOfBuckets(251); // override default (prime numbers work best)
bm.setNumberOfLockBuckets(251); // override default (prime numbers work best)

```

BackingMap plug-ins

The BackingMap interface has several optional plug points for more extensible interactions with the ObjectGrid:

- **ObjectTransformer plug-in:** For some map operations, a BackingMap might need to serialize, deserialize, or copy a key or value of an entry in the BackingMap. The BackingMap can perform these actions by providing a default implementation of the ObjectTransformer interface. An application can improve performance by providing a custom designed ObjectTransformer plug-in that is used by the BackingMap to serialize, deserialize, or copy a key or value of an entry in the BackingMap. See the information about the ObjectTransformer plug-in in the *Programming Guide* for more information.
- **Evictor plug-in:** The built in time to live evictor uses a time-based algorithm to decide when an entry in BackingMap must be evicted. Some applications might need to use a different algorithm for deciding when an entry in a BackingMap needs to be evicted. The Evictor plug-in makes a custom designed Evictor available to the BackingMap to use. The Evictor plug-in is in addition to the built in time to live evictor. It does not replace the time to live evictor. ObjectGrid provides a custom Evictor plug-in that implements well-known algorithms such as "least recently used" or "least frequently used". Applications can either plug-in one of the provided Evictor plug-ins or it can provide its own Evictor plug-in. See the information about eviction in the *Programming Guide*.
- **MapEventListener plug-in:** An application might want to know about BackingMap events such as a map entry eviction or a preload of a BackingMap completion. A BackingMap calls methods on the MapEventListener plug-in to notify an application of BackingMap events. An application can receive notification of various BackingMap events by using the setMapEventListener method to provide one or more custom designed MapEventListener plug-ins to the BackingMap. The application can modify the listed MapEventListener objects by using the addMapEventListener method or the removeMapEventListener method. See the information about the MapEventListener plug-in in the *Programming Guide* for more information.
- **Loader plug-in:** A BackingMap is an in-memory cache of a Map. A Loader plug-in is an option that is used by the BackingMap to move data between memory and is used for a persistent store for the BackingMap. For example, a Java database connectivity (JDBC) Loader can be used to move data in and out of a BackingMap and one or more relational tables of a relational database. A relational database does not need to be used as the persistent store for a BackingMap. The Loader can also be used to moved data between a BackingMap and a file, between a BackingMap and a Hibernate map, between a BackingMap and a Java 2 Platform, Enterprise Edition (JEE) entity bean, between a BackingMap and another application server, and so on. The application must provide a custom-designed Loader plug-in to move data between the BackingMap and the persistent store for every technology that is used. If a Loader is not provided, the BackingMap becomes a simple in-memory cache. See the information about using a Loader in the *Programming Guide* for more information.

- **OptimisticCallback plug-in:** When the LockStrategy attribute for a BackingMap is set to OPTIMISTIC, either the BackingMap or a Loader plug-in must perform comparison operations for the values of the map. The OptimisticCallback plug-in is used by the BackingMap and the Loader to perform the optimistic versioning comparison operations. See the information about the OptimisticCallback plug-in in the *Programming Guide* for more information.
- **MapIndexPlugin plug-in:** A MapIndexPlugin plug-in, or an Index in short, is an option that is used by the BackingMap to build an index that is based on the specified attribute of the stored object. The index allows the application to find objects by a specific value or a range of values. There are two types of index: static and dynamic. Refer to Indexing for detailed information.

For more information regarding plug-ins, see the introduction to plug-ins in the *Programming Guide*.

Map entry locking

An ObjectGrid BackingMap supports several locking strategies for maps to maintain cache entry consistency.

Each BackingMap 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*.

Specifying a lock strategy

The following example demonstrates how the lock strategy can be set on the map1, map2, and map3 BackingMaps, where each map is using a different locking strategy. The first snippet shows how to use XML for lock strategy configuration and the second snippet shows a programmatic approach.

XML approach

BackingMap 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="test">
      <backingMap name="map1"
        lockStrategy="PESSIMISTIC" numberOfLockBuckets="31"/>
      <backingMap name="map2"
        lockStrategy="OPTIMISTIC" numberOfLockBuckets="409"/>
      <backingMap name="map3"
        lockStrategy="NONE"/>
    </objectGrid>
  </objectGrids>
</objectGridConfig>
```

Programmatic approach

BackingMap configuration - programmatic example

```
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("map1");
bm.setLockStrategy( LockStrategy.PESSIMISTIC );
bm.setNumberOfLockBuckets(31);
bm = og.defineMap("map2");
bm.setNumberOfLockBuckets(409);
bm.setLockStrategy( LockStrategy.OPTIMISTIC );
bm = og.defineMap("map3");
bm.setLockStrategy( LockStrategy.NONE );
```

To avoid a `java.lang.IllegalStateException` exception, the `setLockStrategy` method must be called before using the `initialize` or `getSession` methods on a local `ObjectGrid` instance.

For more information see the topic on locking strategies in the *Product Overview*.

Lock manager configuration

When either a `PESSIMISTIC` or an `OPTIMISTIC` lock strategy is used, a lock manager is created for the `BackingMap`. The lock manager uses a hash map to track entries that are locked by one or more transactions. If many map entries exist in the hash map, more lock buckets can result in better performance. The risk of Java synchronization collisions is lower as the number of buckets grows. More lock buckets also lead to more concurrency. The previous examples show how an application can set the number of lock buckets to use for a given `BackingMap` instance.

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

LockDeadlockException

Following is a code example that shows catching the exception, and the resulting message is then displayed.

```
try {  
    ...  
} catch (ObjectGridException oe) {  
    System.out.println(oe);  
}
```

The result is:

```
com.ibm.websphere.objectgrid.plugins.LockDeadlockException: _Message
```

This message represents the string that is passed as a parameter when the exception is created and thrown.

Exception cause

The most common type of deadlock exception happens when you use the pessimistic lock strategy, and two separate clients each own a shared lock on a particular object. Then, both clients attempt to promote to an exclusive lock on that object. The following diagram illustrates such a situation, including transaction blocks that cause the exception to be thrown.

The following Java code snippet demonstrates how to pass in an XML configuration file to create an ObjectGrid.

This is an abstract view of what is occurring in your program when the exception occurs. In an application with many threads updating the same ObjectMap, it is possible to encounter this situation. The following is an example of two clients executing the transaction code blocks, as illustrated in the previous figure.

Possible solutions

You can possibly encounter the situation illustrated in Figure 1 when numerous threads start transactions on a particular map. In this case, the exception is thrown to keep your program from hanging. You can notify yourself, and add code to the catch block for more details on the cause. Since you only see this exception in a pessimistic locking strategy, one simple solution is to simply use an optimistic locking strategy. If you require a pessimistic locking strategy, however, you can use the `getForUpdate` method instead of the `get` method. This eliminates receiving the exceptions for the situation described previously.

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

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

- specify optimistic 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("optimisticMap");
bm.setLockStrategy( LockStrategy.OPTIMISTIC );
```

- Using XML

- specify optimistic 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="optimisticMap"
                lockStrategy="OPTIMISTIC"/>
        </objectGrid>
    </objectGrids>
</objectGridConfig>
```

- **Configure a pessimistic locking strategy**

- **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 );
```

- **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

- specify a no-locking 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("noLockingMap");
bm.setLockStrategy( LockStrategy.NONE);

```

– Using XML

specify a no-locking strategy with XML

```

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

    <objectGrids>
        <objectGrid name="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.

Related concepts

“WebSphere eXtreme Scale client configuration” on page 92

You can configure an eXtreme Scale client based on your requirements such as the need to override settings.

“Integrating with Spring framework” on page 242

Spring is a popular framework for developing Java applications. WebSphere eXtreme Scale provides support to allow Spring to manage eXtreme Scale transactions and configure the clients and servers comprising your deployed in-memory data grid.

Distributed eXtreme Scale grid configuration

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 eXtreme Scale container. The XML file specifies the following information:

- The maps belonging to each map set
- The number of partitions
- The number of synchronous and asynchronous replicas

For information on starting container servers, read about starting eXtreme Scale containers automatically or starting container processes.

The deployment policy also controls the following placement behaviors.

- The minimum number of active containers 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, read about the deployment policy descriptor XML file.

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 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 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 an eXtreme Scale container 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>

```

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. Since maxSyncReplicas 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 eXtreme Scale environment. Read about the deployment policy descriptor XML file.

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 embedding a container in an existing Java process or application. For more information, see the eXtreme Scale `DeploymentPolicy` API documentation.

Related reference

“startOgServer script” on page 270

The startOgServer script starts servers. You can use a variety of parameters when you start your servers to enable trace, specify port numbers, and so on.

“stopOgServer script” on page 281

The stopOgServer script stops catalog and container servers.

“Deployment policy descriptor XML file” on page 190

To configure a deployment policy, use a deployment policy descriptor XML file.

“ObjectGrid descriptor XML file” on page 196

To configure WebSphere eXtreme Scale, use an ObjectGrid descriptor XML file and the ObjectGrid API.

WebSphere eXtreme Scale client configuration

You can configure an eXtreme Scale client based on your requirements such as the need to override settings.

You can configure an eXtreme Scale client in the following ways:

- 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

Configure the client with XML

An ObjectGrid XML file can be used to alter settings on the client side. To change the settings on an eXtreme Scale client, you must create an ObjectGrid XML file that is similar in structure to the file that was used for the eXtreme Scale server.

Assume that the following XML file was paired with a deployment policy XML file, and these files were used to start an 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" />
    </objectGrid>
  </objectGrids>
</objectGridConfig>
```

```

        <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 an 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 `companyGridClientSide.xml` file overrides several attributes and plug-ins on the `CompanyGrid` client as follows:

- 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 ObjectGridManager instance ogManager 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>
  </bean>

```

```

</list>
</property>
<property name="backingMapConfigurations">
  <list>
    <bean class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
      factory-method="createBackingMapConfiguration">
      <constructor-arg type="java.lang.String" value="Customer" />
      <property name="plugins">
        <bean class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
          factory-method="createPlugin">
            <constructor-arg type="com.ibm.websphere.objectgrid.config.PluginType"
              value="EVICTOR" />
            <constructor-arg type="java.lang.String"
              value="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" />
          </bean>
        </property>
        <property name="numberOfBuckets" value="1429" />
      </bean>
    <bean class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
      factory-method="createBackingMapConfiguration">
      <constructor-arg type="java.lang.String" value="OrderLine" />
      <property name="numberOfBuckets" value="701" />
    </bean>
  </list>
</property>
<property name="timeToLive" value="800" />
<property name="ttlEvictorType">
  <value type="com.ibm.websphere.objectgrid.
    TTLType">LAST_ACCESS_TIME</value>
</property>
</bean>
</list>
</property>
</bean>

  <bean id="client" factory-bean="manager" factory-method="connect"
    singleton="true">
    <constructor-arg type="java.lang.String">
      <value>localhost:2809</value>
    </constructor-arg>
    <constructor-arg
      type="com.ibm.websphere.objectgrid.security.
        config.ClientSecurityConfiguration">
      <null />
    </constructor-arg>
    <constructor-arg type="java.net.URL">
      <null />
    </constructor-arg>
  </bean>
</beans>

```

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 integrating with the Spring framework for more information.

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.

See the information about map entry locking in the *Administration Guide* for more information.

Related tasks

“Configuring a locking strategy” on page 87

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

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 class, `com.ibm.websphere.objectgrid.plugins.builtins.JMSObjectGridEventListener`, to enable the client invalidation mechanism within a distributed eXtreme Scale environment that is known as an eXtreme Scale grid.

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 187.

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="" />
      </bean>
    </objectGrid>
  </objectGrids>
</objectGridConfig>
```



```

    <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;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" />
    </objectGrid>
  </objectGrids>
</objectGridConfig>

```



```

<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>

```

Configuring the request retry timeout

With reliable maps, you can supply a retry timeout to WebSphere eXtreme Scale for transaction requests.

There are two ways to configure reliable maps. If the value is set to greater than zero, the request is tried until either the timeout condition is met or a permanent failure such as a `DuplicateKeyException` exception is encountered. A value of zero indicates the fail-fast mode setting and eXtreme Scale does not attempt to retry.

You supply a timeout value in milliseconds on the client properties file or on a session. The session always overrides the client properties setting. During run time, the transaction timeout is used with the retry timeout, ensuring the retry timeout does not exceed the transaction timeout.

Due to variations in how autocommit and non-autocommit transactions (transactions that are using explicit begin and commit methods) are completed, valid exceptions for retry are different.

For transactions that are called within a session, the retry is valid for CORBA `SystemExceptions` and eXtreme Scale `TargetNotAvailable` exceptions.

For autocommit transactions, the retry is valid for CORBA `SystemExceptions` eXtreme Scale `Availability Exceptions` (`ReplicationVotedToRollbackTransactionException`, `TargetNotAvailable`, `AvailabilityException`, and others).

For more information, 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 retry the transaction. These permanent failures include the `DuplicateKeyException` and `KeyNotFoundException` exceptions.

The fail-fast setting returns all exceptions without retrying for any exceptions.

The following lists show the exceptions in more detail:

Exceptions where the client retries

- ReplicationVotedToRollbackTransactionException (only on autocommit)
- TargetNotAvailable
- org.omg.CORBA.SystemException
- AvailabilityException (only on autocommit)
- LockTimeoutException (only on autocommit)
- UnavailableServiceException (only on autocommit)

Other Exceptions

- DuplicateKeyException
- KeyNotFoundException
- LoaderException
- TransactionAffinityException
- LockDeadlockException
- OptimisticCollisionException

Setting the requestRetryTimeout property 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 236. 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.

objectGridClient.properties

```
# eXtreme Scale client config
preferLocalProcess = false
preferLocalhost = false
requestRetryTimeout = 30000
```

The requestRetryTimeout value is specified in milliseconds. In the previous example, if the value is used on an ObjectGrid instance, the requestRetryTimeout value is 30 seconds.

Setting client properties by obtaining an ObjectGrid connection 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 establish a connection with the ObjectGridManager, set the client properties as described. Then, when you have an ObjectGrid instance, it will have 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);
```

Session override example with autocommit

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 grid](#).

Troubleshooting XML configuration

Occasionally, when you configure eXtreme Scale, you can encounter unexpected behavior in XML configuration.

The following sections are various XML configuration problems that can occur.

Mismatched deployment policy and ObjectGrid XML files

The deployment policy and ObjectGrid XML files must match. If they do not have matching ObjectGrid names and map names, errors occur.

Incorrect backingMap and map references

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"/>
    </mapSet>
  </objectgridDeployment>
</deploymentPolicy>
```

```
        <map ref="ledger"/>
    </mapSet>
</objectgridDeployment>
</deploymentPolicy>
```

Messages

An ObjectGridException occurs with a caused by exception of IncompatibleDeploymentPolicyException. For the preceding example, the exception is displayed as the following message:

```
com.ibm.websphere.objectgrid.IncompatibleDeploymentPolicyException: The ledger map reference in the mapSet1 mapSet of accounting objectgridDeployment does not reference a valid backingMap from the ObjectGrid XML.
```

If the deployment policy is missing map references to backingMaps listed in the ObjectGrid XML file, an ObjectGridException also occurs with a caused by exception of IncompatibleDeploymentPolicyException. For example:

```
com.ibm.websphere.objectgrid.IncompatibleDeploymentPolicyException: The accounting objectGrid contains the employee backingMap in the ObjectGrid XML. This map is not referenced in a mapSet within the accounting objectGridDeployment in the deployment policy XML.
```

Problem

The backingMap list in the ObjectGrid XML file and the map references in the deployment policy must match.

Solution

Determine which list is correct for your environment and correct the XML file that contains the incorrect list.

Incorrect ObjectGrid names

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.
```

Problem

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 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".

XML value of attribute is not valid

Some of the attributes in the XML file can only be assigned certain values. These attributes have acceptable values enumerated by the schema. The following list provides some of the attributes:

- authorizationMechanism attribute on the objectGrid element
- copyMode attribute on the backingMap element
- lockStrategy attribute on the backingMap element
- ttlEvictorType attribute on the backingMap element
- type attribute on the property element
- initialState on the objectGrid element
- evictionTriggers on the backingMap element

If one of these attributes is assigned an invalid value, XML validation fails. In the following example XML file, an incorrect value of INVALID_COPY_MODE is used:

INVALID_COPY_MODE example

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

The following message appears in the log.

```
CWOBJ2403E: The XML file is invalid. A problem has been detected with < null >
at line 5. The error message is cvc-enumeration-valid: Value
'INVALID_COPY_MODE' is not facet-valid with respect to enumeration
'[COPY_ON_READ_AND_COMMIT, COPY_ON_READ, COPY_ON_WRITE,
NO_COPY, COPY_TO_BYTES]'. It must be a value from the enumeration.
```

Missing attributes or tags

If an XML file is missing the correct attributes or tags, errors occur. For example, the following 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>
```

The following message appears in the log.

CWOBJ2403E: The XML file is invalid. A problem has been detected with < null > at line 7. The error message is The end-tag for element type "objectGrid" must end with a '>' delimiter.

An ObjectGridException about the invalid XML file occurs with the name of the XML file

Syntax errors

If an XML file is formatted with incorrect or missing syntax, the CWOBJ2403E appears in the log. For example, the following message is displayed when a quote is missing on one of the XML attributes.

CWOBJ2403E: The XML file is invalid. A problem has been detected with < null > at line 7. The error message is Open quote is expected for attribute "maxSyncReplicas" associated with an element type "mapSet".

An ObjectGridException about the invalid XML file also occurs.

Referencing a nonexistent plug-in collection

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 ID of one of the backingMapPluginCollection elements.

Message

If the pluginCollectionRef attribute does not match any ID attributes of any of the backingMapPluginConfiguration elements, the following message, or one that is similar, is displayed in the log.

```
[7/14/05 14:02:01:971 CDT] 686c060e XmlErrorHandl E CWOBJ9002E:
This is an English only Error message: Invalid XML file. Line: 14; URI:
null; Message: Key 'pluginCollectionRef' with
value 'bookPlugins' not found for identity constraint of
element 'objectGridConfig'.
```

Problem

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.

Validating XML without support of an implementation

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. If you want to validate XML by using an SDK that does not contain this implementation, download Apache Xerces, and include its Java archive (JAR) files in the classpath.

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.

To avoid this problem, after you download Xerces and include the JAR files in the classpath, you can validate the XML file successfully.

Related reference

“Deployment policy descriptor XML file” on page 190

To configure a deployment policy, use a deployment policy descriptor XML file.

“objectGrid.xsd file” on page 212

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

“deploymentPolicy.xsd file” on page 194

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

Loaders

With an eXtreme Scale Loader plug-in, an eXtreme Scale map can behave as a memory cache for data that is typically kept in a persistent store on either the same system or another system. Typically, a database or file system is used as the persistent store. A remote Java virtual machine (JVM) can also be used as the

source of data, allowing hub-based caches to be built using eXtreme Scale. A loader has the logic for reading and writing data to and from a persistent store.

Overview

Loaders are backing map plug-ins that are invoked when changes are made to the backing map or when the backing map is unable to satisfy a data request (a cache miss). See the information about caching scenarios in the *Product Overview* for an overview on how eXtreme Scale can interact with a loader.

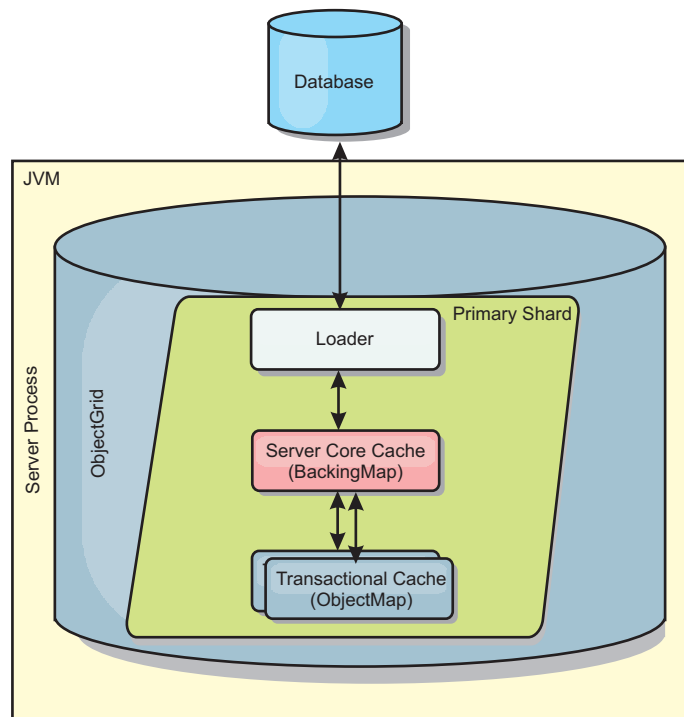


Figure 2. Loader

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

Using a loader

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

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

For more information, see the topic on writing a loader in the *Product Overview*.

XML configuration approach to plug in a Loader

An application-provided Loader can be plugged in by using an XML file. The following example demonstrates how to plug in the "MyLoader" loader into the "map1" backing map. You must specify the className for your loader, the database name and connection details, and the isolation level properties. You can use the same XML structure if you are only using a preloader by specifying the preloader classname instead of a complete loader classname.

Loader configuration with XML

```
<?xml version="1.0" encoding="UTF-8" ?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
  <objectGrid name="grid">
    <backingMap name="map1" pluginCollectionRef="map1" lockStrategy="OPTIMISTIC" />
  </objectGrid>
</objectGrids>
<backingMapPluginCollections>
  <backingMapPluginCollection id="map1">
    <bean id="Loader" className="com.myapplication.MyLoader">
      <property name="dataBaseName"
        type="java.lang.String"
        value="testdb"
        description="database name" />
      <property name="isolationLevel"
        type="java.lang.String"
        value="read committed"
        description="iso level" />
    </bean>
  </backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

Programmatically plug in a Loader

You can only use a programmatic configuration with local, in-memory grids. The following snippet of code demonstrates how to plug an application-provided Loader into the backing map for map1 using the ObjectGrid API:

Programmatic configuration of a Loader

```
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogManager.createObjectGrid( "grid" );
BackingMap bm = og.defineMap( "map1" );
MyLoader loader = new MyLoader();
loader.setDataBaseName("testdb");
loader.setIsolationLevel("read committed");
bm.setLoader( loader );
```

This snippet assumes that the MyLoader class is the application-provided class that implements the com.ibm.websphere.objectgrid.plugins.Loader interface. Because the association of a Loader with a backing map cannot be changed after ObjectGrid is initialized, the code must be run before invoking the initialize method of the ObjectGrid interface that is being called. An IllegalStateException exception occurs on a setLoader method call if it is called after initialization has occurred.

The application-provided Loader can have set properties. In the example, the MyLoader loader is used to read and write data from a table in a relational database. The loader must specify the name of the database and the SQL isolation level. The MyLoader loader has the setDataBaseName and setIsolationLevel methods that allow the application to set these two Loader properties.

Related tasks

“Configuring JPA loaders” on page 124

A Java Persistence API (JPA) Loader is a plug-in implementation that uses JPA to interact with the database.

“Configuring a JPA time-based data updater” on page 126

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.

“Troubleshooting loaders” on page 367

Use this information to troubleshoot issues with your database loaders.

Related reference

“Write-behind dumper class sample code” on page 121

This sample source code shows how to write a watcher (dumper) to handle failed write-behind updates.

Loader configuration

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.

Related tasks

“Configuring JPA loaders” on page 124

A Java Persistence API (JPA) Loader is a plug-in implementation that uses JPA to interact with the database.

“Configuring a JPA time-based data updater” on page 126

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.

“Troubleshooting loaders” on page 367

Use this information to troubleshoot issues with your database loaders.

Related reference

“Write-behind dumper class sample code” on page 121

This sample source code shows how to write a watcher (dumper) to handle failed write-behind updates.

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.

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 6. 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 “BackingMap interface” on page 81.

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 update 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 119 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.

Configuring write-behind 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.

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.

Write-behind dumper class sample code:

This sample source code shows how to write a watcher (dumper) to handle failed write-behind updates.

```
//
//This sample program is provided AS IS and may be used, executed, copied and
//modified without royalty payment by customer (a) for its own instruction and
//study, (b) in order to develop applications designed to run with an IBM
//WebSphere product, either for customer's own internal use or for redistribution
//by customer, as part of such an application, in customer's own products. "
//
//5724-J34 (C) COPYRIGHT International Business Machines Corp. 2009
//All Rights Reserved * Licensed Materials - Property of IBM
//
package utils;

import java.util.Collection;
import java.util.Iterator;
import java.util.concurrent.Callable;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.ScheduledFuture;
import java.util.concurrent.ScheduledThreadPoolExecutor;
import java.util.concurrent.TimeUnit;
import java.util.logging.Logger;

import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridException;
import com.ibm.websphere.objectgrid.ObjectGridRuntimeException;
import com.ibm.websphere.objectgrid.ObjectMap;
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.UndefinedMapException;
import com.ibm.websphere.objectgrid.plugins.ObjectGridEventGroup;
import com.ibm.websphere.objectgrid.plugins.ObjectGridEventListener;
import com.ibm.websphere.objectgrid.writebehind.FailedUpdateElement;
import com.ibm.websphere.objectgrid.writebehind.WriteBehindLoaderConstants;

/**
 * Write behind expects transactions to the Loader to succeed. If a transaction for a key fails then
 * it inserts an entry in a Map called PREFIX + mapName. The application should be checking this
 * map for entries to dump out write behind transaction failures. The application is responsible for
 * analyzing and then removing these entries. These entries can be large as they include the key, before
 * and after images of the value and the exception itself. Exceptions can easily be 20k on their own.
 *
 * The class is registered with the grid and an instance is created per primary shard in a JVM. It creates a single thread
 * and that thread then checks each write behind error map for the shard, prints out the problem and then removes the
 * entry.
 *
 * This means there will be one thread per shard. If the shard is moved to another JVM then the deactivate method stops the
 * thread.
 * @author bnewport
 */
public class WriteBehindDumper implements ObjectGridEventListener, ObjectGridEventGroup.ShardEvents, Callable<Boolean>
{
    static Logger logger = Logger.getLogger(WriteBehindDumper.class.getName());

    ObjectGrid grid;

    /**
     * Thread pool to handle table checkers. If the application has it's own pool
     * then change this to reuse the existing pool
     */
    static ScheduledExecutorService pool = new ScheduledThreadPoolExecutor(2); // two threads to dump records

    // the future for this shard
    ScheduledFuture<Boolean> future;

    // true if this shard is active
    volatile boolean isShardActive;
}
```



```

/**
 * Normal time between checking Maps for write behind errors
 */
final long BLOCKTIME_SECS = 20L;

/**
 * An allocated session for this shard. No point in allocating them again and again
 */
Session session;
/**
 * When a primary shard is activated then schedule the checks to periodically check
 * the write behind error maps and print out any problems
 */
public void shardActivated(ObjectGrid grid)
{
    try
    {
        this.grid = grid;
        session = grid.getSession();

        isShardActive = true;
        future = pool.schedule(this, BLOCKTIME_SECS, TimeUnit.SECONDS); // check every BLOCKTIME_SECS seconds initially
    }
    catch(ObjectGridException e)
    {
        throw new ObjectGridRuntimeException("Exception activating write dumper", e);
    }
}

/**
 * Mark shard as inactive and then cancel the checker
 */
public void shardDeactivate(ObjectGrid arg0)
{
    isShardActive = false;
    // if it's cancelled then cancel returns true
    if(future.cancel(false) == false)
    {
        // otherwise just block until the checker completes
        while(future.isDone() == false) // wait for the task to finish one way or the other
        {
            try
            {
                Thread.sleep(1000L); // check every second
            }
            catch(InterruptedException e)
            {
            }
        }
    }
}

/**
 * Simple test to see if the map has write behind enabled and if so then return
 * the name of the error map for it.
 * @param mapName The map to test
 * @return The name of the write behind error map if it exists otherwise null
 */
static public String getWriteBehindNameIfPossible(ObjectGrid grid, String mapName)
{
    BackingMap map = grid.getMap(mapName);
    if(map != null && map.getWriteBehind() != null)
    {
        return WriteBehindLoaderConstants.WRITE_BEHIND_FAILED_UPDATES_MAP_PREFIX + mapName;
    }
    else
        return null;
}

/**
 * This runs for each shard. It checks if each map has write behind enabled and if it does
 * then it prints out any write behind
 * transaction errors and then removes the record.
 */
public Boolean call()
{
    logger.fine("Called for " + grid.toString());
    try
    {
        // while the primary shard is present in this JVM
        // only user defined maps are returned here, no system maps like write behind maps are in
        // this list.
        Iterator<String> iter = grid.getListOfMapNames().iterator();
        boolean foundErrors = false;
        // iterate over all the current Maps
        while(iter.hasNext() && isShardActive)
        {
            String origName = iter.next();

            // if it's a write behind error map

```



```

String name = getWriteBehindNameIfPossible(grid, origName);
if(name != null)
{
    // try to remove blocks of N errors at a time
    ObjectMap errorMap = null;
    try
    {
        errorMap = session.getMap(name);
    }
    catch(UndefinedMapException e)
    {
        // at startup, the error maps may not exist yet, patience...
        continue;
    }
    // try to dump out up to N records at once
    session.begin();
    for(int counter = 0; counter < 100; ++counter)
    {
        Integer seqKey = (Integer)errorMap.getNextKey(1L);
        if(seqKey != null)
        {
            foundErrors = true;
            FailedUpdateElement elem = (FailedUpdateElement)errorMap.get(seqKey);
            //
            // Your application should log the problem here
            logger.info("WriteBehindDumper ( " + origName + ") for key ( " + elem.getKey() + ") Exception: " +
                elem.getThrowable().toString());
            //
            //
            errorMap.remove(seqKey);
        }
        else
            break;
    }
    session.commit();
}
// do next map
// loop faster if there are errors
if(isShardActive)
{
    // reschedule after one second if there were bad records
    // otherwise, wait 20 seconds.
    if(foundErrors)
        future = pool.schedule(this, 1L, TimeUnit.SECONDS);
    else
        future = pool.schedule(this, BLOCKTIME_SECS, TimeUnit.SECONDS);
}
}
catch(ObjectGridException e)
{
    logger.fine("Exception in WriteBehindDumper" + e.toString());
    e.printStackTrace();

    //don't leave a transaction on the session.
    if(session.isTransactionActive())
    {
        try { session.rollback(); } catch(Exception e2) {}
    }
}
return true;
}

public void destroy() {
    // TODO Auto-generated method stub

}

public void initialize(Session arg0) {
    // TODO Auto-generated method stub

}

public void transactionBegin(String arg0, boolean arg1) {
    // TODO Auto-generated method stub

}

public void transactionEnd(String arg0, boolean arg1, boolean arg2,
    Collection arg3) {
    // TODO Auto-generated method stub

}
}

```

Related concepts

“Loaders” on page 105

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

“Loader configuration” on page 108

Implementing a loader requires configuration for several attributes.

Related tasks

“Configuring JPA loaders”

A Java Persistence API (JPA) Loader is a plug-in implementation that uses JPA to interact with the database.

“Configuring a JPA time-based data updater” on page 126

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.

“Troubleshooting loaders” on page 367

Use this information to troubleshoot issues with your database loaders.

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" />
        </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"/>
        </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. For more details about Spring namespace support, see “Integrating with Spring framework” on page 242.

Related concepts

“Loaders” on page 105

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

“Loader configuration” on page 108

Implementing a loader requires configuration for several attributes.

Loaders overview

Related reference

“Write-behind dumper class sample code” on page 121

This sample source code shows how to write a watcher (dumper) to handle failed write-behind updates.

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.

Related concepts

“Loaders” on page 105

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

“Loader configuration” on page 108

Implementing a loader requires configuration for several attributes.

Related reference

“Write-behind dumper class sample code” on page 121

This sample source code shows how to write a watcher (dumper) to handle failed write-behind updates.

Cache integration

WebSphere eXtreme Scale can integrate with other caching-related products. JPA can be used 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.

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.

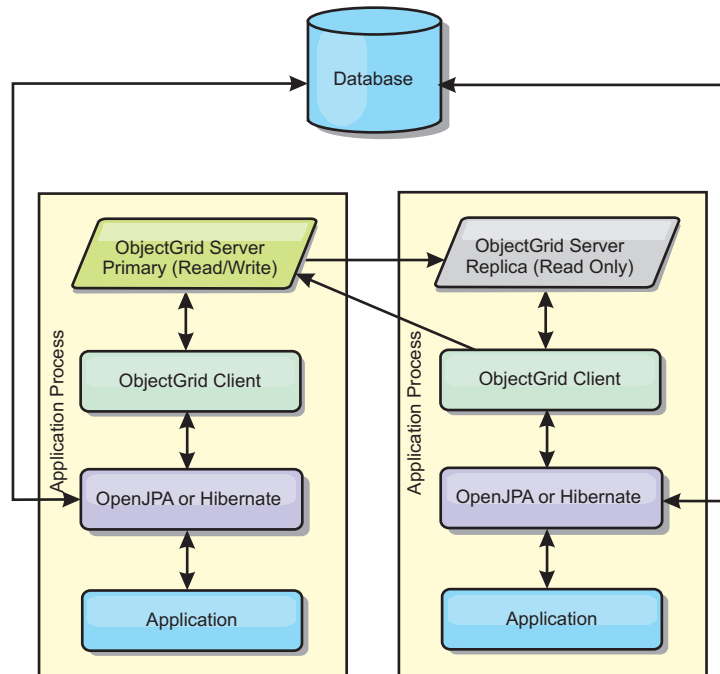


Figure 3. JPA embedded topology

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.

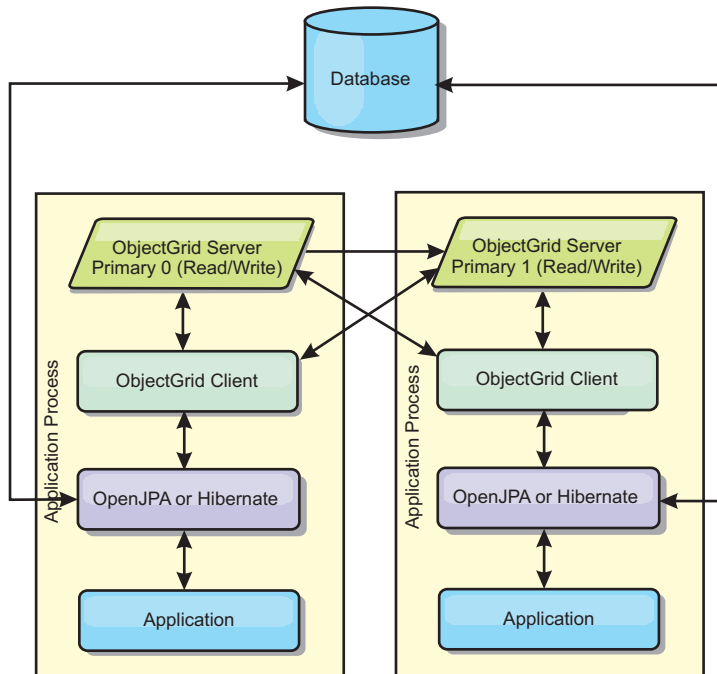


Figure 4. JPA embedded, partitioned topology

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.

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 configure the eXtreme Scale to be partitioned and replicated. A remote configuration is managed independently from the application and JPA provider.

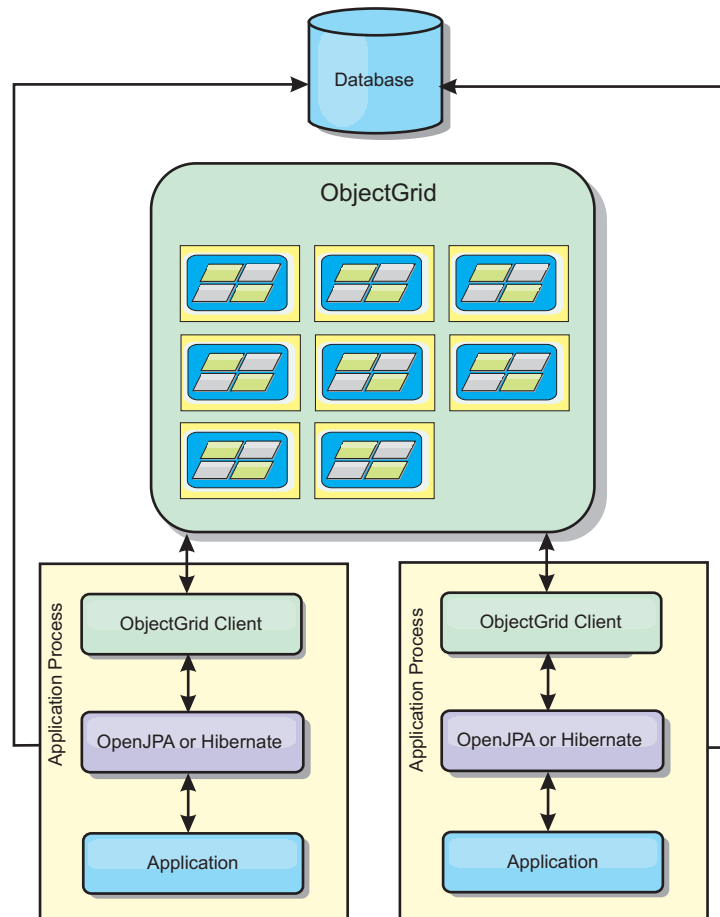


Figure 5. JPA remote topology

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.

JPA cache plug-in configuration

WebSphere eXtreme Scale includes level 2 cache plug-ins for both OpenJPA and Hibernate Java Persistence API (JPA) providers.

ObjectGrid JPA cache configuration properties

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 server grid. 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 grid that is defined for the WebSphere Application Server cell. For more information about defining a catalog service grid, 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 grid 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>
```

Hibernate cache plug-in configuration:

An eXtreme Scale cache can be enabled for Hibernate by setting properties in the configuration file.

7.0.0.0 FIX 4+ For integration with WebSphere Application Server, the hibernate cache plug-in is packaged in `oghibernate-cache.jar` and installed in `WAS_HOME/optionalLibraries/ObjectGrid`. To use the hibernate cache plug-in, you have to include the `oghibernate-cache.jar` 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 into the shared library directory.

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>
```

7.0.0 Fix 4+ Version 7.0 Fix 4 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=EMBEDEDDED,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, the @cache annotation has to be added like the following:

```
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.

- 1. 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.
- 2. 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.
- 3. 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" />
      <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.AnnuityPersistentObject">
    <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>

```

Note:

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 you want to be exclude 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.

HTTP session management

In a WebSphere Application Server Version 5 or later environment, the HTTP session manager that is shipped with WebSphere eXtreme Scale can override the default session manager in the application server.

The WebSphere eXtreme Scale HTTP session manager can also run on WebSphere Application Server Version 6.0.2 or later or in an environment that is not running WebSphere Application Server, such as WebSphere Application Server Community Edition or Apache Tomcat.

Features

The session manager has been designed so that it can run in any Java Platform, Enterprise Edition Version 1.4 container. The session manager does not have any dependencies on WebSphere APIs, so it is capable of supporting various versions of WebSphere Application Server as well as vendor application server environments.

The HTTP session manager provides session management capabilities for an associated application. The session manager creates HTTP sessions and manages the life cycles of HTTP sessions that are associated with the application. These life cycle management activities include: the invalidation of sessions based on a timeout or an explicit servlet or JavaServer Pages (JSP) call and the invocation of session listeners that are associated with the session or the web application. The session manager persists its sessions in an ObjectGrid instance. This instance can be a local, in-memory instance or a fully replicated, clustered and partitioned instance. The use of the latter topology allows the session manager to provide HTTP session failover support when application servers are shut down or end unexpectedly. The session manager can also work in environments that do not support affinity, when affinity is not enforced by a load balancer tier that sprays requests to the application server tier.

Usage scenarios

The session manager can be used in the following scenarios:

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

How the session manager works

The session manager introduces itself into the request path in the form of a servlet filter. You can add this servlet filter to every Web module in your application with tooling that ships with WebSphere eXtreme Scale. You can also manually add these filters to the Web deployment descriptor of your application. This filter receives the request before the servlet or JSP files in the target application. At this time, the filter intercepts the `HttpServletRequest` and `HttpServletResponse` objects and creates the respective wrapper objects for its own implementation.

This filter implementation intercepts all HTTP session related calls that are made on the `HttpServletRequest` and `HttpServletResponse` objects. These calls are satisfied by the session manager and are not passed through to the base session manager in the underlying JEE server implementation. The session manager creates and maintains sessions and their life cycles, including timeout-based invalidations and the firing of listeners on session invalidations and other life cycle events.

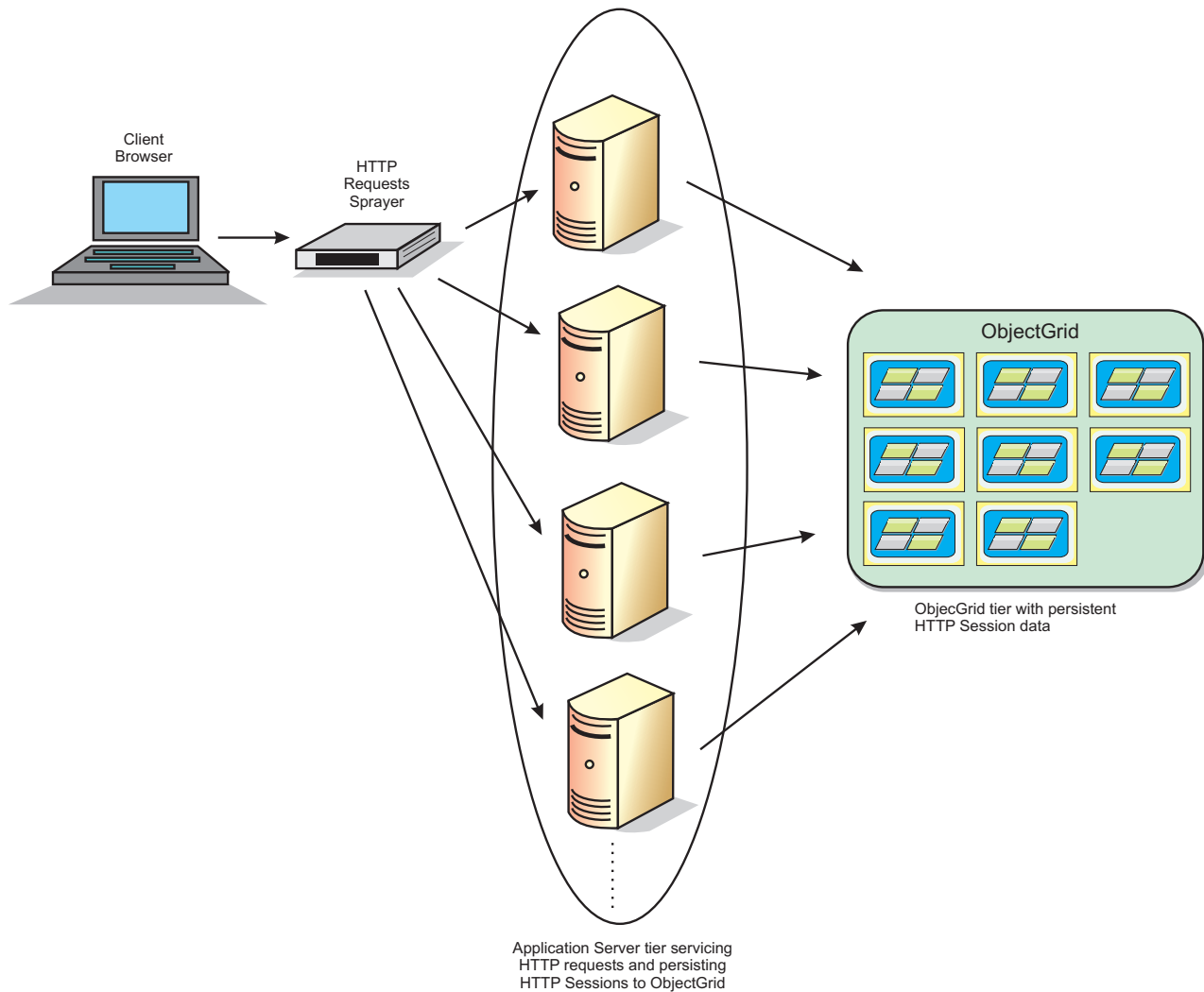


Figure 6. HTTP session management topology with a remote container configuration

Deployment topologies

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

- **Embedded, network attached eXtreme Scale containers**

In this scenario, the eXtreme Scale servers are collocated in the same processes as the servlets. The session manager can communicate directly to the local ObjectGrid instance, avoiding costly network delays.

- **Remote, network attached eXtreme Scale containers**

In this scenario, the eXtreme Scale servers run in external processes the process in which the servlets run. The session manager communicates with a remote eXtreme Scale server grid.

Retrieving an eXtreme Scale session in a session manager application

```
public void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {
```

```
    HttpSession session = req.getSession(true);
```

```

        System.out.println("calling getSession");
        // call getAttribute("com.ibm.websphere.objectgrid.session")
// to get the ObjectGrid session instance
        Session ogSession = (Session)session.getAttribute
("com.ibm.websphere.objectgrid.session");

        System.out.println("ogSession = "+ogSession);
    }

```

Any changes made to the maps with the session that is returned from the `getAttribute` method call are committed when the underlying session is committed.

Related tasks

“Configuring the HTTP session manager with WebSphere Application Server”

While WebSphere Application Server provides session management function, this support does not scale under extreme request loads. WebSphere eXtreme Scale comes bundled with a session management implementation that overrides the default session manager for a Web container and provides better scalability and more robust configuration options.

“Configuring the HTTP session manager with WebSphere Application Server Community Edition” on page 154

WebSphere Application Server Community Edition can share session state, but not in an efficient, scalable manner. WebSphere eXtreme Scale provides a high performance, distributed persistence layer that can be used to replicate state, but does not readily integrate with any application server outside of WebSphere Application Server. You can integrate these two products to provide a scalable session-management solution. You can use the WebSphere Application Server Community Edition modular infrastructure, GBeans, to embed WebSphere eXtreme Scale as the session-state persistence mechanism.

Related reference

“Servlet context initialization parameters” on page 158

The following list of servlet context initialization parameters can be specified in the properties file as required in the script or ANT-based splicing methods.

Configuring the HTTP session manager with WebSphere Application Server

While WebSphere Application Server provides session management function, this support does not scale under extreme request loads. WebSphere eXtreme Scale comes bundled with a session management implementation that overrides the default session manager for a Web container and provides better scalability and more robust configuration options.

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 `objectgridRoot/session/samples/objectGrid.xml` and `objectgridRoot/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 eXtreme Scale servers run in different processes than the servlets. The session manager communicates with a remote eXtreme Scale server. To use a remote, network-attached eXtreme Scale server, the session manager must be configured with the host names and port numbers of the eXtreme Scale catalog service cluster. The session manager then uses an eXtreme Scale client connection to communicate with the catalog server and the eXtreme Scale servers.

If the eXtreme Scale servers are to be started in independent, stand-alone processes, start the eXtreme Scale containers using the `objectGridStandAlone.xml` and `objectGridDeploymentStandAlone.xml` files that are supplied in the session manager samples directory.

Configuring your application to use the eXtreme Scale HTTP session manager

Procedure

1. WebSphere eXtreme Scale must be installed on your WebSphere Application Server or WebSphere Application Server Network Deployment in order to use the eXtreme Scale session Manager.
2. 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:

- **Auto-splice option**

When running in WebSphere Application Server or WebSphere Application Server Network Deployment, you do not need to manually splice your Web applications. You can add a custom property to either a cell or a server that will affect all of the Web applications at that scope. The property name is `com.ibm.websphere.xs.sessionFilterProps`, and you should set its value to the location of the `splicer.properties` file your applications require. The following is an example path for the location of a file: `/opt/splicer.properties`. To specify all Web applications in a cell as spliced use the administration console as follows: Go to **System administration > Cell > Custom properties**, and add the property there. To specify all of the Web applications in a given application server as use the administration console as follows: Go to **Application server > <server name> > Administration > Custom properties**, and add the property there.

The cell and server scopes are the only available scopes, and only available when running in WebSphere Application Server or WebSphere Application Server Deployment Manager. If you require a different scope, you should manually splice your Web applications.

- **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. This script, `objectgridRoot/bin/addObjectGridFilter.sh` or `objectgridRoot/session/bin/addObjectGridFilter.bat`, takes two parameters: the application (absolute path to the enterprise archive file) that needs to be spliced, and the absolute path to the 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 on WebSphere Application Server on Unix:

- a. `cd objectgridRoot/optionalLibraries/ObjectGrid/session/bin`
- b. `addObjectGridFilter.sh /tmp/mySessionTest.ear wasRoot/optionalLibraries/ObjectGrid/session/samples/splicer.properties`

UNIX

Example using eXtreme Scale installed in a stand-alone directory on Unix:

- a. `cd objectgridRoot/session/bin`
- b. `addObjectGridFilter.sh /tmp/mySessionTest.ear objectgridRoot/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 158.

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.

- **Ant build script**

WebSphere eXtreme Scale ships with a `build.xml` file that can be used by Apache Ant, which is included in the `wasRoot/bin` folder of a WebSphere Application Server installation. The `build.xml` can be modified 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` has been modified, the Ant process is invoked 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. You should 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 158.

3. 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.

4. 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), session ID length, 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.

Related concepts

“HTTP session management” on page 147

In a WebSphere Application Server Version 5 or later environment, the HTTP session manager that is shipped with WebSphere eXtreme Scale can override the default session manager in the application server.

Related reference

“Servlet context initialization parameters” on page 158

The following list of servlet context initialization parameters can be specified in the properties file as required in the script or ANT-based splicing methods.

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 7. Custom properties for SIP session management with ObjectGrid

Property	Value	Default
<code>com.ibm.sip.ha.replicator.type</code>	<code>OBJECTGRID: use ObjectGrid as SIP session store</code>	
<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 “Distributed eXtreme Scale grid configuration” on page 89 for more information.	true

Table 7. Custom properties for SIP session management with ObjectGrid (continued)

Property	Value	Default
development.mode	<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

Configuring the HTTP session manager with WebSphere Application Server Community Edition

WebSphere Application Server Community Edition can share session state, but not in an efficient, scalable manner. WebSphere eXtreme Scale provides a high performance, distributed persistence layer that can be used to replicate state, but does not readily integrate with any application server outside of WebSphere Application Server. You can integrate these two products to provide a scalable session-management solution. You can use the WebSphere Application Server Community Edition modular infrastructure, GBeans, to embed WebSphere eXtreme Scale as the session-state persistence mechanism.

Before you begin

You must unzip or install Geronimo or WebSphere Application Server Community Edition and WebSphere eXtreme Scale on your system.

About this task

The backbone of WebSphere Application Server Community Edition, the kernel, relies on outward modules, referred to as GBeans, to provide function and capabilities.

Procedure

1. Distribute the GBeans.
 - a. Navigate to the *extremeScale_root/wasce/bin* directory.
 - b. Run the `addToCeRepository` script for the system that you are running.
 - Linux UNIX `addToCeRepository.sh ceRoot`
 - Windows `addToCeRepository.bat ceRoot`
2. Edit the configuration file. Now that the GBean is distributed, configure WebSphere Application Server Community Edition to know that it should be launched at server startup. You can register the GBean and supply it with configurable arguments that can be used at runtime with the `config.xml` file.
 - a. Navigate to the `WasCeRoot/var/config` directory.
 - b. Open the `config.xml` file in a plain text editor.
 - c. Find the following line in the `config.xml` file:

```
<module name="org.apache.geronimo.plugins/mconsole-ds/2.1.1/car"/>
</attributes>
```

- d. Append the following excerpt before the `</attributes>` tag.

```
<module name="com.ibm.websphere.objectgrid/gbean/1.0/car">
  <gbean name="objectgrid/BringupPlugin">
    <attribute name="objectgridHome">c:\objectgrid</attribute>
    <attribute name="geronimoHome">C:\websphereCE</attribute>
    <attribute name="serverName">server1</attribute>
    <attribute name="catalogServiceEndpoints">host:port</attribute>
```

```

        <attribute name="replicationDisabled">false</attribute>
        <attribute name="traceSpecification">*=all=disabled</attribute>
    </gbean>
</module>

```

The values for the GBean parameters in the previous excerpt are examples. You can configure these values.

objectgridHome

Specifies the installation directory of objectgridRoot.

geronimoHome -

Specifies the installation directory of WebSphere Application Server Community Edition or Apache Geronimo.

serverName

Specifies a server name that must be unique for every WebSphere Application Server Community Edition and eXtreme Scale server instance

catalogServiceEndpoints -

Specifies the host and port of the eventual catalog server.

replicationDisabled -

Specifies if replication of session state is enabled.

traceSpecification -

Specifies a string for trace of the eXtreme Scale container within the Java virtual machine (JVM).

3. Start a catalog server. Your WebSphere Application Server Community Edition image can successfully be able to launch an eXtreme Scale server within its JVM at server startup. For an eXtreme Scale to bootstrap, however, you must first start a catalog server. The catalog server is analogous to a deployment manager within a WebSphere Application Server topology and must be started before WebSphere Application Server Community Edition starts. After the catalog server is started, enter the hostname and port for the catalog server in the config.xml file within the GBean entry. See “Starting a stand-alone catalog service” on page 263 for more information. Be sure to read the section on binding the ORB port to a particular hostname and port. You need to specify those endpoints within the GBean configuration. If you have a port conflict on server startup, you must open the ceRoot/var/config/config-substitutions.properties file and change the NamingPort key to a value other than 1099.
4. Start WebSphere Application Server Community Edition to test the bootstrap.
 - a. Navigate to the WasCeRoot/bin directory.
 - b. Set the JAVA_HOME environment variable.
 - UNIX Linux export JAVA_HOME=javaHome
 - Windows set JAVA_HOME=javaHome
 - c. Run the start command.
 - UNIX Linux geronimo.sh run
 - Windows geronimo.bat run

The server should now start up and begin to initialize its components or GBeans. The eXtreme Scale plug-in is the last plug-in to load and outputs the necessary startup trace and information statements.

Related concepts

“HTTP session management” on page 147

In a WebSphere Application Server Version 5 or later environment, the HTTP session manager that is shipped with WebSphere eXtreme Scale can override the default session manager in the application server.

Related reference

“Servlet context initialization parameters” on page 158

The following list of servlet context initialization parameters can be specified in the properties file as required in the script or ANT-based splicing methods.

Using WebSphere eXtreme Scale for replication of session state in WebSphere Application Server Community Edition:

Every Web application that is deployed to WebSphere Application Server Community Edition requires compliance with the Java Platform, Enterprise Edition specification, along with the `geronimo-web.xml` descriptor file. This file contains access and dependency information specific to the runtime environment for WebSphere Application Server Community Edition. To allow a Java EE Web application to use the session replication feature of WebSphere eXtreme Scale, the Web application must supply data about the eXtreme Scale plug-in as and session-specific attributes.

Before you begin

This process is simple enough to create on any operating system, without an IDE or utilities.

About this task

You can set up a basic Web application that uses the session persistence later in eXtreme Scale, which counts and stores the number of times a user has accessed a particular servlet.

Procedure

1. Create the sample Web application.
 - a. Create a directory at some arbitrary location within your file system. For example, you might create a directory called `app_home`.
 - b. Navigate to the `app_home` directory and create the following directory structure:

```
> app_home
--> META-INF
--> WEB-INF
-----> lib
-----> classes
```
 - c. From the `app_home/WEB-INF` directory, create a `web.xml` file. Copy and paste the following code into the `web.xml` file. Ensure that all context parameters are added to the `web.xml` file, or the filter might not work correctly.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="sample" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
<display-name>
sample</display-name>
<context-param>
<param-name>shareSessionsAcrossWebApps</param-name>
<param-value>>false</param-value>
</context-param>
</context-param>
```

```

        <param-name>sessionIDLength</param-name>
        <param-value>23</param-value>
    </context-param>
    <context-param>
        <param-name>defaultSessionTimeout</param-name>
        <param-value>30</param-value>
    </context-param>
    <context-param>
        <param-name>affinityManager</param-name>
        <param-value>com.ibm.ws.httpsession.NoAffinityManager</param-value>
    </context-param>
    <context-param>
        <param-name>useURLEncoding</param-name>
        <param-value>>false</param-value>
    </context-param>
    <context-param>
        <param-name>objectGridName</param-name>
        <param-value>session</param-value>
    </context-param>
    <context-param>
        <param-name>persistenceMechanism</param-name>
        <param-value>ObjectGridStore</param-value>
    </context-param>
    <filter>
        <filter-name>HttpSessionFilter</filter-name>
        <filter-class>com.ibm.ws.httpsession.HttpSessionFilter</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>HttpSessionFilter</filter-name>
        <url-pattern>*</url-pattern>
    </filter-mapping>
    <servlet>
        <description>
        </description>
        <display-name>
        SampleServlet</display-name>
        <servlet-name>SampleServlet</servlet-name>
        <servlet-class>
        SampleServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>SampleServlet</servlet-name>
        <url-pattern>/SampleServlet</url-pattern>
    </servlet-mapping>
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.htm</welcome-file>
        <welcome-file>index.jsp</welcome-file>
        <welcome-file>default.html</welcome-file>
        <welcome-file>default.htm</welcome-file>
        <welcome-file>default.jsp</welcome-file>
    </welcome-file-list>
</web-app>

```

- d. From the *app_home*/*WEB-INF* directory, create a *geronimo-web.xml* file. Copy and paste the following code into the *geronimo-web.xml* file:

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://geronimo.apache.org/xml/ns/j2ee/web-1.1"
  xmlns:nam="http://geronimo.apache.org/xml/ns/naming-1.1"
  xmlns:sec="http://geronimo.apache.org/xml/ns/security-1.1"
  xmlns:sys="http://geronimo.apache.org/xml/ns/deployment-1.1">
  <sys:environment>
    <sys:moduleId>
      <sys:groupId>com.ibm.websphere.objectgrid</sys:groupId>
      <sys:artifactId>sample</sys:artifactId>
      <sys:version>1.0</sys:version>
      <sys:type>war</sys:type>
    </sys:moduleId>
    <sys:dependencies>
      <sys:dependency>
        <sys:groupId>com.ibm.websphere.objectgrid
        </sys:groupId>
        <sys:artifactId>session</sys:artifactId>
        <sys:version>1.0</sys:version>
        <sys:type>jar</sys:type>
      </sys:dependency>
    </sys:dependencies>
  </sys:environment>
  <context-root>/sample</context-root>
</web-app>

```

- e. Place the *SampleServlet.class* attached below in the *WEB-INF/classes* directory.
- f. Navigate to the *app_home* directory.

- g. Run the following command to package your Web archive (WAR) file:

```
jar -cf sample.war
```
 - h. Your Web application is now ready to be deployed.
2. Deploy your Web application.
- a. Login to the WebSphere Application Server Community Edition home page at the specified hostname:port. By default, the home page is:
<http://localhost:8080/>.
 - b. Click **Administrative Console**.
 - c. Type your username and password, then click **OK**. By default, the credentials are:
 - user name: System
 - password: Manager
 - d. Select **Applications > Deploy New** in the left-hand menu.
 - e. Click **Browse** to the right of Archive, then select the `sample.war` file you created in the previous section.
 - f. Leave the **Plan** field blank, because you bundled the plan with the Web application.
 - g. Select **Start App after Install**, then click **Install**.

After the application installs, a message displays that says the installation was successful.

3. Test the application.
- a. Navigate to the `/sample/SampleServlet` context root to test the application. By default, the context root is: `http://localhost:8080/sample/SampleServlet`.
 - b. You should see the number of times you have visited this servlet, followed by the `HttpSession` implementation. It should read `HttpSessionImpl` if it is using the eXtreme Scale persistence layer, or `StandardSessionFacade` if it is using the Web container's default session-storage mechanism.

Servlet context initialization parameters

The following list of servlet context initialization parameters can be specified in the properties file as required in the script or ANT-based splicing methods.

Parameters

persistenceMechanism

Specifies a string value that defines how the session is stored in eXtreme Scale. Indicate one of the following values:

- **ObjectGridStore:** Each session attribute is stored as a different entry in the eXtreme Scale table.
- **ObjectGridAtomicSessionStore:** The entire session is stored as a single entry in the eXtreme Scale table.

objectGridName

Specifies a string value that defines the name of the ObjectGrid that will contain the session data for this Web application. This should match the ObjectGrid name in the XML files provided with eXtreme Scale. The default is "session".

catalogHostPort

Specifies the catalog server connection information; the value needs to be of the form `host:port<,host:port>`. This list can be arbitrarily long and the first

viable address is used. This property should only be used for the remote, network-attached eXtreme Scale scenario.

defaultSessionTimeout

An integer value that defines the amount of time in minutes that a session can be inactive (for example, not accessed from a servlet) before the session is invalidated and removed from the system. The default is 30 minutes.

sessionIDLength

An integer value that defines the length of the String identifiers that are created for HTTP sessions. The default is 23.

shareSessionsAcrossWebApps

Specifies 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.

cookieName

Specifies a string value that defines the name of the cookie for this Web application. The default is JSESSIONID. If you want to use a unique cookie name, add this property to the `splicer.properties` file.

useURLEncoding

Specifies a string value of either true or false, with false as the default. The true setting informs the filter that cookies are not being used, and the URL request provides the session id and eXtreme Scale SessionHandle. The URI can be retrieved by calling `HttpServletRequest.encodeURL(String uri)`.

Related concepts

“HTTP session management” on page 147

In a WebSphere Application Server Version 5 or later environment, the HTTP session manager that is shipped with WebSphere eXtreme Scale can override the default session manager in the application server.

Related tasks

“Configuring the HTTP session manager with WebSphere Application Server” on page 150

While WebSphere Application Server provides session management function, this support does not scale under extreme request loads. WebSphere eXtreme Scale comes bundled with a session management implementation that overrides the default session manager for a Web container and provides better scalability and more robust configuration options.

“Configuring the HTTP session manager with WebSphere Application Server Community Edition” on page 154

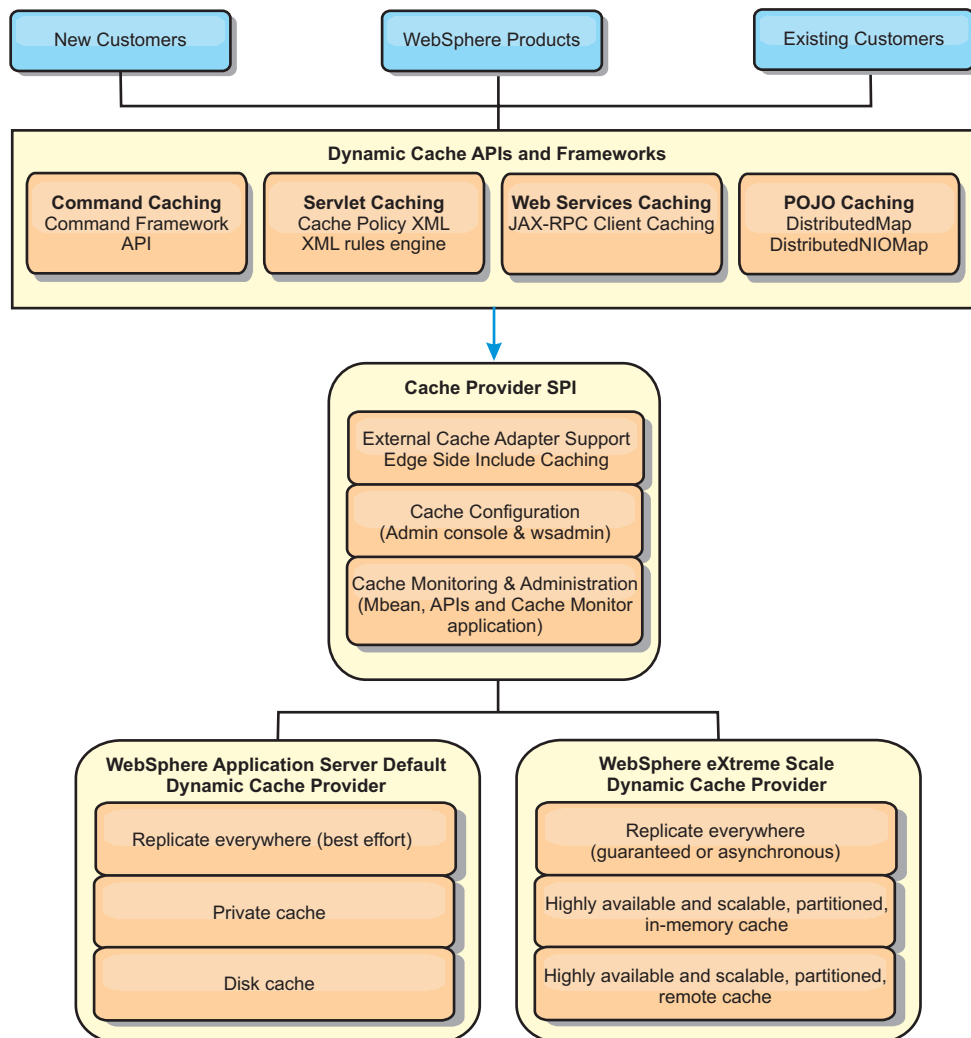
WebSphere Application Server Community Edition can share session state, but not in an efficient, scalable manner. WebSphere eXtreme Scale provides a high performance, distributed persistence layer that can be used to replicate state, but does not readily integrate with any application server outside of WebSphere Application Server. You can integrate these two products to provide a scalable session-management solution. You can use the WebSphere Application Server Community Edition modular infrastructure, GBeans, to embed WebSphere eXtreme Scale as the session-state persistence mechanism.

WebSphere eXtreme Scale dynamic cache provider

The IBM DynaCache API is available to Java EE applications that are deployed in WebSphere Application Server. The dynamic cache provider 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

Previously, the only service provider for the dynamic cache API was the default dynamic cache engine built into WebSphere Application Server. Customers can use the dynamic cache service provider interface in WebSphere Application Server to plug eXtreme Scale into dynamic cache. By setting up this capability, you can enable applications written with the dynamic cache API or applications using container-level caching (such as servlets) to leverage the features and performance capabilities of WebSphere eXtreme Scale.



You can install and configure the dynamic cache provider as described in [Configuring the dynamic cache provider for WebSphere eXtreme Scale](#).

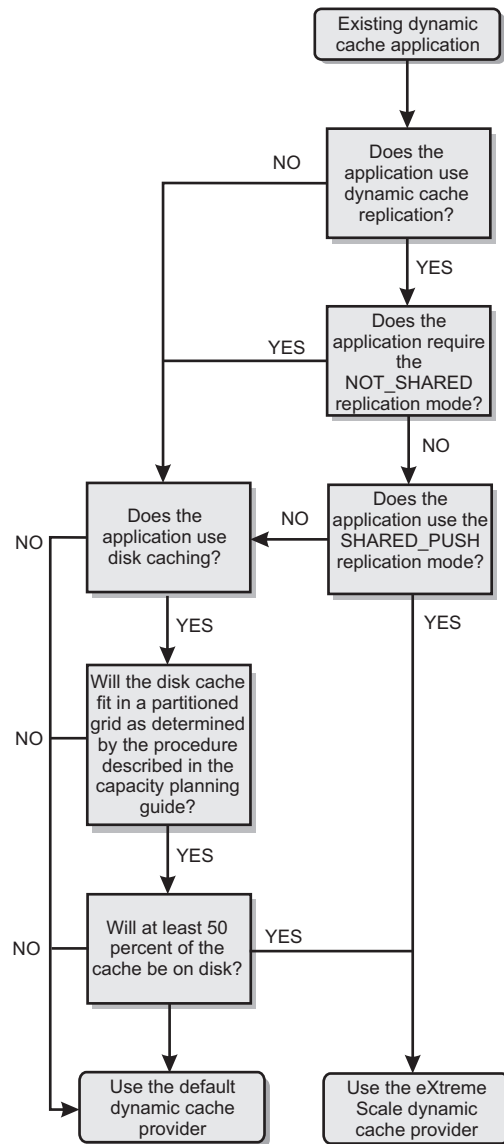
Deciding how to leverage WebSphere eXtreme Scale

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

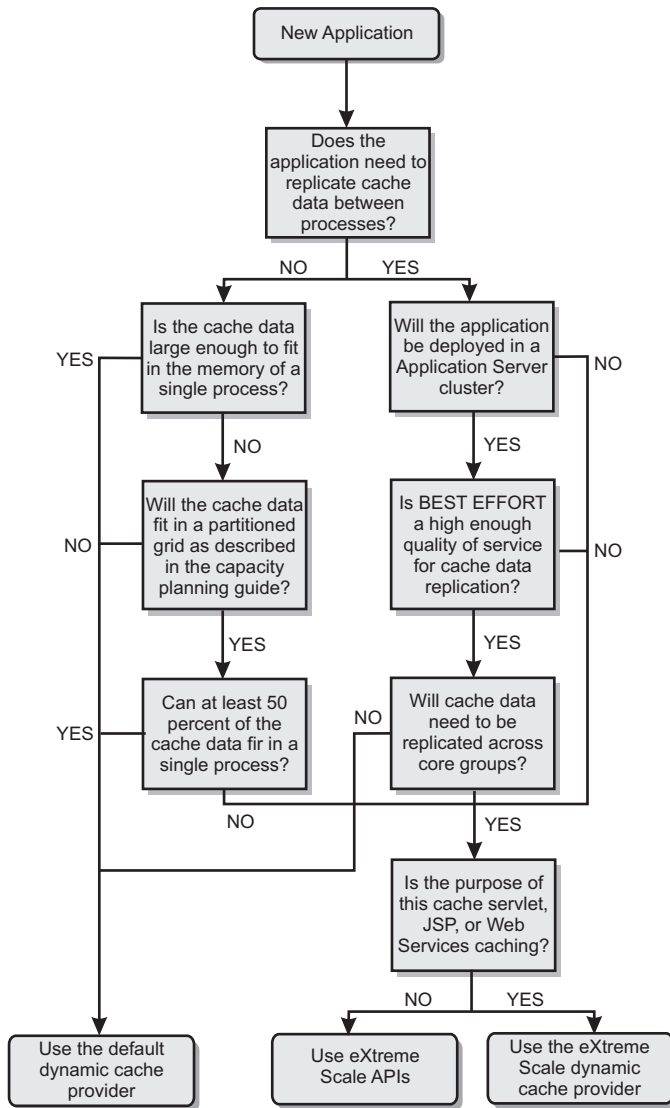
contents for the dynamic cache service. WebSphere eXtreme Scale offers a higher quality of service for cache replication than DRS.

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

Decision tree for migrating existing dynamic cache applications



Decision tree for choosing cache provider for new applications



Feature comparison

Table 8. Feature comparison

Cache features	Default provider	eXtreme Scale provider	eXtreme Scale API
Local, in-memory caching	x	x	x
Distributed caching	Embedded	Embedded, embedded-partitioned and remote-partitioned	Multiple
Linearly scalable		x	x
Reliable replication (synchronous)		ORB	ORB
Disk overflow	x		

Table 8. Feature comparison (continued)

Cache features	Default provider	eXtreme Scale provider	eXtreme Scale API
Eviction	LRU/TTL/heap-based	LRU/TTL (per partition)	Multiple
Invalidation	x	x	x
Relationships	Dependency IDs, templates	Dependency IDs, templates	x
Non-key lookups			Query and index
Back-end integration			Loaders
Transactional		Implicit	x
Key-based storage	x	x	x
Events and listeners	x	x	x
WebSphere Application Server integration	Single cell only	Multiple cell	Cell independent
Java Standard Edition support		x	x
Monitoring and statistics	x	x	x
Security	x	x	x

Table 9. Seamless technology integration

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

Table 10. Programming interfaces

Cache features	Default provider	eXtreme Scale provider	eXtreme Scale API
Command-based API	Command framework API	Command framework API	DataGrid API

Table 10. Programming interfaces (continued)

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

For a more detailed description on how eXtreme Scale distributed caches work, see "Deployment configurations for eXtreme Scale" in the *Programming Guide*.

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

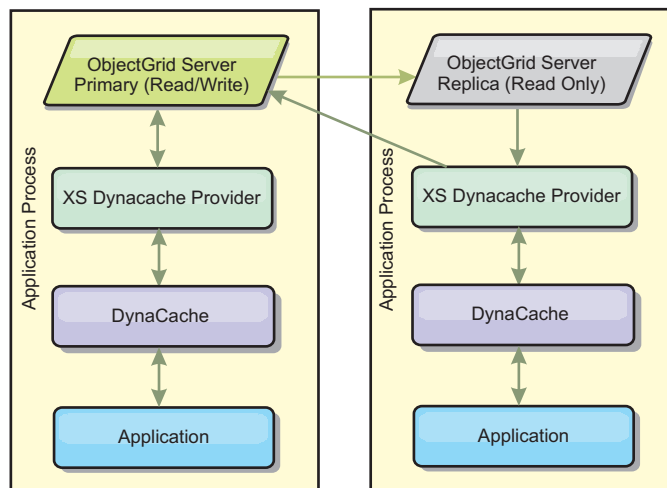
Topology types

A dynamic cache service created with the eXtreme Scale provider can be deployed in any of three available topologies, allowing you to tailor the cache specifically to performance, resource, and administrative needs. These topologies are embedded, embedded partitioned, and remote.

Embedded topology

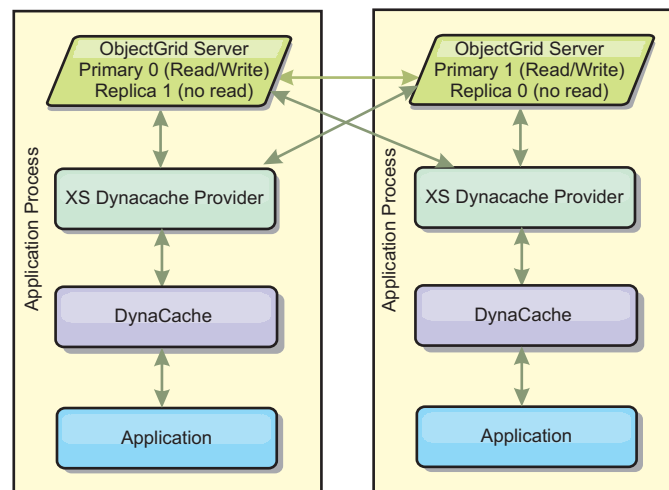
The embedded topology is similar to the default dynamic cache and DRS provider. Distributed cache instances created with the embedded topology keep a full copy of the cache within each eXtreme Scale process that accesses the dynamic cache service, allowing all read operations to occur locally. All write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

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



Embedded partitioned topology

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

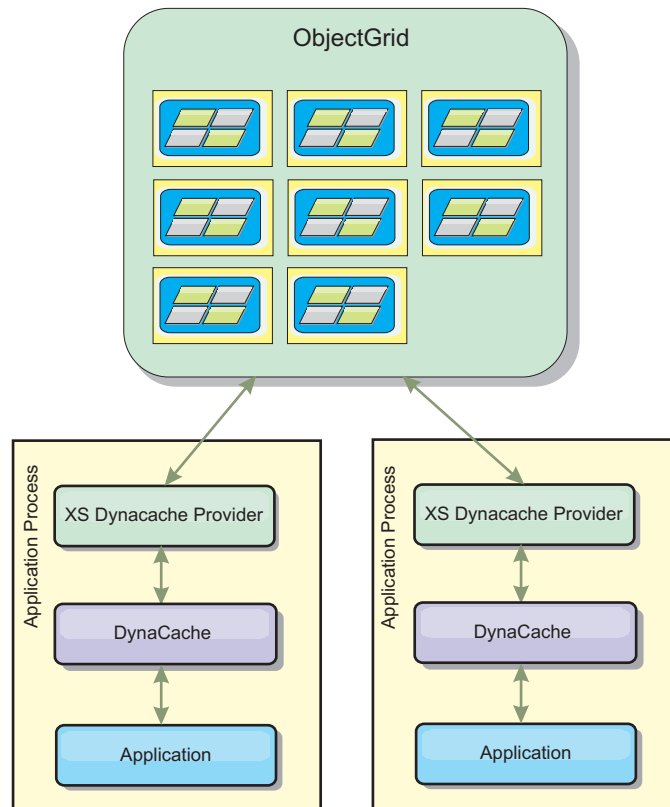


For example, assume you have a grid of server processes with 256 megabytes of free heap each to host a dynamic cache service. The default dynamic cache provider and the eXtreme Scale provider using the embedded topology would both be limited to an in-memory cache size of 256 megabytes minus overhead.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

Remote topology

The remote topology can also be used to eliminate the need for a disk cache. The only difference between the remote and embedded partitioned topologies is that all of the cache data is stored outside of WebSphere Application Server processes when you are using the remote topology. WebSphere eXtreme Scale supports standalone container processes for cache data. These container processes have a lower overhead than a WebSphere Application Server process and are also not

limited to using a particular Java Virtual Machine (JVM). For example, the data for a dynamic cache service being accessed by a 32-bit WebSphere Application Server process could be located in an eXtreme Scale container process running on a 64-bit JVM. This allows users to leverage the increased memory capacity of 64-bit processes for caching, without incurring the additional overhead of 64-bit for application server processes. The remote topology is shown in the following image:



Data compression

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

Local in-memory cache

The WebSphere eXtreme Scale dynamic cache provider can also be used to back dynamic cache instances that have **replication disabled**. Like the default dynamic cache provider, these caches can store non-serializable data. They can also offer better performance than the default dynamic cache provider on large multi-processor enterprise servers because the eXtreme Scale code path is designed to maximize in-memory cache concurrency.

Dynamic cache engine and eXtreme Scale functional differences

In the case of local in-memory caches where replication is disabled, there should be no appreciable functional difference between caches backed by the default dynamic cache provider and WebSphere eXtreme Scale. Users should not notice a functional difference between the two caches except that the WebSphere eXtreme Scale backed caches do not support disk offload or statistics and operations related to the size of the cache in memory.

In the case of caches where replication is enabled there will be no appreciable difference in the results returned by most dynamic cache API calls, regardless of whether the customer is using the default dynamic cache provider or the eXtreme Scale dynamic cache provider. For some operations you cannot emulate the behavior of the dynamic cache engine using eXtreme Scale.

Dynamic cache statistics

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

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

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

- **Cache Hits** : Cache hits are tracked per server. If traffic on Server A generates 10 cache hits and traffic on Server B generates 20 cache hits, the cache statistics will report 10 cache hits on Server A and 20 cache hits on Server B.
- **Cache Misses**: Cache misses are tracked per server just like cache hits.
- **Memory Cache Entries**: This statistic reports the number of cache entries in the distributed cache. Every server that accesses the cache will report the same value for this statistic, and that value will be the total number of cache entries in memory over all the servers.
- **Memory Cache Size in MB**: This metric is not currently supported and will always return -1.
- **Cache Removes**: This statistic reports the total number of entries removed from the cache by any method, and is an aggregate value for the whole distributed cache. If traffic on Server A generates 10 invalidations and traffic on Server B generates 20 invalidations, then the value on both servers will be 30.
- **Cache Least Recently Used (LRU) Removes**: This statistic is aggregate, like cache removes. It tracks the number of entries that were removed to keep the cache under its maximum size.
- **Timeout Invalidations**: This is also an aggregate statistic, and it tracks the number of entries that were removed because they timed out.

- **Explicit Invalidations** : Also an aggregate statistic, this tracks the number of entries that were removed with direct invalidation by key, dependency ID or template.
- **Extended Stats** : The eXtreme Scale dynamic cache provider exports the following extended stat key strings.
 - **com.ibm.websphere.xs.dynacache.remote_hits**: The total number of cache hits tracked at the eXtreme Scale container. This is an aggregate statistic, and its value in the extended stats map is a long.
 - **com.ibm.websphere.xs.dynacache.remote_misses**: The total number of cache misses tracked at the eXtreme Scale container. An aggregate statistic, its value in the extended stats map is a long.

Reporting reset statistics

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

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

Dynamic cache events

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

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

Embedded partitioned caches will generate events on any server that hosts a partition of the cache. So if a cache has 11 partitions and each server in an 11 server WebSphere Network Deployment grid hosts one of the partitions, then each server will receive the dynamic cache events for the cache entries that it hosts. No single server process would see all of the events unless all 11 partitions were hosted in that server process. As with the embedded topology, it is not possible to ensure that a particular server process will receive a particular set of events or any events at all.

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

MBean calls

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

Dynamic cache replication policy mapping

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

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

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

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

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

Security

You can secure dynamic cache instances that are running in embedded or embedded partitioned topologies with the security functionality built into WebSphere Application Server. See the documentation on Securing application servers in the WebSphere Application Server Information Center.

When a cache is running in remote topology, it is possible for a standalone eXtreme Scale client to connect to the cache and affect the contents of the dynamic cache instance. The eXtreme Scale dynamic cache provider has a low overhead encryption feature that can prevent cache data from being read or changed by non-WebSphere Application Server clients. To enable this feature, set the optional parameter `com.ibm.websphere.xs.dynacache.encryption_password` to the same value on every WebSphere Application Server instance that accesses the dynamic

cache provider. This will encrypt the value and user metadata for the CacheEntry using 128-bit AES encryption. It is very important that the same value be set on all servers. Servers will not be able to read data put into the cache by servers with a different value for this parameter.

If the eXtreme Scale provider detects that different values are set for this variable in the same cache, it generate a warning in the log of the eXtreme Scale container process.

See the eXtreme Scale documentation on WebSphere eXtreme Scale security if SSL or client authentication is required.

Additional information

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

Configuring the dynamic cache provider for WebSphere eXtreme Scale

Installing and configuring the dynamic cache provider for WebSphere 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 “Integrating WebSphere eXtreme Scale with WebSphere Application Server” on page 39 for more information

For information about using the eXtreme Scale dynamic cache provider with IBM WebSphere Commerce, see the following topics in the IBM WebSphere Commerce documentation:

- WebSphere Commerce Version 7 Information Center: Enabling the dynamic cache service and servlet caching
- WebSphere Commerce Version 7 Information Center: Enabling WebSphere commerce data cache

About this task

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.

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 grid.

A single catalog service grid can service multiple dynamic cache service providers that are backed by eXtreme Scale. When you run multiple catalog servers, you can connect all the servers together by creating the `catalog.services.cluster` custom property and setting the value to a list of the catalog server endpoints.

For more information about creating catalog server endpoints, see “Starting the catalog service process in a WebSphere Application Server environment” on page 283.

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 267 for more information.

8. Optional: If you are using a remote topology, configure the eXtreme Scale container servers.

- **Embedded or embedded partitioned topology:**

The cached data is stored in WebSphere eXtreme Scale container servers. Container servers can run inside or outside of WebSphere Application Server processes. The eXtreme Scale provider automatically creates containers inside the WebSphere process when you are using embedded or embedded partitioned topologies for a cache instance. No further configuration is needed for these topologies.

- **Remote topology:**

When you are using the remote topology, you must start up stand-alone eXtreme Scale container servers before the WebSphere Application Server instances that access the cache instance start. To start stand-alone containers, see “Starting container processes” on page 267. 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>/optionalLibraries/ObjectGrid/dynacache/etc` directory for installations on top of WebSphere Application Server, or the `<wxs_install_root>/ObjectGrid/dynacache/etc` directory for stand-alone installations. The files are named `dynacache-remote-objectgrid.xml` and `dynacache-remote-definition.xml`. Make copies of these files to edit and use when you are starting stand-alone containers for the eXtreme Scale dynamic cache provider. The **numInitialContainers** parameter in the `dynacache-remote-deployment.xml` file must match the number of container processes that are running.

Note: 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 “Capacity planning and high availability” 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
```

Capacity planning and high availability

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 Network Deployment 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” on page 176.

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.

Once 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.

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.disable_recursive_invalidate:** Dynamic cache invalidation is recursive by default. With the WebSphere eXtreme Scale dynamic cache provider, additional overhead is associated with doing recursive invalidation in embedded partitioned and remote topologies. If a cache does not need recursive invalidation, set this parameter to true to avoid overhead.
- **com.ibm.websphere.xs.dynacache.enable_compression:** The eXtreme Scale dynamic cache provider can compress the cache entries in memory to increase

the density of the cache. This introduces additional overhead for read and write operations, but can save a significant amount of memory for applications like servlet caching.

Configuring evictors

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

About this task

For configuring with XML, see “ObjectGrid descriptor XML file” on page 196.

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.

When using the TTLType.LAST_ACCESS_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 value. The ObjectMap and JavaMap interfaces can override the time-to-live value only when you are using the LAST_ACCESS_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"
      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 a LAST_ACCESS_TIME TTL evictor type 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 effect 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.

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.

Configuring the HashIndex

The HashIndex plug-in supports both the MapIndex and MapRangeIndex interfaces. Defining and using indexes properly can significantly improve query performance.

The following attributes can be used to configure HashIndex using either the ObjectGrid deployment descriptor XML file or a programmatic approach:

- **Name:** The name of the index. The name must be unique for each map. The name is used to retrieve the index object from the ObjectMap instance for the BackingMap.
- **AttributeName:** 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 there is only one attribute name, the HashIndex is a single attribute index, and 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 attribute's getter method is used to access the data.
- **POJOKeyIndex:** Used for non-entity maps. If true, the index will introspect the object in the key part of the map. This 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 will introspect 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 only cast the retrieved index object to the MapIndex interface.

Single-attribute HashIndex vs. 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 may 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. See query optimization using indexes in the *Programming Guide*.

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 supports functions to find data using range functions, such as greater than, less than, or both, while a MapIndex only supports equals functions. 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 following is a summary for using range index.

Table 11. Support for range index

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

Query optimization using HashIndex

Defining and using indexes properly can significantly improve query performance. WebSphere® eXtreme Scale queries can use built-in HashIndex plug-ins to improve performance of queries. Although using indexes can significantly improve query performance, it may have a performance impact on transactional map operations.

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 data 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 187.

See “Enabling the client invalidation mechanism” on page 97 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 Java virtual machines

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 (JMSException e) {
        System.out.println("JMS Exception: " + e);
    } catch (ObjectGridException e) {
        System.out.println("ObjectGrid exception: " + e);
        System.out.println("Caused by: " + e.getCause());
    } catch (Throwable e) {
        System.out.println("Exception : " + e);
    }
    System.out.println("Listener stopped");
}
}

```

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` is a child class of `JMSObjectGridEventListener` and can only override two methods: the `initialize` (optional) and `onMessage` methods. If a child class of `JMSObjectGridEventListener` 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`.

Configuring with XML files

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

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 194 for an example of the deployment policy XML schema.

deploymentPolicy element

The `deploymentPolicy` element is the top-level element of the deployment policy XML file. This element sets up the namespace of the file and the schema location. The schema is defined in the `deploymentPolicy.xsd` file.

- Number of occurrences: One
- Child element: objectgridDeployment element

objectgridDeployment element

The objectgridDeployment element is used to reference an ObjectGrid instance from the ObjectGrid XML file. Within the objectgridDeployment element, you can divide your maps into map sets.

- Number of occurrences: One or more
- Child element: mapSet element

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)

```
<objectgridDeployment objectgridName="objectgridName"/>
```

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 196 which you should couple with the deployment policy file for each ObjectGrid instance.

mapSet element

The mapSet element is used to group maps together. The maps within a mapSet element are partitioned and replicated similarly. Each map must belong to only one mapSet element.

- Number of occurrences: One or more
- Child element: map element

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 containers 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 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. (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 containers 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 eXtreme Scale containers 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 an ObjectGrid instance online. (Optional)

Starting an eXtreme Scale container sends an event to the catalog service. The first time that the number of active containers is equal to the `numInitialContainers` value for a `mapSet` element, the catalog service places the shards from the `mapSet`, provided that `minSyncReplicas` can also be satisfied. After the `numInitialContainers` value has been met, each container-started event can trigger a rebalance of unplaced and previously placed shards. If you know approximately how many containers 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 start. Placement occurs only when you reach the `numInitialContainers` value specified in the `mapSet` element.

autoReplaceLostShards

Specifies if lost shards are placed on other containers. The default value is `true`. When a container is stopped or fails, the shards running on the container 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. (Optional)

placementStrategy

There are two placement strategies. The default strategy is `FIXED_PARTITION`, where the number of primary shards that are placed across available containers 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 is equal to the number of partitions that are defined, with an equal number of replicas placed on other containers. (Optional)

The following code shows the available attributes for a given mapSet element.

```
<mapSet
(1)  name="mapSetName"
(2)  numberOfPartitions="numberOfPartitions"
(3)  minSyncReplicas="minimumNumber"
(4)  maxSyncReplicas="maximumNumber"
(5)  maxAsyncReplicas="maximumNumber"
(6)  replicaReadEnabled="true" | "false"
(7)  numInitialContainers="numberOfInitialContainersBeforePlacement"
(8)  autoReplaceLostShards="true" | "false"
(9)  developmentMode="true" | "false"
(10) placementStrategy="FIXED_PARTITION"|"PER_CONTAINER"
/>
```

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 containers: 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 containers are required to satisfy the replication settings, the numInitialContainers attribute requires 10 available containers before the catalog service attempts to place any of the shards in this mapSet element. After the domain has 10 containers 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 failure is automatically replaced onto another container, provided that a container 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.

map element

Each map in a mapSet element references one of the backingMap elements that is defined in the corresponding ObjectGrid XML file. Every map in a distributed eXtreme Scale environment can belong to only one mapSet element. See the "objectGrid.xsd file" on page 212 for more information on the ObjectGrid XML file.

- 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)

```
<map
(1)  ref="backingMapReference"
/>
```

The companyGridDpMapSetAttr.xml file uses the ref attribute on the map to reference each of the backingMap elements from the companyGrid.xml file.

For information on avoiding XML configuration conflicts, see “Troubleshooting XML configuration” on page 101.

Related concepts

“Troubleshooting XML configuration” on page 101

Occasionally, when you configure eXtreme Scale, you can encounter unexpected behavior in XML configuration.

Related tasks

“Starting container processes” on page 267

You can start eXtreme Scale from the command line using a deployment topology or using a server.properties file.

Related reference

“Distributed eXtreme Scale grid configuration” on page 89

Use the deployment policy descriptor XML file and the objectgrid descriptor XML file to manage your topology.

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 190 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:unique>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

```

        <xsd:field xpath="@ref" />
      </xsd:unique>
    </xsd:element>
  </xsd:sequence>
  <xsd:attribute name="objectgridName" type="xsd:string"
    use="required" />
</xsd:complexType>

<xsd:complexType name="mapSet">
  <xsd:sequence>
    <xsd:element name="map" type="dp:map" maxOccurs="unbounded"
      minOccurs="1" />
    <xsd:element name="zoneMetadata" type="dp:zoneMetadata"
      maxOccurs="1" minOccurs="0">

      <xsd:key name="zoneRuleName">
        <xsd:selector xpath="dp:zoneRule" />
        <xsd:field xpath="@name" />
      </xsd:key>

      <xsd:keyref name="zoneRuleRef"
        refer="dp:zoneRuleName">
        <xsd:selector xpath="dp:shardMapping" />
        <xsd:field xpath="@zoneRuleRef" />
      </xsd:keyref>

    </xsd:element>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string" use="required" />
  <xsd:attribute name="numberOfPartitions" type="xsd:int"
    use="optional" />
  <xsd:attribute name="minSyncReplicas" type="xsd:int"
    use="optional" />
  <xsd:attribute name="maxSyncReplicas" type="xsd:int"
    use="optional" />
  <xsd:attribute name="maxAsyncReplicas" type="xsd:int"
    use="optional" />
  <xsd:attribute name="replicaReadEnabled" type="xsd:boolean"
    use="optional" />
  <xsd:attribute name="numInitialContainers" type="xsd:int"
    use="optional" />
  <xsd:attribute name="autoReplaceLostShards" type="xsd:boolean"
    use="optional" />
  <xsd:attribute name="developmentMode" type="xsd:boolean"
    use="optional" />
  <xsd:attribute name="placementStrategy"
    type="dp:placementStrategy" use="optional" />
</xsd: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>

```

Related concepts

“Troubleshooting XML configuration” on page 101

Occasionally, when you configure eXtreme Scale, you can encounter unexpected behavior in XML configuration.

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 the “objectGrid.xsd file” on page 212 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 0, transactions never time out. (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_JAAS"
      permissionCheckPeriod="45"
      entityMetadataXMLFile="companyGridEntities.xml">
      <backingMap name="Customer"/>
    </objectGrid>
  </objectGrids>
</objectGridConfig>
```

The following code sample demonstrates the programmatic approach to achieving the same configuration as the `companyGridObjectGridAttr.xml` file in the preceding example.

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid companyGrid = objectGridManager.createObjectGrid("CompanyGrid", false);

companyGrid.setSecurityEnabled();
companyGrid.setAuthorizationMechanism(SecurityConstants.AUTHORIZATION_MECHANISM_JAAS);
companyGrid.setPermissionCheckPeriod(45);
companyGrid.registerEntities(new URL("file:companyGridEntities.xml"));
```

backingMap element

The `backingMap` element is used to define a `BackingMap` instance on an `ObjectGrid`. Each of the attributes on the `backingMap` element corresponds to a method on the `BackingMap` interface. For details, see “[BackingMap interface](#)” on page 81.

- 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. Calls to `Session.getMap(String)` results in dynamic map creation if the name passed to the method matches the regular expression specified in the `name` attribute of this `backingMap`. The default value is `false`. (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 “Map entry locking” on page 85.

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 the a copy of the key is required when a map entry is created. Copying the key object allows the application to use the same key object for each ObjectMap operation. Set the value to true to copy the key object when a map entry is created. The default value is false. (Optional)

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 three values: CREATION_TIME, LAST_ACCESS_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 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)

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|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);

```

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 Evictor` 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);

```

backingMapPluginCollections element

The `backingMapPluginCollections` 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 LRUevictor. The Item and OrderLine BackingMaps reference the collection2 backingMapPluginCollection. These BackingMaps each have an LFUEvictor set.

```
<?xml version="1.0" encoding="UTF-8"?>  
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"  
  xmlns="http://ibm.com/ws/objectgrid/config">  
  
  <objectGrids>  
    <objectGrid name="CompanyGrid">  
      <backingMap name="Customer"  
        pluginCollectionRef="customerPlugins"/>  
      <backingMap name="Item" pluginCollectionRef="collection2"/>  
      <backingMap name="OrderLine"  
        pluginCollectionRef="collection2"/>  
      <backingMap name="Order"/>  
    </objectGrid>  
  </objectGrids>  
  <backingMapPluginCollections>  
    <backingMapPluginCollection id="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 bi-directional 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>
      </streamQuerySet>
    </objectGrid>
  </objectGrids>
</objectGridConfig>
```

```

    <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>

```

Related tasks

“Starting container processes” on page 267

You can start eXtreme Scale from the command line using a deployment topology or using a `server.properties` file.

Related reference

“Distributed eXtreme Scale grid configuration” on page 89

Use the deployment policy descriptor XML file and the objectgrid descriptor XML file to manage your topology.

objectGrid.xsd file

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

See the “ObjectGrid descriptor XML file” on page 196 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 247.

```

<?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:element>
  <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: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="ttlEvictorType" type="dgc:ttlEvictorType" 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="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 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: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:restriction>
</xsd:simpleType>

</xsd:schema>
```

Related concepts

“Troubleshooting XML configuration” on page 101

Occasionally, when you configure eXtreme Scale, you can encounter unexpected behavior in XML configuration.

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 222 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 218

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 218

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 218

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 218

attributes element

Child elements

- “id element” on page 217
- “basic element” on page 217
- “version element” on page 218
- “many-to-one element” on page 219
- “one-to-many element” on page 219
- “one-to-one element” on page 218
- “many-to-many element”
- “transient element” on page 218

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 218
- “attributes element” on page 220

entity-mappings element

Child elements

- **description:** Specifies a description.
- “Entity element” on page 220

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 222

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 217 for the descriptions of each element and attribute of the emd.xsd file.

emd.xsd file

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:emd="http://ibm.com/ws/projector/config/emd"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://ibm.com/ws/projector/config/emd"
  elementFormDefault="qualified" attributeFormDefault="unqualified"
  version="1.0">

  <!-- ***** -->
  <xsd:element name="entity-mappings">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="description" type="xsd:string" minOccurs="0" />
        <xsd:element name="entity" type="emd:entity" minOccurs="1" maxOccurs="unbounded" />
      </xsd:sequence>
    </xsd:complexType>
    <xsd:unique name="uniqueEntityClassName">
      <xsd:selector xpath="emd:entity" />
      <xsd:field xpath="@class-name" />
    </xsd:unique>
  </xsd:element>

  <!-- ***** -->
  <xsd:complexType name="entity">
    <xsd:sequence>
      <xsd:element name="description" type="xsd:string" minOccurs="0" />
      <xsd:element name="id-class" type="emd:id-class" minOccurs="0" />
      <xsd:element name="attributes" type="emd:attributes" minOccurs="0" />
      <xsd:element name="entity-listeners" type="emd:entity-listeners" minOccurs="0" />
      <xsd:element name="pre-persist" type="emd:pre-persist" minOccurs="0" />
      <xsd:element name="post-persist" type="emd:post-persist" minOccurs="0" />
      <xsd:element name="pre-remove" type="emd:pre-remove" minOccurs="0" />
      <xsd:element name="post-remove" type="emd:post-remove" minOccurs="0" />
      <xsd:element name="pre-invalidate" type="emd:pre-invalidate" minOccurs="0" />
      <xsd:element name="post-invalidate" type="emd:post-invalidate" minOccurs="0" />
      <xsd:element name="pre-update" type="emd:pre-update" minOccurs="0" />
      <xsd:element name="post-update" type="emd:post-update" minOccurs="0" />
      <xsd:element name="post-load" type="emd:post-load" minOccurs="0" />
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required" />
    <xsd:attribute name="class-name" type="xsd:string" use="required" />
    <xsd:attribute name="access" type="emd:access-type" />
    <xsd:attribute name="schemaRoot" type="xsd:boolean" />
  </xsd:complexType>

  <!-- ***** -->
  <xsd:complexType name="attributes">
    <xsd:sequence>
      <xsd:choice>
        <xsd:element name="id" type="emd:id" minOccurs="0" maxOccurs="unbounded" />
      </xsd:choice>
      <xsd:element name="basic" type="emd:basic" minOccurs="0" maxOccurs="unbounded" />
      <xsd:element name="version" type="emd:version" minOccurs="0" maxOccurs="unbounded" />
      <xsd:element name="many-to-one" type="emd:many-to-one" minOccurs="0" maxOccurs="unbounded" />
      <xsd:element name="one-to-many" type="emd:one-to-many" minOccurs="0" maxOccurs="unbounded" />
      <xsd:element name="one-to-one" type="emd:one-to-one" minOccurs="0" maxOccurs="unbounded" />
      <xsd:element name="many-to-many" type="emd:many-to-many" minOccurs="0" maxOccurs="unbounded" />
      <xsd:element name="transient" type="emd:transient" minOccurs="0" maxOccurs="unbounded" />
    </xsd:sequence>
  </xsd:complexType>

  <!-- ***** -->
  <xsd:simpleType name="access-type">
```

```

        <xsd:restriction base="xsd:token">
            <xsd:enumeration value="PROPERTY" />
            <xsd:enumeration value="FIELD" />
        </xsd:restriction>
    </xsd:simpleType>

<!-- ***** -->
<xsd:complexType name="id-class">
    <xsd:attribute name="class-name" type="xsd:string" use="required" />
</xsd:complexType>

<!-- ***** -->
<xsd:complexType name="id">
    <xsd:attribute name="name" type="xsd:string" use="required" />
    <xsd:attribute name="type" type="xsd:string" />
    <xsd:attribute name="alias" type="xsd:string" use="optional" />
</xsd:complexType>

<!-- ***** -->
<xsd:complexType name="transient">
    <xsd:attribute name="name" type="xsd:string" use="required" />
</xsd:complexType>

<!-- ***** -->
<xsd:complexType name="basic">
    <xsd:attribute name="name" type="xsd:string" use="required" />
    <xsd:attribute name="alias" type="xsd:string" />
    <xsd:attribute name="type" type="xsd:string" />
    <xsd:attribute name="fetch" type="emd:fetch-type" />
</xsd:complexType>

<!-- ***** -->
<xsd:simpleType name="fetch-type">
    <xsd:restriction base="xsd:token">
        <xsd:enumeration value="LAZY" />
        <xsd:enumeration value="EAGER" />
    </xsd:restriction>
</xsd:simpleType>

<!-- ***** -->
<xsd:complexType name="many-to-one">
    <xsd:sequence>
        <xsd:element name="cascade" type="emd:cascade-type" minOccurs="0" />
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required" />
    <xsd:attribute name="alias" type="xsd:string" />
    <xsd:attribute name="target-entity" type="xsd:string" />
    <xsd:attribute name="fetch" type="emd:fetch-type" />
    <xsd:attribute name="id" type="xsd:boolean" />
</xsd:complexType>

<!-- ***** -->
<xsd:complexType name="one-to-one">
    <xsd:sequence>
        <xsd:element name="cascade" type="emd:cascade-type" minOccurs="0" />
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required" />
    <xsd:attribute name="alias" type="xsd:string" />
    <xsd:attribute name="target-entity" type="xsd:string" />
    <xsd:attribute name="fetch" type="emd:fetch-type" />
    <xsd:attribute name="mapped-by" type="xsd:string" />
    <xsd:attribute name="id" type="xsd:boolean" />
</xsd:complexType>

<!-- ***** -->
<xsd:complexType name="one-to-many">
    <xsd:sequence>
        <xsd:element name="order-by" type="emd:order-by" minOccurs="0" />
        <xsd:element name="cascade" type="emd:cascade-type" minOccurs="0" />
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required" />
    <xsd:attribute name="alias" type="xsd:string" />
    <xsd:attribute name="target-entity" type="xsd:string" />
    <xsd:attribute name="fetch" type="emd:fetch-type" />
    <xsd:attribute name="mapped-by" type="xsd:string" />
</xsd:complexType>

<!-- ***** -->
<xsd:complexType name="many-to-many">
    <xsd:sequence>
        <xsd:element name="order-by" type="emd:order-by" minOccurs="0" />
        <xsd:element name="cascade" type="emd:cascade-type" minOccurs="0" />
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required" />
    <xsd:attribute name="alias" type="xsd:string" />
    <xsd:attribute name="target-entity" type="xsd:string" />
    <xsd:attribute name="fetch" type="emd:fetch-type" />
    <xsd:attribute name="mapped-by" type="xsd:string" />
</xsd:complexType>

<!-- ***** -->
<xsd:simpleType name="order-by">
    <xsd:restriction base="xsd:string" />

```



```

</xsd:simpleType>

<!-- ***** -->
<xsd:complexType name="cascade-type">
  <xsd:sequence>
    <xsd:element name="cascade-all" type="emd:emptyType" minOccurs="0" />
    <xsd:element name="cascade-persist" type="emd:emptyType" minOccurs="0" />
    <xsd:element name="cascade-remove" type="emd:emptyType" minOccurs="0" />
    <xsd:element name="cascade-invalidate" type="emd:emptyType" minOccurs="0" />
    <xsd:element name="cascade-merge" type="emd:emptyType" minOccurs="0" />
    <xsd:element name="cascade-refresh" type="emd:emptyType" minOccurs="0" />
  </xsd:sequence>
</xsd:complexType>

<!-- ***** -->
<xsd:complexType name="emptyType" />

<!-- ***** -->
<xsd:complexType name="version">
  <xsd:attribute name="name" type="xsd:string" use="required"/>
  <xsd:attribute name="alias" type="xsd:string" />
  <xsd:attribute name="type" type="xsd:string" />
</xsd:complexType>

<!-- ***** -->
<xsd:complexType name="entity-listeners">
  <xsd:sequence>
    <xsd:element name="entity-listener" type="emd:entity-listener" minOccurs="0" maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:complexType>

<!-- ***** -->
<xsd:complexType name="entity-listener">
  <xsd:sequence>
    <xsd:element name="pre-persist" type="emd:pre-persist" minOccurs="0" />
    <xsd:element name="post-persist" type="emd:post-persist" minOccurs="0" />
    <xsd:element name="pre-remove" type="emd:pre-remove" minOccurs="0" />
    <xsd:element name="post-remove" type="emd:post-remove" minOccurs="0" />
    <xsd:element name="pre-invalidate" type="emd:pre-invalidate" minOccurs="0" />
    <xsd:element name="post-invalidate" type="emd:post-invalidate" minOccurs="0" />
    <xsd:element name="pre-update" type="emd:pre-update" minOccurs="0" />
    <xsd:element name="post-update" type="emd:post-update" minOccurs="0" />
    <xsd:element name="post-load" type="emd:post-load" minOccurs="0" />
  </xsd:sequence>
  <xsd:attribute name="class-name" type="xsd:string" use="required" />
</xsd:complexType>

<!-- ***** -->
<xsd:complexType name="pre-persist">
  <xsd:attribute name="method-name" type="xsd:string" use="required" />
</xsd:complexType>

<!-- ***** -->
<xsd:complexType name="post-persist">
  <xsd:attribute name="method-name" type="xsd:string" use="required" />
</xsd:complexType>

<!-- ***** -->
<xsd:complexType name="pre-remove">
  <xsd:attribute name="method-name" type="xsd:string" use="required" />
</xsd:complexType>

<!-- ***** -->
<xsd:complexType name="post-remove">
  <xsd:attribute name="method-name" type="xsd:string" use="required" />
</xsd:complexType>

<!-- ***** -->
<xsd:complexType name="pre-invalidate">
  <xsd:attribute name="method-name" type="xsd:string" use="required" />
</xsd:complexType>

<!-- ***** -->
<xsd:complexType name="post-invalidate">
  <xsd:attribute name="method-name" type="xsd:string" use="required" />
</xsd:complexType>

<!-- ***** -->
<xsd:complexType name="pre-update">
  <xsd:attribute name="method-name" type="xsd:string" use="required" />
</xsd:complexType>

<!-- ***** -->
<xsd:complexType name="post-update">
  <xsd:attribute name="method-name" type="xsd:string" use="required" />
</xsd:complexType>

<!-- ***** -->
<xsd:complexType name="post-load">

```

```

        <xsd:attribute name="method-name" type="xsd:string" use="required" />
    </xsd:complexType>
</xsd:schema>

```

Security descriptor XML file

Use an ObjectGrid security descriptor XML file to configure an eXtreme Scale deployment topology with security enabled. The following sample XML files describe several configurations.

Each element and attribute of the cluster XML file is described in the following list. Use the examples to learn how to use these elements and attributes to configure the environment.

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 342. (Optional)

singleSignOnEnabled

Allows a client to connect to any server after it has authenticated with one of the servers if 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 346. (Optional)

The following security.xml file is a sample configuration to enable the eXtreme Scale 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 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 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`. In this example, the system credential generator is a `com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredentialGenerator`, 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 to an eXtreme Scale deployment.

See the “Security descriptor XML file” on page 225 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:restriction>
  </xsd:simpleType>
</xsd:schema>
```

```

<xsd:enumeration value="byte" />
<xsd:enumeration value="java.lang.Short" />
<xsd:enumeration value="short" />
<xsd:enumeration value="java.lang.Long" />
<xsd:enumeration value="long" />
<xsd:enumeration value="java.lang.Float" />
<xsd:enumeration value="float" />
<xsd:enumeration value="java.lang.Character" />
<xsd:enumeration value="char" />
</xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="adminAuthorizationMechanism">
<xsd:restriction base="xsd:string">
  <xsd:enumeration value="AUTHORIZATION_MECHANISM_JAAS" />
  <xsd:enumeration value="AUTHORIZATION_MECHANISM_CUSTOM" />
</xsd:restriction>
</xsd:simpleType>
</xsd:schema>

```

Properties file reference

Server properties files contain settings for running your catalog servers and container servers. You can specify a server properties file for either a stand-alone or WebSphere Application Server configuration. Client property files contain settings for your client.

Sample properties files

You can use the following sample properties files that are in the *extremescale_root\properties* directory to create your properties file:

- `sampleServer.properties`
- `sampleClient.properties`

Deprecated system properties

-Dcom.ibm.websphere.objectgrid.CatalogServerProperties

The property was deprecated in WebSphere eXtreme Scale Version 7.0. Use the **-Dobjectgrid.server.props** property.

-Dcom.ibm.websphere.objectgrid.ClientProperties

The property was deprecated in WebSphere eXtreme Scale Version 7.0. Use the **-Dobjectgrid.client.props** property.

-Dobjectgrid.security.server.prop

The property was deprecated in WebSphere eXtreme Scale Version 6.1.0.3. Use the **-Dobjectgrid.server.prop** property.

-serverSecurityFile

This argument was deprecated in WebSphere eXtreme Scale Version 6.1.0.3. This option is passed into the `startOgServer` script. Use the **-serverProps** argument.

Related concepts

“Transport layer security and secure sockets layer” on page 349
WebSphere eXtreme Scale supports both TCP/IP and Transport Layer Security/Secure Sockets Layer (TLS/SSL) for secure communication between clients and servers.

Related tasks

“Starting stand-alone WebSphere eXtreme Scale servers” on page 262
When you are running a stand-alone WebSphere eXtreme Scale configuration, the environment is comprised of catalog servers, container servers, and eXtreme Scale client processes. You must manually configure and start these processes.

“Administering WebSphere eXtreme Scale with WebSphere Application Server” on page 283

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.

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 the catalog service and container servers.

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

You can specify the server 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 server properties file, the properties in that file override the values in the `objectGridServer.properties` file that is in the classpath.

1. As a well-named file in the classpath. 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`
2. As a system property in either a stand-alone or WebSphere Application Server configuration that specifies a file in the system current directory. In WebSphere Application Server, put the file in the `<was_root>/properties` directory. The file cannot be in the classpath:
`-Dobjectgrid.server.props=file_name`
3. 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`
4. As a programmatic override using 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

7.0.0.0 FIX 2+ 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

7.0.0.0 FIX 2+ 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.

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 cluster. 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 grid 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 grid 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 grid end points for the catalog service. This property specifies the catalog service end points to start the catalog service grid. 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 344 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 key store. This property is used if the key store 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 key store. 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 trust store. 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 key store file.

Example:

```
etc/test/security/client.private
```

trustStore

Specifies a fully qualified path to the trust store file.

Example:

```
etc/test/security/server.public
```

keyStorePassword

Specifies the string password to the key store. You can encode this value or use the actual value.

trustStorePassword

Specifies a string password to the trust store. 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 setting

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 342

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 key store 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 key store 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.

Related concepts

“Transport layer security and secure sockets layer” on page 349
WebSphere eXtreme Scale supports both TCP/IP and Transport Layer Security/Secure Sockets Layer (TLS/SSL) for secure communication between clients and servers.

Related tasks

“Starting stand-alone WebSphere eXtreme Scale servers” on page 262
When you are running a stand-alone WebSphere eXtreme Scale configuration, the environment is comprised of catalog servers, container servers, and eXtreme Scale client processes. You must manually configure and start these processes.

“Administering WebSphere eXtreme Scale with WebSphere Application Server” on page 283

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.

Client properties file

You can create a properties file based on your requirements for eXtreme Scale client processes.

Sample client properties file

You can use the `sampleClient.properties` file that is in the `extremescale_root\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 classpath.

1. As a well-named file anywhere in the classpath. 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 classpath:
`-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

`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 security enabled 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 retried if the credential is expired. If the value is set to 0, attempts to authenticate are not retried.

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 key store. This property is used if the key store 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 key store. 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 trust store. 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 key store file.

Example:

```
etc/test/security/client.private
```

trustStore

Specifies a fully qualified path to the trust store file.

Example:

```
etc/test/security/server.public
```

keyStorePassword

Specifies the string password to the key store. You can encode this value or use the actual value.

trustStorePassword

Specifies a string password to the trust store. You can encode this value or use the actual value.

Related concepts

“Transport layer security and secure sockets layer” on page 349
WebSphere eXtreme Scale supports both TCP/IP and Transport Layer Security/Secure Sockets Layer (TLS/SSL) for secure communication between clients and servers.

Related tasks

“Starting stand-alone WebSphere eXtreme Scale servers” on page 262
When you are running a stand-alone WebSphere eXtreme Scale configuration, the environment is comprised of catalog servers, container servers, and eXtreme Scale client processes. You must manually configure and start these processes.

“Administering WebSphere eXtreme Scale with WebSphere Application Server” on page 283

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.

ORB properties file

The `orb.properties` file is used to pass the properties that are used by the Object Request Broker (ORB) to modify the transport behavior of the data grid.

Location

The `orb.properties` file is in the `java/jre/lib` directory. When you modify the file in a WebSphere Application Server `java/jre/lib` directory, the application servers that are configured under that installation also use the settings from the file.

Baseline settings

The following settings are a good baseline but not necessarily the best settings for every environment. You should 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.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.

Request timeout

Property name: com.ibm.CORBA.RequestTimeout

Value: Integer value for number of seconds.

Description: Indicates how many seconds any request should wait 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 time-outs.

Connect timeout

Property name: com.ibm.CORBA.ConnectTimeout

Value: Integer value for number of seconds.

Description: Indicates how many seconds a socket connection attempt should wait 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 a connections should be relatively constant.

Fragment timeout

Property name: com.ibm.CORBA.FragmentTimeout

Value: Integer value for number of seconds.

Description: Indicates how many seconds a fragment request should wait 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 time-outs.

Connection multiplicity

Property name: com.ibm.CORBA.ConnectionMultiplicity

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.**Description:** Allows the ORB to use multiple connections to any server. In theory, setting this value should promote 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

Value: An integer value for the number of connections.**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 the com.ibm.CORBA.MaxOpenConnections value is passed. Purging connections might cause poor behavior in the data grid.

Is Growable

Property name: com.ibm.CORBA.ThreadPool.IsGrowable

Value: Boolean; set to true or false. **Description:** If enabled, allows the thread pool that the ORB uses for incoming requests to 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.

Server socket queue depth

Property name: com.ibm.CORBA.ServerSocketQueueDepth

Value: An integer value for the number of connections. **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

Value: An integer number that specifies the number of bytes. The default is 1024. **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 overhead.

No local copies

Property name: com.ibm.CORBA.iiop.NoLocalCopies

Value: Boolean; set to true or false. **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 invoked 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

Value: Boolean; set to true or false. **Description:** Specifies whether the ORB invokes 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 overhead that using local interceptors introduces.

Attention: If you have WebSphere eXtreme Scale security enabled, set the com.ibm.CORBA.NoLocalInterceptors property value to false. The security infrastructure uses interceptors for authentication.

When you want to enforce transport security between ObjectGrid clients and servers, you have to add more properties to the orb.properties file. For more information about these properties, see the section about the orb.properties file for transport security support in “Transport layer security and secure sockets layer” on page 349.

Related concepts

“Transport layer security and secure sockets layer” on page 349
WebSphere eXtreme Scale supports both TCP/IP and Transport Layer Security/Secure Sockets Layer (TLS/SSL) for secure communication between clients and servers.

Related tasks

“Configuring a custom Object Request Broker” on page 68
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 need to use a "custom" ORB, the ORB supplied with the IBM SDK is a good choice, although you will need to perform some configuration, as described here. ORBs from other vendors can be used, also with configuration.

“Using the Object Request Broker with stand-alone WebSphere eXtreme Scale processes” on page 38

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.

Integrating with Spring framework

Spring is a popular framework for developing Java applications. WebSphere eXtreme Scale provides support to allow Spring to manage eXtreme Scale transactions and configure the clients and servers comprising your deployed in-memory data grid.

Spring managed native transactions

Spring provides container-managed transactions that are similar to a Java Platform, Enterprise Edition application server. However, the Spring mechanism can plug in different implementations. WebSphere eXtreme Scale provides transaction manager integration which allows Spring to manage the ObjectGrid transaction life cycles. See the information about native transactions in the *Programming Guide* for details.

Spring managed extension beans and namespace support

Also, eXtreme Scale integrates with Spring to allow Spring-style beans defined for extension points or plug-ins. This feature provides more sophisticated configurations and more flexibility for configuring the extension points.

In addition to Spring managed extension beans, eXtreme Scale provides a Spring namespace called "objectgrid". Beans and built-in implementations are pre-defined in this namespace, which makes it easier for users to configure eXtreme Scale. Refer to “Spring extension beans and namespace support” on page 243 for more details on these topics and a sample of how to start an eXtreme Scale container server using Spring configurations.

Shard scope support

With the traditional style Spring configuration, an ObjectGrid bean can either be a singleton type or prototype type. ObjectGrid also supports a new scope called the "shard" scope. If a bean is defined as shard scope, then only one bean is created

per shard. All requests for beans with an ID or ids matching that bean definition in the same shard will result in that one specific bean instance being returned by the Spring container.

The following example shows that a `com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl` bean is defined with scope set to `shard`. Therefore, only one instance of the `JPAPropFactoryImpl` class is created per shard.

```
<bean id="jpaPropFactory" class="com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl" scope="shard" />
```

Spring Web Flow

Spring Web Flow stores its session state in the HTTP Session by default. If a web application is configured to use eXtreme Scale for session management then it is used automatically by Spring to store this state and it is made fault tolerant in the same manner as the session.

Packaging

The eXtreme Scale Spring extensions are in the `ogspring.jar` file. This Java archive (JAR) file must be on the class path for Spring support to work. If a JEE application that is running in a WebSphere Extended Deployment augmented WebSphere Application Server Network Deployment, then the application should place the `spring.jar` file and its associated files in the enterprise archive (EAR) modules. You must also place the `ogspring.jar` file in the same location.

Related tasks

“Configuring a locking strategy” on page 87

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

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 cases, you must use Spring to configure certain plug-in objects. Take the following configuration as an 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, `com.ibm.websphere.objectgrid.jpa.JPATxCallback` class, is configured as the `TransactionCallback` class. This class is configured with one property `persistenceUnitName` 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 registerSpringBeanAdapterFactory(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-ins. 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
- 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*.

Still 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.

Starting container server with Spring extension beans

In this example, we will show how to start an ObjectGrid server using ObjectGrid Spring managed extension beans and namespace support.

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>

```

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>

```

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 6 beans defined in this spring XML file:

1. *objectgrid:register*: This register the default bean factory for the ObjectGrid "Grid".
2. *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.
3. *objectgrid:catalog*: This defines an ObjectGrid catalog service endpoint, which is set to "localhost:2809".
4. *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.
5. *objectgrid:LRUEvictor*: This defines an LRUEvictor with the number of LRU queues to use set to 31.
6. *bean partitionListener*: This defines a *ShardListener* plug-in. This class is a class plugged in by users, 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.

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();
    }
}

```

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 252 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
(1) id="register id"
(2) gridname="ObjectGrid name"
/>
```

server element

Use the server element to define an eXtreme Scale 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.

```
<server
(1) id="server id"
(2) tracespec="the server trace specification"
(3) tracefile="the server trace file"
(4) statspec="the server statistic specification"
(5) jmxport="JMX port number"
(6) isCatalog="true"|"false"
(7) name="the server name"
/>
```

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
(1) host="catalog service host name"
(2) port="catalog service port number"
/>
```

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
(1) objectgridxml="the objectgrid descriptor XML file"
(2) deploymentxml ="the objectgrid deployment descriptor XML file "
(3) 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
(1) entityClassName="the entity class name"
(2) 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
(1) persistenceUnitName="the JPA persistence unit name"
(2) jpaPropertyFactory = "JPAPropertyFactory bean reference"
(3) 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
(1) entityClassName="the entity class name"
(2) 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
(1) maxSize="int"
(2) sleepTime ="seconds"
(3) 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
(1) maxSize="int"
(2) sleepTime ="seconds"
(3) 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

```
(1) name="index name"
(2) attributeName="attribute name"
(3) rangeIndex ="true"|"false"
(4) fieldAccessAttribute ="true"|"false"
(5) 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 247 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: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="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>

```

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 -XmxNNM` 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 WebSphere eXtreme Scale JVM tuning. 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. There are some important modifications to note in this list of best practices for a stand-alone WebSphere eXtreme Scale environment when deploying into a WebSphere Application Server Network Deployment environment.

You must place any additional JVM command line parameters in the same location as the garbage collection policy parameters specified in the previous section.

An acceptable initial target for sustained processor loads is 50% with short duration peek loads hitting up to 75%. Beyond this, you must add additional capacity before you see measurable degradation in predictability and consistency. You can increase performance slightly if you can tolerate longer response times. Exceeding an 80% threshold often leads to significant degradation in consistency and predictability.

Chapter 7. Administering the WebSphere eXtreme Scale environment

Administering the product environment includes starting and stopping servers in stand-alone mode or in WebSphere Application Server. You can also use WebSphere eXtreme Scale as a session manager in a WebSphere Application Server environment.

About this task

Server types

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*. Container servers are the Java virtual machines that store the application data for the grid.

Stand-alone mode

Stand-alone mode refers to a WebSphere eXtreme Scale configuration that is running by itself, without any other application server products.

Running within WebSphere Application Server

When you run WebSphere eXtreme Scale on top of WebSphere Application Server, catalog servers automatically start on the WebSphere Application Server servers. To start container servers, you must package and deploy your application with ObjectGrid XML files.

Related concepts

“Hardware and software requirements” on page 7

You are not required to use a specific level of hardware or operating system for WebSphere eXtreme Scale.

Setting the availability of an ObjectGrid

The availability state of an ObjectGrid instance determines which requests can be processed at any particular time.

There are four availability states:

- ONLINE
- QUIESCE
- OFFLINE
- PRELOAD

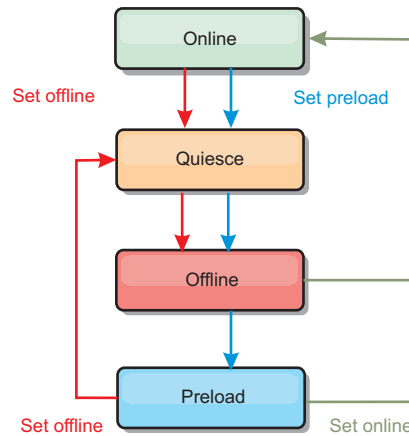


Figure 7. Availability states of an ObjectGrid

Setting the availability state

The default availability state for an ObjectGrid is ONLINE. 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 state is a transitional state. An ObjectGrid that is in QUIESCE will soon be in the OFFLINE state. While in QUIESCE, an ObjectGrid is allowed to process outstanding transactions. However, any new transactions will be rejected. An ObjectGrid can remain in QUIESCE for up to 30 seconds. After this time, the availability state will be moved to OFFLINE.

An ObjectGrid in the OFFLINE state will reject all transactions.

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 will be rejected.

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.

Retrieving 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.

Appropriate availability states for various requests

A request will be rejected if an ObjectGrid is not in the appropriate availability state to support that request. An AvailabilityException exception results whenever a request is rejected.

The initialState attribute

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 the AvailabilityState API documentation for more information.

If initialState is set on an ObjectGrid, the state must be explicitly set back to online or the ObjectGrid will remain unavailable. It will throw AvailabilityExceptions.

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 is promoted to a primary, it will not use the initialState setting. If the primary is moved for a rebalance, the initialState setting will not be 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 value if failover occurs and a new primary must be placed.

Starting stand-alone WebSphere eXtreme Scale servers

When you are running a stand-alone WebSphere eXtreme Scale configuration, the environment is comprised of catalog servers, container servers, and eXtreme Scale client processes. You must manually configure and start these processes.

Before you begin

You can start WebSphere eXtreme Scale servers in an environment that does not have WebSphere Application Server installed. If you are using WebSphere Application Server, see “Administering WebSphere eXtreme Scale with WebSphere Application Server” on page 283.

What to do next

Stop your eXtreme Scale processes. See “Stopping stand-alone eXtreme Scale servers” on page 280 for more information.

Related concepts

“Starting and stopping secure eXtreme Scale servers” on page 358
Servers often need to be secure for your deployment environment, which requires specific configuration for starting and stopping.

Related reference

“Server properties file” on page 230

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 the catalog service and container servers.

“Client properties file” on page 236

You can create a properties file based on your requirements for eXtreme Scale client processes.

“Properties file reference” on page 229

Server properties files contain settings for running your catalog servers and container servers. You can specify a server properties file for either a stand-alone or WebSphere Application Server configuration. Client property files contain settings for your client.

“startOgServer script” on page 270

The startOgServer script starts servers. You can use a variety of parameters when you start your servers to enable trace, specify port numbers, and so on.

“Logs and trace” on page 335

You can use logs and trace to monitor and troubleshoot your environment. Logs are in different locations depending on your configuration. You might need to provide trace for a server when you work with IBM support.

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 one of the existing processes. See *Starting the catalog service in WebSphere Application Server* for more information.

About this task

The catalog service can run in a single process or can include multiple catalog servers to form the catalog service domain. A catalog server grid is required in a production environment for high availability. For more information on how to configure a catalog service domain, see the information about catalog service domains in the *Product Overview*. The catalog service, whether it is placed in a grid or in a single process, is started using the startOgServer script. 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.

When you call the start command, use the startOgServer.sh script on Unix platforms or startOgServer.bat on Windows.

Procedure

- Starting 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 270. 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.

- **Starting a multi-process catalog service domain**

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`. 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 shows how to start the first of three Java virtual machines to host a catalog service:

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

- **Binding 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.

- **Naming the catalog service domain**

A domain name is not required when starting a catalog service. 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
```

- **Start a secure catalog service**

You can start a secure catalog service by providing the following arguments:

- **-clusterSecurityFile and -clusterSecurityUrl**

These arguments specify the **objectGridSecurity.xml** file, 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.

- **-serverProps**

Specifies 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**.

For an example of how to start a secure catalog service, see step 2 of the Java SE security tutorial in the *Product Overview*. For an example of the **objectGridSecurity.xml** file, see "Security descriptor XML file" on page 225.

Related concepts

“Starting and stopping secure eXtreme Scale servers” on page 358
Servers often need to be secure for your deployment environment, which requires specific configuration for starting and stopping.

Related reference

“startOgServer script” on page 270

The startOgServer script starts servers. You can use a variety of parameters when you start your servers to enable trace, specify port numbers, and so on.

“Logs and trace” on page 335

You can use logs and trace to monitor and troubleshoot your environment. Logs are in different locations depending on your configuration. You might need to provide trace for a server when you work with IBM support.

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.bat` | `sh` script or by using a Windows service to start the servers.

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 212.

Procedure

- **Start the container process from the command line.**
 1. From the command line, navigate to the bin directory:

```
cd objectgridRoot/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 190. 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 objectgridRoot/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 objectgridRoot/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 objectgridRoot/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 objectgridRoot/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 230.

Related concepts

“Starting and stopping secure eXtreme Scale servers” on page 358
Servers often need to be secure for your deployment environment, which requires specific configuration for starting and stopping.

Related reference

“startOgServer script”

The startOgServer script starts servers. You can use a variety of parameters when you start your servers to enable trace, specify port numbers, and so on.

“stopOgServer script” on page 281

The stopOgServer script stops catalog and container servers.

“Deployment policy descriptor XML file” on page 190

To configure a deployment policy, use a deployment policy descriptor XML file.

“ObjectGrid descriptor XML file” on page 196

To configure WebSphere eXtreme Scale, use an ObjectGrid descriptor XML file and the ObjectGrid API.

“Logs and trace” on page 335

You can use logs and trace to monitor and troubleshoot your environment. Logs are in different locations depending on your configuration. You might need to provide trace for a server when you work with IBM support.

startOgServer script

The startOgServer script starts servers. You can use a variety of parameters when you start your servers to enable trace, specify port numbers, and so on.

Purpose

You can use the startOgServer script to start servers.

Location

The startOgServer script is in the bin directory of the root directory, for example:

```
cd objectgridRoot/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

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
```

-quorum true|false

-clusterSecurityFile <cluster security xml file>

Specifies the location of the security descriptor XML file.

-clusterSecurityUrl <cluster security xml URL>

-domain <domain name>

-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.

-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.

-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`.

-JMXServicePort <port>

Default: 1099

Specifies the port number for communication with Java Management Extensions (JMX).

-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.

-script <script file>

Creates a custom script for commands you specify to start catalog servers or containers and then parameterize or edit as you require.

-jvmArgs <JVM arguments>

Specifies a set of JVM arguments. Every parameter after the **-jvmArgs** parameter 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

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>

Default: localhost:2809

-deploymentPolicyFile <deployment policy xml file>

Specifies the path to the deployment policy file.

Example: ../xml/SimpleDP.xml

-deploymentPolicyUrl <deployment policy url>

- objectgridFile <ObjectGrid descriptor xml file>**
Specifies the path to the ObjectGrid descriptor file.
- objectgridUrl <ObjectGrid descriptor url>**
- 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.
- serverProps <server properties file>**
Specifies the path to the server property file.
Example: ../security/server.props
- zone <zone name>**
Specifies the zone to use for all of the containers within the server.
- 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.
- script <script file>**
Creates a custom script for commands you specify to start catalog servers or containers and then parameterize or edit as you require.
- jvmArgs <JVM arguments>**
Specifies a set of JVM arguments. Every parameter after the **-jvmArgs** parameter is used to start the server Java virtual machine (JVM). When the **-jvmArgs** parameter is used, ensure that it is the last optional script argument specified.
Example:-jvmArgs -Xms256M -Xmx1G

Related concepts

“Starting and stopping secure eXtreme Scale servers” on page 358
Servers often need to be secure for your deployment environment, which requires specific configuration for starting and stopping.

Related tasks

“Starting container processes” on page 267

You can start eXtreme Scale from the command line using a deployment topology or using a `server.properties` file.

“Starting stand-alone WebSphere eXtreme Scale servers” on page 262

When you are running a stand-alone WebSphere eXtreme Scale configuration, the environment is comprised of catalog servers, container servers, and eXtreme Scale client processes. You must manually configure and start these processes.

“Starting a stand-alone catalog service” on page 263

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.

Related reference

“Distributed eXtreme Scale grid configuration” on page 89

Use the deployment policy descriptor XML file and the objectgrid descriptor XML file to manage your topology.

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 WebSphere Extended Deployment DataGrid 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.”

Using the embedded server API

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)
    }
}
}
}

```

Configuring ports in stand-alone mode

A Java virtual machine that hosts a catalog service instance requires four ports. Two ports are used for internal use, the third port is used for clients and container shards to communicate with Internet Inter-ORB Protocol (IIOP), and the fourth port is used for Java Management Extensions (JMX) communication.

About this task

Catalog service Java virtual machine (JVM) end points

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>
```

With the **-catalogServiceEndPoints** command line option, you can configure two ports per server. The IIOP ports are configured using the following command line options:

```
-listenerHost <host_name>
-listenerPort <port>
```

When each catalog service JVM is started, specify the complete set of catalog service end points along with a single listener port for that JVM.

Container JVM end points

The container Java virtual machines use two ports. One port is for internal use and the other port is for IIOP communication. The container Java virtual machines normally automatically find unused ports and then configure themselves to use two dynamically created ports. This automatic process minimizes configuration. However, when firewalls are being used or you want explicitly configure ports, you can use a command line option to specify the Object Request Broker (ORB) port to use:

```
-listenerHost <host_name>
-listenerPort <port>
```

With these command line options, you can specify the host name (important for binding to correct network card) and the port to be specified for that JVM. You can specify the internal port with the **-haManagerPort** command line argument. However, for the simplest configuration, you can let the run time choose the ports.

Procedure

1. 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
```

2. Start the second catalog server on hostB. An example of the command follows:


```
./start0gServer.sh cs2 -listenerHost hostB -listenerPort 2809  
-catalogServiceEndPoints cs1:hostA:6601:6611,cs2:hostB:6601:6611
```

Clients need only 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
```

To start a container JVM to use the example catalog service, use the following command:

```
./start0gServer.sh c0 -catalogServiceEndPoints hostA:2809,hostB:2809
```

Stopping stand-alone eXtreme Scale servers

You can use the stop0gServer script to stop server processes.

Procedure

- Stop eXtreme Scale container processes.
 1. From the command line, navigate to the bin directory.

```
cd objectGridRoot/bin
```
 2. Run the **stop0gServer** script to stop the server. The following example stops the c0 server:

```
stop0gServer c0 -catalogServiceEndPoints MyServer1.company.com:2809
```

Use the same script to stop multiple servers with a comma-delimited list as follows:

```
stop0gServer c0,c1,c2 -catalogServiceEndPoints MyServer1.company.com:2809
```

Attention: 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 **-listenerHost** and **-listenerPort** options to specify the host and port for ORB binding or accepts the default binding. When you are stopping a container, use the **-catalogServiceEndPoints** option to reference the values 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.

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, use the **-catalogServiceEndPoints** option to specify the ports that are necessary for static server configuration.

- Stop eXtreme Scale catalog service processes.
 1. From the command line, navigate to the bin directory.

```
cd objectGridRoot/bin
```
 2. Run the **stop0gServer** script to stop the server.

```
stop0gServer.sh catalogServer -bootstrap MyServer1.company.com:6601
```

Identify the process to stop with the catalogServer argument, the first argument that is passed to the script. For this example, assume that the catalog service was started on the MyServer1.company.com domain with a clientAccessPort value of 6601.

- 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 objectGridRoot/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 337.

Related concepts

“Hardware and software requirements” on page 7

You are not required to use a specific level of hardware or operating system for WebSphere eXtreme Scale.

Related reference

“stopOgServer script”

The stopOgServer script stops catalog and container servers.

stopOgServer script

The stopOgServer script stops catalog and container servers.

Purpose

Use the **stopOgServer** script to stop a server. You must provide the name of the server and its catalog service endpoints.

Location

The **stopOgServer** script is in the bin directory of the root directory, for example:

```
cd objectgridRoot/bin
```

Usage

To stop a catalog server:

Windows

```
stopOgServer.bat <server> -catalogServiceEndPoints  
<csHost:csListenerPort,csHost:csListenerPort> [options]
```

UNIX

```
stopOgServer.sh <server> -catalogServiceEndPoints  
<csHost:csListenerPort,csHost:csListenerPort> [options]
```

To stop an ObjectGrid container server:

Windows

```
stopOgServer.bat <server> -catalogServiceEndPoints  
<csHost:csListenerPort,csHost:csListenerPort> [options]
```

UNIX

```
stopOgServer.sh -catalogServiceEndPoints  
<csHost:csListenerPort,csHost:csListenerPort> [options]
```

Options

-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
```

-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 236 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>

Every parameter after the **-jvmArgs** option is used to start the server Java virtual machine (JVM). When the **-jvmArgs** option is used, ensure that it is the last optional script argument specified.

Related tasks

“Starting container processes” on page 267

You can start eXtreme Scale from the command line using a deployment topology or using a `server.properties` file.

“Stopping stand-alone eXtreme Scale servers” on page 280

You can use the `stop0gServer` script to stop server processes.

Related reference

“Distributed eXtreme Scale grid configuration” on page 89

Use the deployment policy descriptor XML file and the objectgrid descriptor XML file to manage your topology.

Administering 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.

Related reference

“Server properties file” on page 230

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 the catalog service and container servers.

“Client properties file” on page 236

You can create a properties file based on your requirements for eXtreme Scale client processes.

“Properties file reference” on page 229

Server properties files contain settings for running your catalog servers and container servers. You can specify a server properties file for either a stand-alone or WebSphere Application Server configuration. Client property files contain settings for your client.

Starting the catalog service process in a WebSphere Application Server environment

WebSphere eXtreme Scale catalog servers can automatically start in a WebSphere Application Server or WebSphere Application Server Network Deployment environment. You can configure the catalog service to run in any process within the WebSphere cell. A single-server non-clustered catalog service is acceptable for development environments. For a production environment, you should use a catalog service grid with multiple catalog servers.

Before you begin

You must install WebSphere Application Server and WebSphere eXtreme Scale. See Chapter 4, “Installing and deploying WebSphere eXtreme Scale,” on page 35 for more information.

About this task

The catalog service process can run in a WebSphere Application Server process.

For the base WebSphere Application Server, the catalog service runs in the application server process. By default, it is *not* enabled to start automatically.

For WebSphere Application Server Network Deployment, the catalog service runs in the deployment manager process automatically, but you can configure it to run in one or more node-agent or application-server processes.

Defining the `catalog.services.cluster` property

By defining the `catalog.services.cluster` cell, node or server custom property, you can customize your catalog service environment. The environment can consist of a single catalog server or a cluster (grid) of catalog servers. A single catalog server is recommended for a development environment, and a grid of catalog servers for a production environment. This key-value pair identifies which server process will be enabled.

Set the value of the custom property in the following form:

```
catalog.services.cluster ::= hosted_endpoints
hosted_endpoints        ::= serverName ":" endpoints {" ," hosted_endpoints}
endpoints               ::= hostName ":" clientPort ":" peerPort ":" listenerPort
```

Each attribute for the custom property value is defined as follows:

serverName

Specifies the fully qualified name of the WebSphere process, such as the `cellName`, `nodeName`, and `serverName` of the server that hosts the catalog service. If the `serverName` is not the local process, clients and containers use this information to connect to the catalog service.

The following is an example for a remote catalog service cluster:

```
server1:host.local.domain:6601:6602:2809,server2:host.local.domain:6601:6602:2809
```

hostName

Specifies the name of the hosting server.

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 and can be anything you choose in a WebSphere Application Server environment.

listenerPort

The `listenerPort` must match the `BOOTSTRAP_ADDRESS` value that is defined in the WebSphere server configuration.

A process that matches the `serverName` property enables the catalog service on that server process.

Ensure that you do not collocate the catalog service with eXtreme Scale containers in a production environment. Include the catalog service in multiple node agent processes or an application server cluster that is not hosting an eXtreme Scale application.

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. The following are examples: When servers with the name `server1` on each node are used as a grid of catalog services, or where multiple node agents are used to form a grid of catalog services.

Examples:

- For a non-clustered catalog service, define a single value on an example `server1` as follows:

```
cellA\node1\server1:<hostname>:<clientport>:<peerport>:<listenerport>
```

- For a clustered catalog service as in the following example with 3 servers, define the property in a comma-delimited list:

```
cellA\node1\server1:<hostname1>:<clientport>:<peerport>:<listenerport>,  
cellA\node2\server1:<hostname2>:<clientport>:<peerport>:<listenerport>,  
cellA\node3\server1:<hostname3>:<clientport>:<peerport>:<listenerport>
```

In the previous example, `server1` is defined on all 3 hosts: `hostname1`, `hostname2`, and `hostname3`. This property specifies the fully qualified name of the WebSphere process, such as the `cellName`, `nodeName`, and `serverName` of the server that hosts the catalog service. Also, `clientport` and `peerport` are non-conflicting ports and `listenerport` must match the `BOOTSTRAP_ADDRESS` for the servers.

With some exceptions, the `catalog.services.cluster` custom property is similar to the `-catalogServiceEndpoints` parameter that is used when starting a stand-alone catalog server with the addition of the `-listenerPort` parameter.

Remember: The format and requirements for the custom property file are similar, but depending on your WebSphere Application Server deployment (base or Network Deployment), you must choose a different menu path to set the parameters, as follows.

Procedure

- **Starting a catalog server in base WebSphere Application Server**

When WebSphere eXtreme Scale is integrated into a non-federated WebSphere Application Server profile, the catalog service will not automatically start except in the following two circumstances:

1. An application that is configured to automatically start an eXtreme Scale container: Both the catalog service and container will start automatically. See *Starting eXtreme Scale containers automatically in WebSphere eXtreme Scale applications* for more information.
2. If the `catalog.services.cluster` custom property is defined.

If you need to start the catalog service without installing an application, define the custom property `catalog.services.cluster` to activate the service.

7.0.0.0 FIX 2+ To create the custom property in the administrative console select **Servers > Server types > WebSphere application servers > server1 > Administration > Custom properties > New** .

For Version 6.x, select **Servers > Application servers > server1 > Administration > Custom properties > New**.

In either version, specify the name of the new property as `catalog.services.cluster` and the value in the appropriate form using the defined attributes as described previously. This configuration only supports a non-clustered catalog service. Only a single value can be defined for this property.

For more information, see the topic on starting containers in a WebSphere Application Server environment.

This feature simplifies unit testing in development environments such as Rational® Application Developer because you do not need to explicitly start a catalog service.

- **Starting a catalog service grid in WebSphere Application Server Network Deployment**

If WebSphere eXtreme Scale is installed on WebSphere Application Server Network Deployment, the catalog service automatically starts in the deployment manager process if it is augmented to allow for quick development out of the box.

An eXtreme Scale catalog server grid can be explicitly defined on a deployment manager, node agent, or application server process using the `catalog.services.cluster` custom property, which is set in the WebSphere Application Server cell, node or server configuration.

Attention: Do not collocate the catalog services with eXtreme Scale containers in a production environment. Include the catalog service in multiple node agent processes or an application server grid that is not hosting an eXtreme Scale application.

To create the custom property in the administrative console click **System administration > Cell > Custom properties > New**. Specify the name as `catalog.services.cluster` and the value in the appropriate form, using the defined attributes. The following is an example of a valid value:

```
cell1A\node1\nodeagent:host.local.domain:6600:6601:2809,cell1A\node2\nodeagent:host.local.domain:6600:6601:2809
```

After you set the custom property, you must restart each identified server. When you restart these servers, the catalog service grid automatically starts.

The catalog grid uses the core group mechanisms of WebSphere Application Server. As a result, the members of a catalog grid cannot span the boundaries of a core group, and a catalog grid therefore cannot span cells. However, eXtreme Scale can span cells by connecting to a catalog server across cell boundaries, such as either a stand-alone catalog grid or a catalog grid embedded in another cell.

Remember:

You can define the `catalog.services.cluster` property for a cell, node or server. Cell and node properties are used for WebSphere Application Server Network Deployment. Server properties are used for Network Deployment and base WebSphere Application Server. If multiple locations are set, the server properties override the cell and node properties, and node properties override the cell property.

Starting eXtreme Scale containers automatically in WebSphere Application Server applications

Container processes 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 WebSphere eXtreme Scale within a Java EE server. A Java EE application is typically a single Enterprise Archive (EAR) file with one or more Enterprise JavaBeans (EJB) or Web archive (WAR) modules deployed in it.

WebSphere eXtreme Scale watches for each module start and looks for eXtreme Scale XML files. If WebSphere eXtreme Scale detects that a module starts with the XML files, it registers the application server as an eXtreme Scale container Java virtual machine (JVM) with the catalog service. By registering the containers with the catalog service, the same application can be deployed in different data grids and still be treated as a single data grid by the catalog service. The catalog service is not concerned with cells, data grids or dynamic data grids. Everything either is an eXtreme Scale container JVM or is not. A single logical data grid can span multiple WebSphere Extended Deployment 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 a 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.xml` file, commonly referred to as an eXtreme Scale descriptor XML file.
- An `objectGridDeployment.xml` file, commonly referred to as a deployment descriptor XML file.
- An `entity.xml` file, if entities are used (optional). The `entity.xml` file name must match the name that is specified in the `objectGrid.xml` file.

The eXtreme Scale 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 you plan to try an application with entities using one eXtreme Scale server, set the `minSyncReplicas` value to 0 in the deployment descriptor XML file to avoid a `CWPRJ1005E: Error resolving entity association exception caused by The EntityMetadata repository is not available. Timeout threshold reached` message.

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 using different ClassLoaders. If application A stores a Person object in an eXtreme Scale 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 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, but this complicates deployment and is not recommended.

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 as a 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 and call the ObjectGridManager.getObjectGrid(ccc, "grid name") method to gain a reference to the eXtreme Scale. If any application objects are stored in Maps, verify that that those objects are present in a common ClassLoader.

Java Platform, Standard Edition clients or clients outside the cell can connect using the bootstrap IIOP port of the catalog service (default in Websphere is the deployment manager) to obtain a ClientClusterContext and then obtain a named eXtreme Scale the usual way.

Entity manager

The entity manager helps because the Tuples are stored in the Maps, not the application objects, so that there are less class loader problems of concern. Plug-ins can be a problem, however. Also note that a client override eXtreme Scale descriptor XML file is always required when connecting to an eXtreme Scale that has entities defined: ObjectGridManager.connect("host:port[,host:port], null, objectGridOverride) or ObjectGridManager.connect(null, objectGridOverride).

Configuring ports in a WebSphere Application Server environment

The catalog service runs a single instance within the deployment manager by default, and uses the Internet Inter-ORB Protocol (IIOP) bootstrap port for the deployment manager Java virtual machine.

About this task

Web applications or Enterprise JavaBeans (EJB) applications within a cell can connect to grids that are within the same cell using the `null,null` connect call instead of specifying catalog service bootstrap ports.

If the catalog service grid in WebSphere Application Server is hosted by the deployment manager, clients outside the cell (including Java Platform, Standard Edition clients) must connect to the catalog service using the deployment manager host name and the IIOP bootstrap port.

When the catalog service runs in WebSphere Application Server cells while the clients run outside the cells, you must locate client connect ports by locating the cell custom properties with the `catalog.services.cluster` name. If this entry is found, then use the entry to connect the clients to the catalog service, but if not, use the IIOP port on the deployment manager for the client connection. If your clients are in WebSphere Application Server cells, you can retrieve ports from the `CatalogServerProperties` interface directly.

eXtreme Scale reuses the high availability manager Distribution and Consistency Services (DCS) ports for group membership. Java Management Extensions (JMX) ports are also reused.

Chapter 8. Monitoring your deployment environment

You can use APIs, MBeans, logs, and utilities to monitor the performance of your application environment.

Related concepts

“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.

“Vendor tools” on page 322

WebSphere eXtreme Scale can be monitored using several popular enterprise monitoring solutions. Plug-in agents are included for IBM Tivoli Monitoring and Hyperic HQ, which monitor WebSphere eXtreme Scale using publicly accessible management beans. CA Wily Introscope uses Java method instrumentation to capture statistics.

Related reference

“Using Managed Beans (MBeans) to administer your environment” on page 312

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, objectGrid, server, replication group, or replication group member.

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 eXtreme Scale.

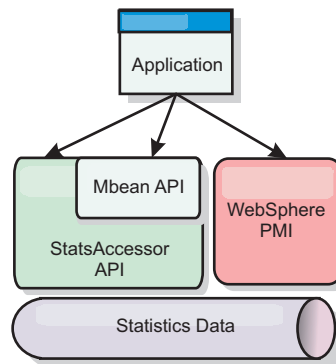


Figure 8. Statistics overview

Each of these APIs offer a view into the statistics tree, but are used for different reasons:

- **Statistics API:** Use the Statistics API as a generic lookup mechanism for data in the internal statistic tree. You can use the Statistics API for clients or embedded applications. You cannot use the Statistics API if you have a distributed eXtreme Scale.

- **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 eXtreme Scale.
- **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 performance with WebSphere Application Server PMI" on page 299 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.

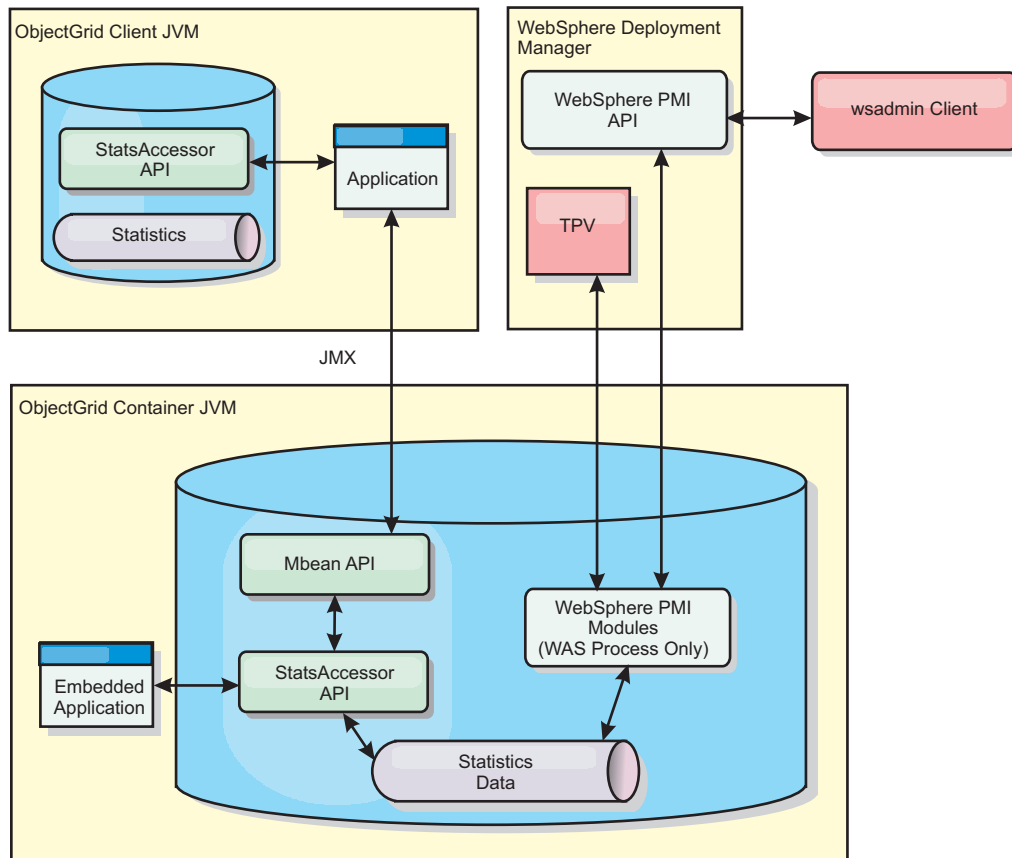


Figure 9. MBean overview

See “Using the xsAdmin sample utility” on page 319 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 333
- “Monitoring eXtreme Scale applications with CA Wily Introscope” on page 329

Related tasks

Chapter 8, “Monitoring your deployment environment,” on page 291

You can use APIs, MBeans, logs, and utilities to monitor the performance of your application environment.

“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.

“Monitoring performance with WebSphere Application Server PMI” on page 299
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.

“Enabling PMI” on page 300

You can use WebSphere Application Server Performance Monitoring Infrastructure (PMI) to enable or disable statistics at any level. For example, you can choose to enable the map hit rate statistic for a particular map, but not the number of entry statistic or the loader batch update time statistic. You can enable PMI in the administrative console or with scripting.

“Retrieving PMI statistics” on page 302

By retrieving PMI statistics, you can see the performance of your eXtreme Scale applications.

“Using the xsAdmin sample utility” on page 319

With the xsAdmin sample 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.

“Monitoring eXtreme Scale with Hyperic HQ” on page 333

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.

“Monitoring with the IBM Tivoli Enterprise Monitoring Agent for WebSphere eXtreme Scale” on page 323

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.

Related reference

“Using Managed Beans (MBeans) to administer your environment” on page 312

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, objectGrid, server, replication group, or replication group member.

“PMI modules” on page 304

You can monitor the performance of your applications with the performance monitoring infrastructure (PMI) modules.

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 data grid to force data to be collected for monitoring. To collect useful data for statistics, you must send transactions to the data grid. The following code excerpt inserts a record into MapA, which is in ObjectGridA. Because the statistics are at the ObjectGrid level, any map within the ObjectGrid yields the same results.

```
public static void main(String[] args) {
    // retrieve a handle to the StatsAccessor
```

```

StatsAccessor accessor = StatsAccessorFactory.getStatsAccessor();

// Set the spec via the static field
StatsSpec spec = new StatsSpec(StatsSpec.OG_ALL);
accessor.setStatsSpec(spec);

ObjectGridManager manager =
ObjectGridmanagerFactory.getObjectGridManager();
ObjectGrid grid = manager.getObjectGrid("ObjectGridA");
Session session = grid.getSession();
Map map = session.getMap("MapA");

// Drive insert
session.begin();
map.insert("SomeKey", "SomeValue");
session.commit();
}

```

4. Query a StatsFact by using the StatsAccessor API. Every statistics path is associated with a StatsFact interface. The StatsFact interface is a generic placeholder that is used to organize and contain a StatsModule object. Before you can access the actual statistics module, the StatsFact object must be retrieved.

```

public static void main(String[] args)
{
    // retrieve a handle to the StatsAccessor
    StatsAccessor accessor = StatsAccessorFactory.getStatsAccessor();

    // Set the spec via the static field
    StatsSpec spec = new StatsSpec(StatsSpec.OG_ALL);
    accessor.setStatsSpec(spec);

    ObjectGridManager manager =
ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid grid = manager.getObjectGrid("ObjectGridA");
Session session = grid.getSession();
Map map = session.getMap("MapA");

// Drive insert
session.begin();
map.insert("SomeKey", "SomeValue");
session.commit();

// Retrieve StatsFact

StatsFact fact = accessor.getStatsFact(new String[] {"EmployeeGrid"},
StatsModule.MODULE_TYPE_OBJECT_GRID);
}

```

5. Interact with the StatsModule object. The StatsModule object is contained within the StatsFact interface. You can obtain a reference to the module by using the StatsFact interface. Since the StatsFact interface is a generic interface, you must cast the returned module to the expected StatsModule type. Because this task collects eXtreme Scale statistics, the returned StatsModule object is cast to an OGStatsModule type. After the module is cast, you have access to all of the available statistics.

```

public static void main(String[] args) {

    // retrieve a handle to the StatsAccessor
    StatsAccessor accessor = StatsAccessorFactory.getStatsAccessor();

    // Set the spec via the static field
    StatsSpec spec = new StatsSpec(StatsSpec.OG_ALL);
}

```

```

        accessor.setStatsSpec(spec);

        ObjectGridManager manager =
ObjectGridmanagerFactory.getObjectGridManager();
        ObjectGrid grid = manager.getObjectGrid("ObjectGridA");
        Session session = grid.getSession();
        Map map = session.getMap("MapA");

        // Drive insert
        session.begin();
        map.insert("SomeKey", "SomeValue");
        session.commit();

        // Retrieve StatsFact
        StatsFact fact = accessor.getStatsFact(new String[] {"EmployeeGrid"},
StatsModule.MODULE_TYPE_OBJECT_GRID);

        // Retrieve module and time
        OGStatsModule module = (OGStatsModule)fact.getStatsModule();
        ActiveTimeStatistic timeStat =
module.getTransactionTime("Default", true);
        double time = timeStat.getMeanTime();
    }

```

Related concepts

“Statistics overview” on page 291

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.

“Vendor tools” on page 322

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.

“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.

Related reference

“Using Managed Beans (MBeans) to administer your environment” on page 312

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, objectGrid, server, replication group, or replication group member.

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.

Related tasks

“Monitoring with the statistics API” on page 294

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.

Monitoring performance 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” on page 300 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 302 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 314.

Related concepts

“Statistics overview” on page 291

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.

“Vendor tools” on page 322

WebSphere eXtreme Scale can be monitored using several popular enterprise monitoring solutions. Plug-in agents are included for IBM Tivoli Monitoring and Hyperic HQ, which monitor WebSphere eXtreme Scale using publicly accessible management beans. CA Wily Introscope uses Java method instrumentation to capture statistics.

Related reference

“Using Managed Beans (MBeans) to administer your environment” on page 312

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, objectGrid, server, replication group, or replication group member.

“PMI modules” on page 304

You can monitor the performance of your applications with the performance monitoring infrastructure (PMI) modules.

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 `install_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.

Related concepts

“Statistics overview” on page 291

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.

Related reference

“PMI modules” on page 304

You can monitor the performance of your applications with the performance monitoring infrastructure (PMI) modules.

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 300 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 304.

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 `install_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.

Related concepts

“Statistics overview” on page 291

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.

Related reference

“PMI modules”

You can monitor the performance of your applications with the performance monitoring infrastructure (PMI) modules.

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.

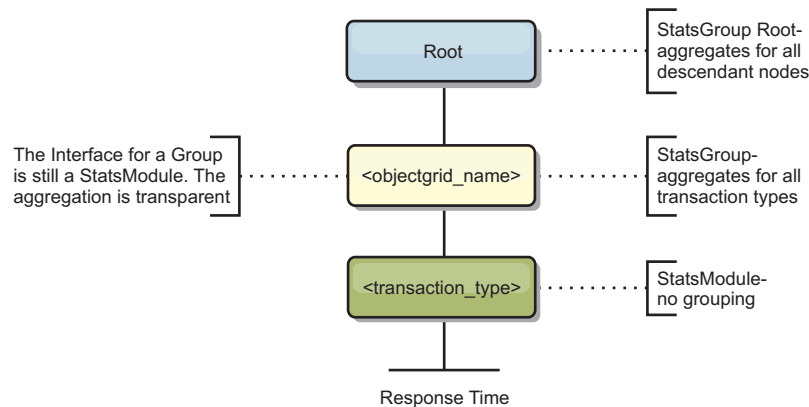


Figure 10. ObjectGridModule module structure

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.

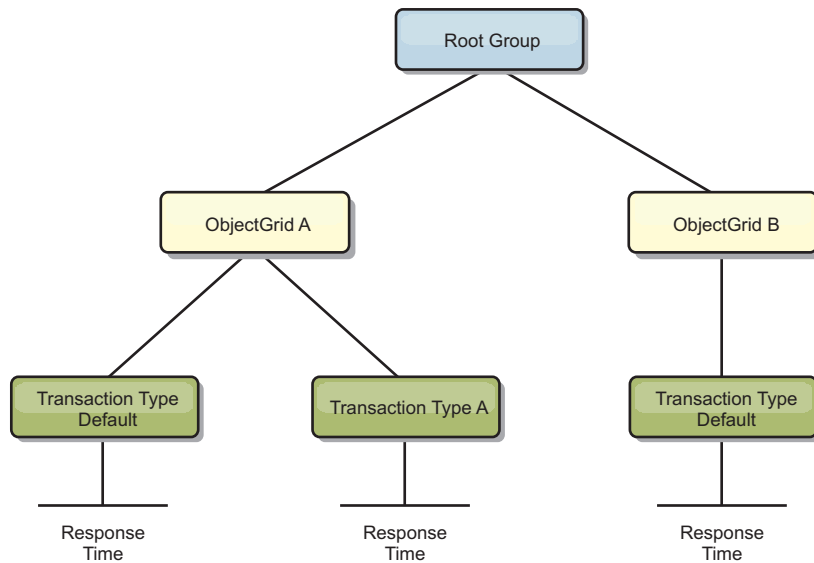


Figure 11. ObjectGridModule module structure example

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:

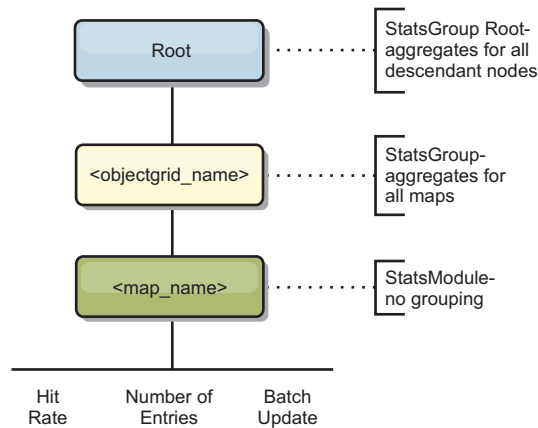
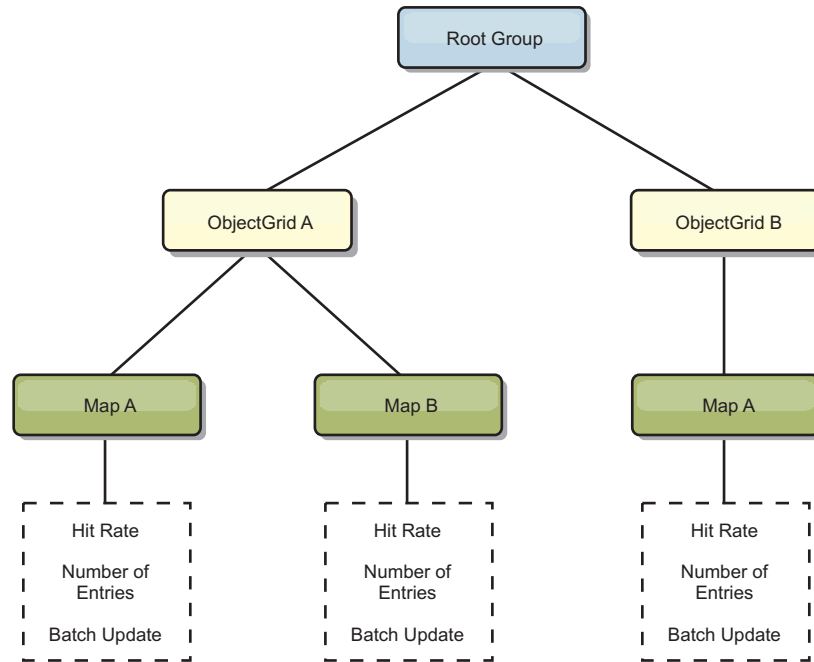


Figure 12. mapModule structure

The following diagram shows an example of the mapModule structure:

Figure 13. 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:

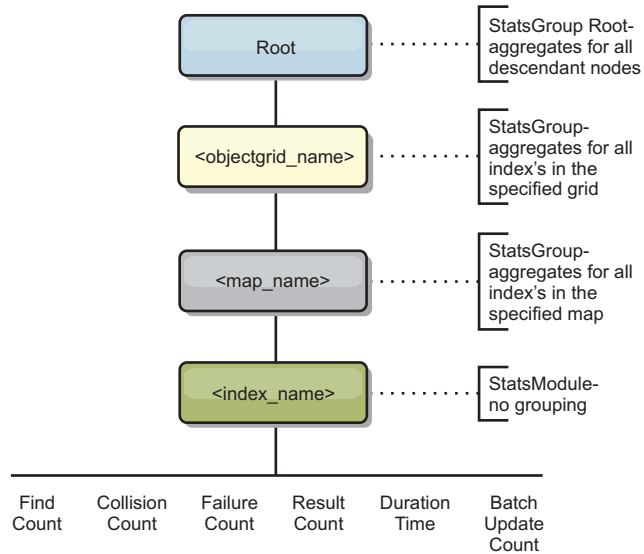


Figure 14. hashIndexModule module structure

The following diagram shows an example of the hashIndexModule structure:

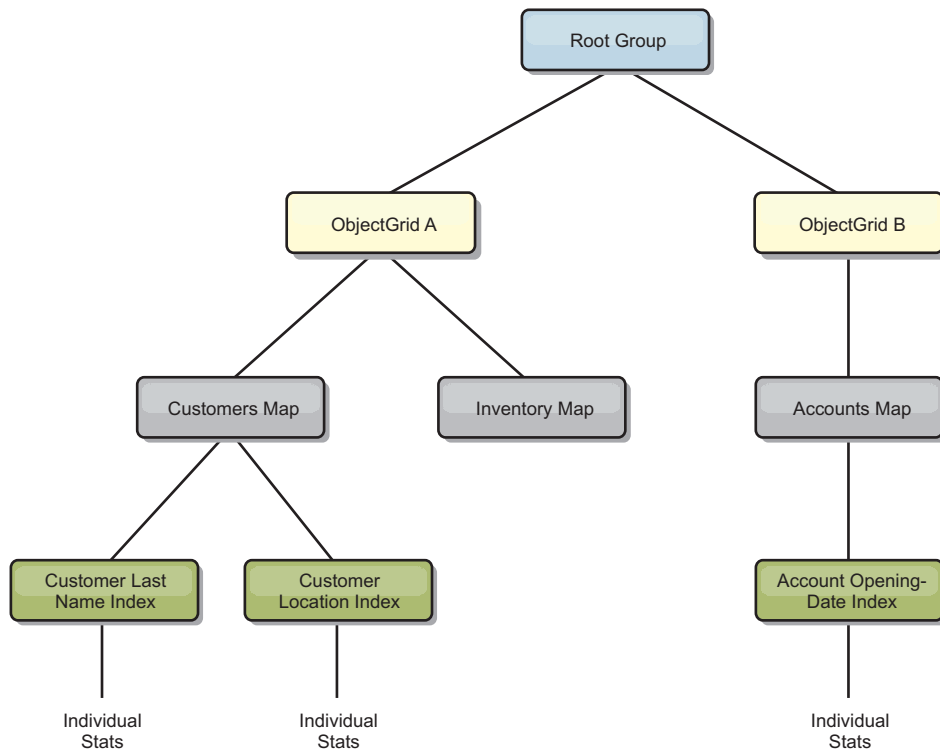


Figure 15. hashIndexModule module structure example

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 the three listed statistics.

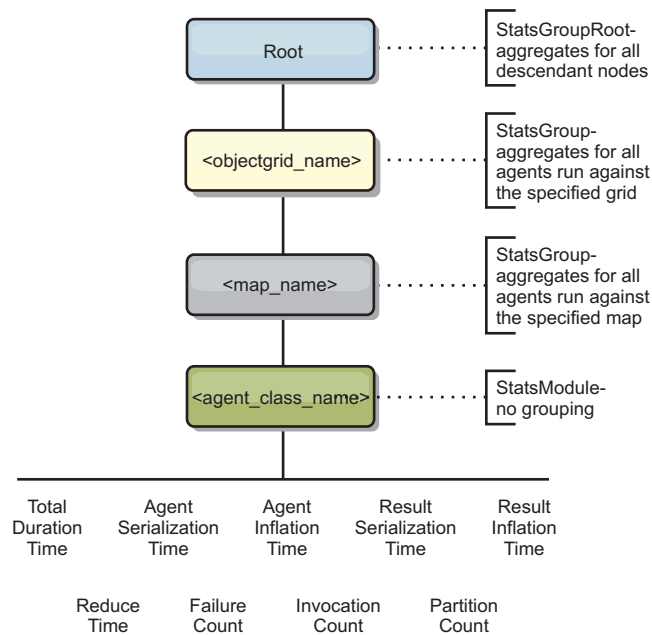


Figure 16. agentManagerModule structure

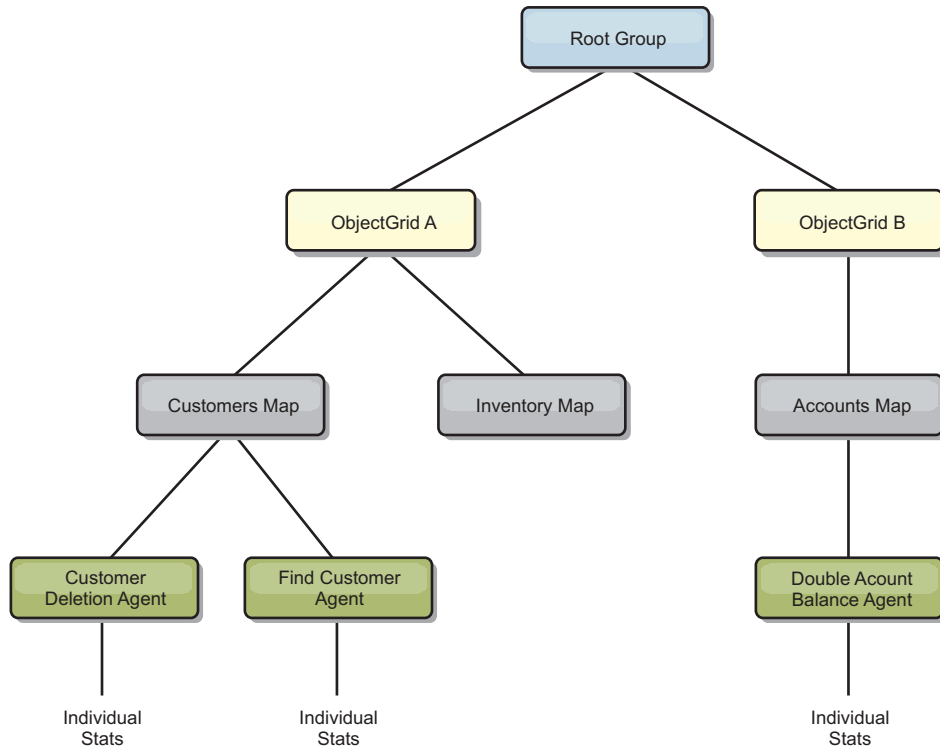


Figure 17. agentManagerModule structure example

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.

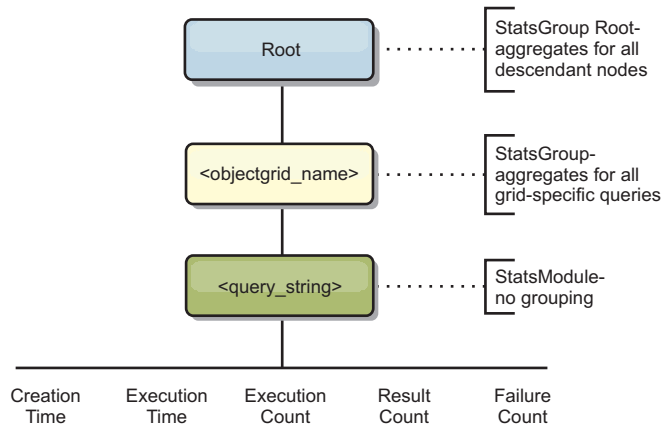


Figure 18. queryModule structure

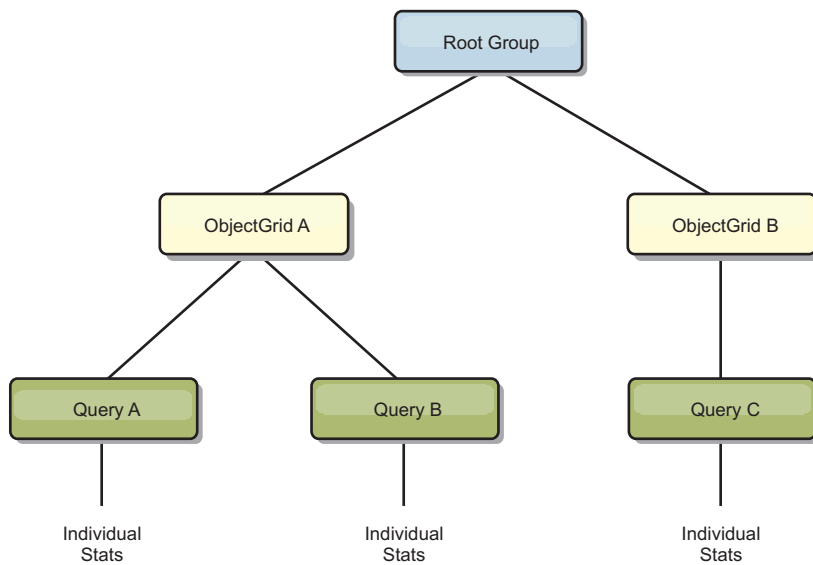


Figure 19. QueryStats.jpg queryModule structure example

Related concepts

“Statistics overview” on page 291

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.

Related tasks

“Monitoring performance with WebSphere Application Server PMI” on page 299
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.

“Enabling PMI” on page 300

You can use WebSphere Application Server Performance Monitoring Infrastructure (PMI) to enable or disable statistics at any level. For example, you can choose to enable the map hit rate statistic for a particular map, but not the number of entry statistic or the loader batch update time statistic. You can enable PMI in the administrative console or with scripting.

“Retrieving PMI statistics” on page 302

By retrieving PMI statistics, you can see the performance of your eXtreme Scale applications.

“Using the xsAdmin sample utility” on page 319

With the xsAdmin sample 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.

Using Managed Beans (MBeans) to administer your environment

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, objectGrid, server, replication group, or replication group member.

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` .

Related concepts

“Statistics overview” on page 291

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.

“Vendor tools” on page 322

WebSphere eXtreme Scale can be monitored using several popular enterprise monitoring solutions. Plug-in agents are included for IBM Tivoli Monitoring and Hyperic HQ, which monitor WebSphere eXtreme Scale using publicly accessible management beans. CA Wily Introscope uses Java method instrumentation to capture statistics.

Related tasks

Chapter 8, “Monitoring your deployment environment,” on page 291

You can use APIs, MBeans, logs, and utilities to monitor the performance of your application environment.

“Monitoring with the statistics API” on page 294

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.

“Monitoring performance with WebSphere Application Server PMI” on page 299
WebSphere eXtreme Scale supports Performance Monitoring Infrastructure (PMI) when running in a WebSphere Application Server or WebSphere Extended Deployment application server. PMI collects performance data on runtime applications and provides interfaces that support external applications to monitor performance data. You can use the administrative console or the wsadmin tool to access monitoring data.

Monitoring 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 230.

- **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.”

- **Access MBean statistics programmatically.**

For more information, see “Accessing Managed Beans (MBeans) programmatically.”

Example

For an example of how to use managed beans, see “Using the xsAdmin sample utility” on page 319.

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 20. *CollectPlacementPlan.java*

A few notes regarding the sample program:

- The JMXServiceURL for the catalog service is always of the 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 provided via the `-JMXServicePort` option when starting the catalog service. If no port is specified, the default is 1099.

- In order for the ObjectGrid or map statistics to be enabled, you need to specify the following property in the server properties file when starting an ObjectGrid container: statsSpec=all=enabled
- To disable the MBeans running in the ObjectGrid containers, specify the following property in the server properties file: enableMBeans=false.

An example of the output follows. This output indicates that two containers 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 21. *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="myhost.mycompany.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 22. *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 `dir_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 352.

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 `dir_wxs_home/samples/xsadmin.jar` file in a stand-alone installation, or in the `dir_wxs_home/xsadmin.jar` file in a WebSphere Application Server installation. See “Using the xsAdmin sample utility” for more information about the operations you can complete with the **xsAdmin** sample application.

You can also find more information about MBeans in the `com.ibm.websphere.objectgrid.management` package.

Using the xsAdmin sample utility

With the xsAdmin sample utility, you can format and display textual information about your WebSphere eXtreme Scale topology. The sample utility provides a method for parsing and discovering current deployment data, and can be used as a foundation for writing custom utilities.

Before you begin

You must have WebSphere eXtreme Scale installed.

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 `dir_wxs_home/samples/xsadmin.jar` file in a stand-alone installation, or in the `dir_wxs_home/xsadmin.jar` file in a WebSphere Application Server installation.

You can use the xsAdmin sample utility to provide feedback on the current layout and specific state of the grid, such as map content. In this example, the layout of the grid in this task consists of a single grid, named *ObjectGridA* with one defined map, named *MapA*, belonging to the mapset, entitled *MapSetA*. This example demonstrates how you can display all active containers within a grid and print out filtered metrics regarding the map size of *MapA*. 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. Navigate to the bin directory.

```
cd objectGridRoot/bin
```

3. Launch 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 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:

```
This administrative utility is provided as a sample only and is not to be
considered a fully supported component of the WebSphere eXtreme Scale product
Connecting to Catalog service at localhost:1099
*** Show all online containers for grid - ObjectGridA & mapset - MapSetA
Host: 192.168.0.186
  Container: server1_C-0, Server:server1, Zone:DefaultZone
    P:0 Primary
    Num containers matching = 1
    Total known containers = 1
    Total known hosts = 1
```

- **To display the number of entries in all the maps for a grid, 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:

```
This administrative utility is provided as a sample only and is not to be
considered a fully supported component of the WebSphere eXtreme Scale product

Connecting to Catalog service at localhost:1099

****Displaying Results for Grid - ObjectGridA, MapSet - MapSetA****

*** Listing Maps for server1 ***
  Map Name: MapA  Partition #: 0  Map Size: 0  Shard Type: Primary
  Server Total: 0
```

- **To specify the JMX port for the catalog service, run the following command:** The xsAdmin sample utility connects to the MBean server that is running on a catalog server. A catalog server can run in a standalone 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.

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 size of the specified map is displayed. An example of the output follows:

This administrative utility is provided as a sample only and is not to be considered a fully supported component of the WebSphere eXtreme Scale product

Connecting to Catalog service at 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 script included in the *was_root/bin* directory, which is available when installing WebSphere eXtreme Scale on WebSphere Application Server or WebSphere Application Server Network Deployment.

- a. Navigate to the WebSphere Application Server bin directory:

```
cd wasRoot/bin
```

- b. 2. Launch the xsAdmin utility using the following command:

```
xsadmin.sh -g ObjectGridA -m MapSetA -mapsizes -fm MapA -dmgr
```

Windows

```
xsadmin.bat -g ObjectGridA -m MapSetA -mapsizes -fm MapA -dmgr
```

- c. The size of the specified map is displayed.

This administrative utility is provided as a sample only and is not to be considered a fully supported component of the WebSphere eXtreme Scale product

Connecting to Catalog service at localhost: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

Related concepts

“Statistics overview” on page 291

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.

“Catalog server quorums” on page 18

Quorum is the minimum number of catalog servers necessary to conduct placement operations for the data grid. The minimum number is the full set of catalog servers unless quorum has been overridden.

Related reference

“PMI modules” on page 304

You can monitor the performance of your applications with the performance monitoring infrastructure (PMI) modules.

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.

Related tasks

Chapter 8, “Monitoring your deployment environment,” on page 291

You can use APIs, MBeans, logs, and utilities to monitor the performance of your application environment.

“Monitoring with the statistics API” on page 294

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.

“Monitoring performance with WebSphere Application Server PMI” on page 299

WebSphere eXtreme Scale supports Performance Monitoring Infrastructure (PMI) when running in a WebSphere Application Server or WebSphere Extended Deployment application server. PMI collects performance data on runtime applications and provides interfaces that support external applications to monitor performance data. You can use the administrative console or the wsadmin tool to access monitoring data.

“Monitoring eXtreme Scale with Hyperic HQ” on page 333

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.

“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.

Related reference

“Using Managed Beans (MBeans) to administer your environment” on page 312

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, objectGrid, server, replication group, or replication group member.

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.
- 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 # ./itmcmd config -A xt
Agent configuration started...
Enter instance name (default is: ): inst1
Edit "Monitoring Agent for WebSphere eXtreme Scale" settings? [ 1=Yes, 2=No ] (default is: 1): 1
Edit 'Java' settings? [ 1=Yes, 2=No ] (default is: 1): 1
Java home (default is: C:\Program Files\IBM\Java50): /opt/WAS61/java
Java trace level [ 1=Error, 2=Warning, 3=Information, 4=Minimum Debug, 5=Medium Debug, 6=Maximum Debug,
7=All ] (default is: 1):
JVM arguments (default is: ):
Edit 'Connection' settings? [ 1=Yes, 2=No ] (default is: 1):
MBean server connection type [ 1=JSR-160-Compliant Server, 2=WebSphere Application Server version 6.0,
3=WebSphere Application Server version 6.1, 4=WebSphere Application Server version 7.0 ] (default is: 1): 4
Edit 'WebSphere Application Server version 7.0' settings? [ 1=Yes, 2=No ] (default is: 1):WAS user ID (default is: ):
Enter WAS password (default is: ):
Re-type : WAS password (default is: ):
WAS host name (default is: localhost): rhea00b02
WAS port (default is: 2809):
WAS connector protocol [ 1=rmi, 2=soap ] (default is: 1):
WAS profile name (default is: ): default
-----
WAS Class Path Information
WAS base paths (default is: C:\Program Files\IBM\WebSphere\AppServer;opt/IBM/WebSphere/AppServer): /opt/WAS61
WAS class path (default is: runtimes/com.ibm.ws.admin.client_6.1.0.jar;runtimes/com.ibm.ws.ejb.thinclient_7.0.0.jar):
WAS JAR directories (default is: lib;plugins):
Edit 'WebSphere eXtreme Scale Grid Servers' settings? [ 1=Yes, 2=No ] (default is: 1):
No 'WebSphere eXtreme Scale Grid Servers' settings available?
Edit 'WebSphere eXtreme Scale Grid Servers' settings, [1=Add, 2=Edit, 3=Del, 4=Next, 5=Exit] (default is: 4): 1
WebSphere eXtreme Scale Grid Servers (default is: ): rhea00b02
JMX service URL (default is: service:jmx:rmi:///jndi/rmi://localhost:<port>/objectgrid/MBeanServer):
```

```

'WebSphere eXtreme Scale Grid Servers' settings: WebSphere eXtreme Scale Grid Servers=rhea00b02
Edit 'WebSphere eXtreme Scale Grid Servers' settings, [1=Add, 2=Edit, 3=Del, 4=Next, 5=Exit] (default is: 4): 5
Edit 'WebSphere eXtreme Scale Catalog Service' settings? [ 1=Yes, 2=No ] (default is: 1): 2
Will this agent connect to a TEMS? [1=YES, 2=NO] (Default is: 1):
TEMS Host Name (Default is: rhea00b02):

Network Protocol [ip, sna, ip.pipe or ip.spipe] (Default is: ip.pipe):

    Now choose the next protocol number from one of these:
    - ip
    - sna
    - ip.spipe
    - 0 for none
Network Protocol 2 (Default is: 0):
IP.PIPE Port Number (Default is: 1918):
Enter name of KDC_PARTITION (Default is: null):

Configure connection for a secondary TEMS? [1=YES, 2=NO] (Default is: 2):
Enter Optional Primary Network Name or 0 for "none" (Default is: 0):
Agent configuration completed...
rhea00b02 #

```

For WebSphere Application Server deployments, you do not need to create multiple sub nodes. The 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 eXtreme Scale 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 `startOgServer.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 WebSphere 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 WebSphere 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`

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: **WebSphere 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 workspaces. For more information, see the information about customizing workspaces in the IBM Tivoli Monitoring Information Center.

Related concepts

“Vendor tools” on page 322

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.

“Statistics overview” on page 291

Statistics in WebSphere eXtreme Scale are built on an internal statistics tree. The StatsAccessor API, Performance Monitoring Infrastructure (PMI) modules, and MBean API are built from the internal tree.

Monitoring 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 `objectgridRoot/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. You also must enable statistics in the server properties file. Catalog servers are not detected by this filter.
 - For information about starting servers with the `-jmxServicePort` option, see “startOgServer script” on page 270.
 - For information about enabling statistics in the server properties file, see the information about the `statsSpec` property in “Server properties file” on page 230.
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. 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.
6. 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.

Related concepts

“Vendor tools” on page 322

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.

“Statistics overview” on page 291

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.

Logs and trace

You can use logs and trace to monitor and troubleshoot your environment. Logs are in different locations depending on your configuration. You might need to provide trace for a server when you work with IBM support.

Logs with WebSphere Application Server

See the WebSphere Application Server Information Center for more information.

Logs with WebSphere eXtreme Scale in a stand-alone environment

With stand-alone catalog and container servers, you set the location of logs and any trace specification. The catalog server logs are in the location where you ran the start server command.

Setting the log location for container servers

By default, the logs for a container are in the directory where the server command was run. If you start the servers in the `<eXtremeScale_home>/bin` directory, the logs and trace files are in the `logs/<server_name>` directories in the `bin` directory. To specify an alternate location of a container server logs, create a properties file, such as a `server.properties` file, 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 if trace was enabled with the `traceSpec` option. To use a properties file during container startup, use the `-serverProps` option and provide the server properties file location.

See “Starting stand-alone WebSphere eXtreme Scale servers” on page 262 and “startOgServer script” on page 270 for more information.

Common information messages to look for in the `SystemOut.log` file are start confirmation messages. For more information about a specific message, see “Messages” on page 369.

Trace with WebSphere Application Server

See the WebSphere Application Server Information Center for more information.

Trace on a stand-alone catalog service

You can set trace on a catalog service by using the **-traceSpec** and **-traceFile** parameters during catalog service startup. For example:

```
startOgServer.sh catalogServer -traceSpec  
ObjectGridPlacement=all=enabled -traceFile  
/home/user1/logs/trace.log
```

If you start the catalog service in the `<eXtremeScale_home>/bin` directory, the logs and trace files will be in a `logs/<catalog_service_name>` directory in the `bin` directory. See “Starting a stand-alone catalog service” on page 263 for more details on starting a catalog service.

Trace on a stand-alone container server

You can enable trace on a container server in two ways. You can create a server properties file as explained in the logs section, or you can enable trace by using the command line on startup. To enable container trace with a server properties file, update the **traceSpec** property with the required trace specification. To enable container trace using start parameters, use the **-traceSpec** and **-traceFile** parameters. For example:

```
startOgServer.sh c0 -objectGridFile ../xml/myObjectGrid.xml  
-deploymentPolicyFile ../xml/myDepPolicy.xml -catalogServiceEndpoints  
server1.rchland.ibm.com:2809 -traceSpec  
ObjectGridPlacement=all=enabled -traceFile /home/user1/logs/trace.log
```

If you start the server in the `<eXtremeScale_home>/bin` directory, the logs and trace files are in the `logs/<server_name>` directories in the `bin` directory. See “Starting container processes” on page 267 for more details on starting a container process.

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.

Trace with the ObjectGridManager interface

Another option is to 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");
```

Trace with the xsadmin utility

To enable trace with the xsadmin utility, use the **setTraceSpec** option. Use the xsadmin utility to enable trace on a stand-alone environment during run time instead of during startup. You can enable trace with xsadmin on container servers only:

```
xsadmin.bat -setTraceSpec "ObjectGridReplication=all=enabled"
```

You can also disable trace by setting the trace specification to `*=all=disabled`.

See “Using the xsAdmin sample utility” on page 319 for more information.

ffdc directory and 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
```

Related concepts

“Starting and stopping secure eXtreme Scale servers” on page 358

Servers often need to be secure for your deployment environment, which requires specific configuration for starting and stopping.

Related tasks

“Starting stand-alone WebSphere eXtreme Scale servers” on page 262

When you are running a stand-alone WebSphere eXtreme Scale configuration, the environment is comprised of catalog servers, container servers, and eXtreme Scale client processes. You must manually configure and start these processes.

“Starting a stand-alone catalog service” on page 263

You must start the catalog service manually when you are using a distributed WebSphere eXtreme Scale environment that is not running in WebSphere Application Server.

“Starting container processes” on page 267

You can start eXtreme Scale from the command line using a deployment topology or using a `server.properties` file.

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. Similarly to WebSphere Application Server trace, 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.

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.

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.

Chapter 9. Securing the deployment environment

To protect your WebSphere eXtreme Scale data, eXtreme Scale 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 eXtreme Scale. 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 object grids.

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.

Data grid security

WebSphere eXtreme Scale data grid security ensures that a joining server has the right credentials, so a malicious server cannot join the 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 230 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 key store 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.

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, there are provided plug-ins to obtain and validate Subject objects.

The following sections describe the two ways in which you can enable local security:

You can use the ObjectGrid XML file to define an ObjectGrid and enable the security for that ObjectGrid. 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>
```

You can also enable 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();
```

For more information, see the API documentation.

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_HOME/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 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 `credentialAuthentication` on the client and server properties files to enable the credential authentication. For more information, see “Client properties file” on page 236 and “Server properties file” on page 230.

The following table provides which authentication mechanism to use under different settings.

Table 12. 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>
```

```
</security>
</securityConfig>
```

Use the `-clusterSecurityFile` option when starting a secure server to set the security XML file. Refer to 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 listed on List of methods and their required permissions 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.

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="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.

There are three different access by creator only modes:

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:

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>
```

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.

TLS/SSL 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.

You can set the `transportType` property in the following client and server configuration files:

- To set the property in the client security configuration, see "Client properties file" on page 236.
- To set the property in the container server security configuration, see "Server properties file" on page 230.
- To set the property in the catalog server security configuration, see "Server properties file" on page 230.

Table 13. Transport protocol to use under client transport and server transport settings

Client <code>transportType</code> property	Server <code>transportType</code> property	Resulting protocol
TCP/IP	TCP/IP	TCP/IP
TCP/IP	SSL-supported	TCP/IP
TCP/IP	SSL-required	Error

Table 13. Transport protocol to use under client transport and server transport settings (continued)

Client transportType property	Server transportType property	Resulting protocol
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

When 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 a WebSphere Application Server, then you can use WebSphere Application Server's transports security support to configure SSL parameters.

Configuring the orb.properties file for transport security support

You can use TLS/SSL when the transportType property has a value of SSL-Supported.

To support secure transport in a Java Platform, Standard Edition environment, you must modify the "ORB properties file" on page 239 file to include the following properties:

```
# 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

# WS Interceptors
org.omg.PortableInterceptor.ORBInitializerClass.com.ibm.ws.objectgrid.corba.ObjectGridInitializer
org.omg.PortableInterceptor.ORBInitializerClass.com.ibm.ISecurityLocalObjectBaseL13Impl.SecurityComponentFactory

# 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.TransportMode=Pluggable
com.ibm.CORBA.ServerName=ogserver
```

Configuring SSL parameters for eXtreme Scale clients

You can configure SSL parameters for clients in the following ways:

1. Create a `com.ibm.websphere.objectgrid.security.config.SSLConfiguration` object by using the `com.ibm.websphere.objectgrid.security.config.ClientSecurityConfigurationFactory` factory class. For more details, refer to the `ClientSecurityConfigurationFactory` API Documentation.
2. Configure the parameters in the `client.properties` file, and then use the `ClientSecurityConfigurationFactory.getClientSecurityConfiguration(String)` method to populate the object instance.

See the section on security client properties in the "Client properties file" on page 236 for examples of properties that you can set on a client.

Configuring SSL parameters for eXtreme Scale servers

SSL parameters are configured for servers using a server properties file, such as the examples of `server.properties` file referred to above. This property file can be passed as a parameter when starting an eXtreme Scale server. For more information about the SSL parameters you can set for eXtreme Scale servers, see “Server properties file” on page 230.

Transport 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 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 Application Server CSIV2 transport settings. Therefore, the SSL properties still need to be configured in the server properties file for the catalog server.

Related reference

“Properties file reference” on page 229

Server properties files contain settings for running your catalog servers and container servers. You can specify a server properties file for either a stand-alone or WebSphere Application Server configuration. Client property files contain settings for your client.

“Server properties file” on page 230

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 the catalog service and container servers.

“Client properties file” on page 236

You can create a properties file based on your requirements for eXtreme Scale client processes.

“ORB properties file” on page 239

The `orb.properties` file is used to pass the properties that are used by the Object Request Broker (ORB) to modify the transport behavior of the data grid.

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.

Java Management Extensions (JMX) security

You can secure managed beans (MBean) invocations in a distributed environment.

For more information on the MBeans available, see “Using Managed Beans (MBeans) to administer your environment” on page 312.

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 JavaTM 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",
"xxxxxx"));

    // 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` is provided with the user ID set to `admin` and the password set to `xxxxxx`. 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 trust store, trust store type, and trust store 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

Security descriptor XML file

Use an ObjectGrid security descriptor XML file to configure an eXtreme Scale deployment topology with security enabled. The following sample XML files describe several configurations.

Each element and attribute of the cluster XML file is described in the following list. Use the examples to learn how to use these elements and attributes to configure the environment.

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 342. (Optional)

singleSignOnEnabled

Allows a client to connect to any server after it has authenticated with one of the servers if 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 346. (Optional)

The following security.xml file is a sample configuration to enable the eXtreme Scale 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 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 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`. In this example, the system credential generator is a `com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredentialGenerator`, 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” on page 228 for more information.

Security integration with WebSphere Application Server

WebSphere eXtreme Scale provides several security features to integrate with the WebSphere Application Server security infrastructure.

Authentication integration

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.

Distributed security integration with WebSphere Application Server

For the distributed ObjectGrid model, the security integration can be completed by using 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

See “Application client authentication” on page 344 for more information. The following example illustrates how to use the WSTokenCredentialGenerator class:

```
/**
 * connect to the ObjectGrid Server.
 */
protected ClientClusterContext connect() throws ConnectException {
    ClientSecurityConfiguration csConfig = ClientSecurityConfigurationFactory
        .getClientSecurityConfiguration(profile);

    CredentialGenerator gen = getWSCredGen();

    csConfig.setCredentialGenerator(gen);

    return objectGridManager.connect(csConfig, null);
}

/**
 * Get a WSTokenCredentialGenerator
 */
private CredentialGenerator getWSCredGen() {
    WSTokenCredentialGenerator gen = new WSTokenCredentialGenerator(
        WSTokenCredentialGenerator.RUN_AS_SUBJECT);
    return gen;
}
```

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, the security integration can be completed by using the following two 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.

Starting and stopping secure eXtreme Scale servers

Servers often need to be secure for your deployment environment, which requires specific configuration for starting and stopping.

Starting a secure server in a Java SE environment

You can start a catalog service or container servers as follows.

Starting a secure eXtreme Scale catalog service

Starting a secure eXtreme Scale catalog service process requires two more security 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 specific to the server.

When you use `startOgServer.sh` or `startOgServer.cat` command to start a secure eXtreme Scale catalog service process, you can use the `-clusterSecurityFile` or `-clusterSecurityUrl` to set the security descriptor XML file as a file type or URL type, and you can use `-serverProps` to set the server property file.

Starting a secure eXtreme Scale container server

Starting a secure eXtreme Scale container server requires one 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 230 for more details.

When you use `startOgServer.sh` or `startOgServer.cat` command to start a secure eXtreme Scale container server, you can use `-serverProps` to set the server property file. There are more ways to set the server property file, refer to the server properties file for more details.

For more details on how to use the `startOgServer.sh` or `startOgServer.bat` command and its options, refer to “startOgServer script” on page 270.

Stopping a secure eXtreme Scale server

Stopping a secure eXtreme Scale catalog service process or container server requires one 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 236 for more details.

When you use `stopOgServer.sh` or `stopOgServer.cat` command to stop a secure eXtreme Scale catalog service process or container server, you can use `-clientSecurityFile` to set the client security properties.

For more details on how to use the `stopOgServer.sh` or `stopOgServer.cat` command and its options, refer to “stopOgServer script” on page 281.

Starting a secure server in WebSphere Application Server

Starting a secure ObjectGrid server in WebSphere Application Server is similar to starting a non-secure ObjectGrids server except that you need to pass the security configuration files. Instead of using the `-[PROPERTY_FILE]` (for example `-serverProps`) in the command as in the Java SE environment, you use the `-D[PROPERTY_FILE]` in the generic Java Virtual Machine (JVM) arguments.

Starting a secure catalog service in Websphere Application Server

A catalog server contains two different levels of security information:

- `-Dobjectgrid.cluster.security.xml.url`: This specifies the `objectGridSecurity.xml` file which describes the security properties common to all servers (including catalog servers and container servers). One example 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`: This specifies the server property file which 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"`. Note that the use of `-Dobjectgrid.security.server.props` is deprecated, but you can continue using it for backward compatibility.

To start a secure catalog service in WebSphere Application Server, follow the "Embedded in WebSphere Application Server" in the "Data grid security" on page 342.

Next, set the security property in the generic JVM argument of the process.

```
-Dobjectgrid.cluster.security.xml.url=file:///tmp/og/objectGridSecurity.xml-Dobjectgrid.server.props=/tmp/og/catalog.server.props
```

Steps to add the generic JVM arguments are as follows:

- Expand "System administration" on the left-side task view.
- Click on the WebSphere Application Server process that the catalog service is deployed on, for example, "Deployment manager".
- On the right page, expand "Java and Process Management" under "Server Infrastructure".
- Click on "Process Definition".
- Click on "Java Virtual Machine" under "Additional Properties".
- Type the properties in the Generic JVM arguments textbox.

Starting a secure container server in WebSphere Application Server

A container server, when connecting to the catalog server, will get all the security configurations configured in the `objectGridSecurity.xml`, such as authenticator configuration or login session timeout setting. Also, a container server has to configure its own server-specific security properties in the `-Dobjectgrid.server.props` property.

You must use `-Dobjectgrid.server.props` property instead of `-Dobjectgrid.security.server.props` property because we also put other non-security related properties in this property file. 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.

Related tasks

“Starting stand-alone WebSphere eXtreme Scale servers” on page 262

When you are running a stand-alone WebSphere eXtreme Scale configuration, the environment is comprised of catalog servers, container servers, and eXtreme Scale client processes. You must manually configure and start these processes.

“Starting a stand-alone catalog service” on page 263

You must start the catalog service manually when you are using a distributed WebSphere eXtreme Scale environment that is not running in WebSphere Application Server.

“Starting container processes” on page 267

You can start eXtreme Scale from the command line using a deployment topology or using a `server.properties` file.

Related reference

“startOgServer script” on page 270

The `startOgServer` script starts servers. You can use a variety of parameters when you start your servers to enable trace, specify port numbers, and so on.

“Logs and trace” on page 335

You can use logs and trace to monitor and troubleshoot your environment. Logs are in different locations depending on your configuration. You might need to provide trace for a server when you work with IBM support.

Chapter 10. Troubleshooting

In addition to the logs and trace, messages, and release notes, you can use monitoring tools to troubleshoot your configuration.

Using monitoring tools for troubleshooting

In addition to the logs and trace, messages, and release notes, you can use monitoring tools to figure out issues such as the location of data in the environment, the availability of servers in the 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.

For more information about monitoring tools, see Chapter 8, “Monitoring your deployment environment,” on page 291.

Logs and trace

You can use logs and trace to monitor and troubleshoot your environment. Logs are in different locations depending on your configuration. You might need to provide trace for a server when you work with IBM support.

Logs with WebSphere Application Server

See the WebSphere Application Server Information Center for more information.

Logs with WebSphere eXtreme Scale in a stand-alone environment

With stand-alone catalog and container servers, you set the location of logs and any trace specification. The catalog server logs are in the location where you ran the start server command.

Setting the log location for container servers

By default, the logs for a container are in the directory where the server command was run. If you start the servers in the `<extremeScale_home>/bin` directory, the logs and trace files are in the `logs/<server_name>` directories in the `bin` directory. To specify an alternate location of a container server logs, create a properties file, such as a `server.properties` file, 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 if trace was enabled with the `traceSpec` option. To use a properties file during container startup, use the `-serverProps` option and provide the server properties file location.

See “Starting stand-alone WebSphere eXtreme Scale servers” on page 262 and “startOgServer script” on page 270 for more information.

Common information messages to look for in the SystemOut.log file are start confirmation messages. For more information about a specific message, see “Messages” on page 369.

Trace with WebSphere Application Server

See the WebSphere Application Server Information Center for more information.

Trace on a stand-alone catalog service

You can set trace on a catalog service by using the **-traceSpec** and **-traceFile** parameters during catalog service startup. For example:

```
startOgServer.sh catalogServer -traceSpec  
ObjectGridPlacement=all=enabled -traceFile  
/home/user1/logs/trace.log
```

If you start the catalog service in the `<eXtremeScale_home>/bin` directory, the logs and trace files will be in a `logs/<catalog_service_name>` directory in the `bin` directory. See “Starting a stand-alone catalog service” on page 263 for more details on starting a catalog service.

Trace on a stand-alone container server

You can enable trace on a container server in two ways. You can create a server properties file as explained in the logs section, or you can enable trace by using the command line on startup. To enable container trace with a server properties file, update the **traceSpec** property with the required trace specification. To enable container trace using start parameters, use the **-traceSpec** and **-traceFile** parameters. For example:

```
startOgServer.sh c0 -objectGridFile ../xml/myObjectGrid.xml  
-deploymentPolicyFile ../xml/myDepPolicy.xml -catalogServiceEndpoints  
server1.rchland.ibm.com:2809 -traceSpec  
ObjectGridPlacement=all=enabled -traceFile /home/user1/logs/trace.log
```

If you start the server in the `<eXtremeScale_home>/bin` directory, the logs and trace files are in the `logs/<server_name>` directories in the `bin` directory. See “Starting container processes” on page 267 for more details on starting a container process.

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.

Trace with the ObjectGridManager interface

Another option is to 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");
```

Trace with the xsadmin utility

To enable trace with the xsadmin utility, use the **setTraceSpec** option. Use the xsadmin utility to enable trace on a stand-alone environment during run time instead of during startup. You can enable trace with xsadmin on container servers only:

```
xsadmin.bat -setTraceSpec "ObjectGridReplication=all=enabled"
```

You can also disable trace by setting the trace specification to `*=all=disabled`.

See “Using the xsAdmin sample utility” on page 319 for more information.

ffdc directory and 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_20002000_07.03.05_10.52.18_0.txt
```

Related concepts

“Starting and stopping secure eXtreme Scale servers” on page 358

Servers often need to be secure for your deployment environment, which requires specific configuration for starting and stopping.

Related tasks

“Starting stand-alone WebSphere eXtreme Scale servers” on page 262

When you are running a stand-alone WebSphere eXtreme Scale configuration, the environment is comprised of catalog servers, container servers, and eXtreme Scale client processes. You must manually configure and start these processes.

“Starting a stand-alone catalog service” on page 263

You must start the catalog service manually when you are using a distributed WebSphere eXtreme Scale environment that is not running in WebSphere Application Server.

“Starting container processes” on page 267

You can start eXtreme Scale from the command line using a deployment topology or using a `server.properties` file.

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. Similarly to WebSphere Application Server trace, 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.

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.

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 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" />
```

Related concepts

“Loaders” on page 105

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

“Loader configuration” on page 108

Implementing a loader requires configuration for several attributes.

Loaders overview

Related reference

“Write-behind dumper class sample code” on page 121

This sample source code shows how to write a watcher (dumper) to handle failed write-behind updates.

Troubleshooting client connectivity problems

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`.

- 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.

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 370

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.

- 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.

Chapter 11. Glossary

This glossary includes terms and definitions for WebSphere eXtreme Scale.

The following cross-references are used in this glossary:

1. See refers the reader from a term to a preferred synonym, or from an acronym or abbreviation to the defined full form.
2. See also refers the reader to a related or contrasting term.

To view glossaries for other IBM products, go to www.ibm.com/software/globalization/terminology.

administrator. A person responsible for administrative tasks such as access authorization and content management. Administrators can also grant levels of authority to users.

agent. A program that performs an action on behalf of a user or other program without user intervention or on a regular schedule, and reports the results back to the user or program.

APAR. See authorized program analysis report.

API. See application programming interface.

application. One or more computer programs or software components that provide a function in direct support of a specific business process or processes.

application programming interface (API). An interface that allows an application program that is written in a high-level language to use specific data or functions of the operating system or another program.

application server. A server program in a distributed network that provides the execution environment for an application program.

asynchronous. Pertaining to events that are not synchronized in time or do not occur in regular or predictable time intervals.

asynchronous messaging. A method of communication between programs in which a program places a message on a message queue, then proceeds with its own processing without waiting for a reply to its message.

asynchronous replica. A shard that receives updates after the transaction commits. This method is faster than a synchronous replica, but introduces the possibility of data loss because the asynchronous replica can be several transactions behind the primary shard.

authenticated user. A portal user who has logged in to the portal with a valid account (user ID and password). Authenticated users have access to all public places.

authentication. A security service that provides proof that a user of a computer system is genuinely who that person claims to be. Common mechanisms for implementing this service are passwords and digital signatures. Authentication is distinct from authorization; authentication is not concerned with granting or denying access to system resources.

authentication alias. An alias that authorizes access to resource adapters and data sources. An authentication alias contains authentication data, including a user ID and password.

authorization. The process of granting a user, system, or process either complete or restricted access to an object, resource, or function.

authorization policy. A policy whose policy target is a business service and whose contract contains one or more assertions that grant permission to run a channel action.

authorization table. A table that contains the role to user or group mapping information that identifies the permitted access of a client to a particular resource.

authorized program analysis report (APAR). A request for correction of a defect in a supported release of an IBM-supplied program.

autodiscovery. The discovery of service artifacts in a file system, external registry, or another source.

autonomic manager. A set of software or hardware components, configured by policies, which manage the behavior of other software or hardware components as a human might manage them. An autonomic manager includes a control loop that consists of monitor, analyze, plan, and execute components.

availability.

1. The condition allowing users to access and use their applications and data.
2. The time periods during which a resource is accessible. For example, a contractor might have an availability of 9 AM to 5 PM every weekday, and 9 AM to 3 PM on Saturdays.

bean. A definition or instance of a JavaBeans component. See also JavaBeans, enterprise bean.

bean class. In Enterprise JavaBeans (EJB) programming, a Java class that implements a `javax.ejb.EntityBean` class or `javax.ejb.SessionBean` class.

Bean Scripting Framework. An architecture for incorporating scripting language functions to Java applications.

bean-managed messaging. A function of asynchronous messaging that gives an enterprise bean complete control over the messaging infrastructure.

bean-managed persistence (BMP). The mechanism whereby data transfer between an entity bean's variables and a resource manager is managed by the entity bean. (Sun)

bean-managed transaction (BMT). The capability of the session bean, servlet, or application client component to manage its own transactions directly, instead of through a container.

binary format. Representation of a decimal value in which each field must be 2 or 4 bytes long. The sign (+ or -) is in the far left bit of the field, and the number value is in the remaining bits of the field. Positive numbers have a 0 in the sign bit and are in true form. Negative numbers have a 1 in the sign bit and are in twos complement form.

BMP. See bean-managed persistence.

BMT. See bean-managed transaction.

bootstrap. A small program that loads larger programs during system initialization.

bootstrapping. The process by which an initial reference of the naming service is obtained. The bootstrap setting and the host name form the initial context for Java Naming and Directory Interface (JNDI) references.

bottleneck. A place in the system where contention for a resource is affecting performance.

bottom-up development. In Web services, the process of developing a service from an existing artifact such as a Java bean or enterprise bean rather than a Web Services Description Language (WSDL) file.

breakpoint. A marked point in a process or programmatic flow that causes that flow to pause when the point is reached, typically to allow debugging or monitoring.

build definition file. An XML file that identifies components and characteristics for a customized installation package (CIP).

build path. The path that is used during compilation of Java source code, in order to find referenced classes that reside in other projects.

build plan. An XML file that defines the processing necessary to build generation outputs and that specifies the machine where processing takes place.

build time data. Objects that are not used by the translator, such as EDI standards, record oriented data document types, and maps.

bytecode. Machine-independent code generated by the Java compiler and executed by the Java interpreter. (Sun)

cache instance resource. A location where any Java Platform, Enterprise Edition (Java EE) application can store, distribute, and share data.

cache replication. The sharing of cache IDs, cache entries, and cache invalidations with other servers in the same replication domain.

catalog. A container that, depending on the container type, holds processes, data, resources, organizations, or reports in the project tree.

catalog service. A service that controls placement of shards and discovers and monitors the health of containers.

category. A container used in a structure diagram to group elements based on a shared attribute or quality.

cell.

1. A group of managed processes that are federated to the same deployment manager and can include high-availability core groups.

2. One or more processes that each host runtime components. Each has one or more named core groups.

cell-scoped binding. A binding scope where the binding is not specific to, and not associated with any node or server. This type of name binding is created under the persistent root context of a cell.

chassis. The metal frame in which various electronic components are mounted.

child node. A node within the scope of another node.

CIP. See customized installation package.

class. In object-oriented design or programming, a model or template that can be used to create objects with a common definition and common properties, operations, and behavior. An object is an instance of a class.

class file. A compiled Java source file.

class hierarchy. The relationships between classes that share a single inheritance.

class loader. Part of the Java virtual machine (JVM) that is responsible for finding and loading class files. A class loader affects the packaging of applications and the runtime behavior of packaged applications deployed on application servers.

class path. A list of directories and JAR files that contain resource files or Java classes that a program can load dynamically at run time.

classifier. A specialized attribute used for grouping and color-coding process elements.

client. A software program or computer that requests services from a server. See also host.

client application. An application, running on a workstation and linked to a client, that gives the application access to queuing services on a server.

client/server. Pertaining to the model of interaction in distributed data processing in which a program on one computer sends a request to a program on another computer and awaits a response. The requesting program is called a client; the answering program is called a server.

Cloudscape. An embeddable, all Java, object-relational database management system (ORDBMS).

cluster. A group of application servers that collaborate for the purposes of workload balancing and failover.

coarse-grained. Pertaining to viewing a group of objects from an abstract or high level.

coherent cache. Cache that maintains integrity so that all clients see the same data.

collection certificate store. A collection of intermediate certificates or certificate revocation lists (CRL) that are used by a certificate path to build up a certificate chain for validation.

comma delimited file. A file whose records contain fields that are separated by a comma.

command bean. A proxy that can invoke a single operation using an execute() method.

command line. The blank line on a display where commands, option numbers, or selections can be entered.

compilation unit. A portion of a computer program sufficiently complete to be compiled correctly.

compile time. The time period during which a computer program is being compiled into an executable program.

component.

1. A reusable object or program that performs a specific function and works with other components and applications.
2. In Eclipse, one or more plug-ins that work together to deliver a discrete set of functions.

component element. An entity in a component where a breakpoint can be set, such as an activity or Java snippet in a business process, or a mediation primitive or node in a mediation flow.

component instance. A running component that can be running in parallel with other instances of the same component.

component test. An automated test of one or more components of an enterprise application, which may include Java classes, EJB beans, or Web services.

container server. A server instance that can host multiple shards. One Java virtual machine (JVM) can host multiple container servers.

converter. In Enterprise JavaBeans (EJB) programming, a class that translates a database representation to an object type and back.

create method. In enterprise beans, a method defined in the home interface and invoked by a client to create an enterprise bean. (Sun)

credential. In the Java Authentication and Authorization Service (JAAS) framework, a subject class that owns security-related attributes. These attributes can contain information used to authenticate the subject to new services.

customized installation package (CIP). A customized installation image that can include one or more maintenance packages, a configuration archive file from a stand-alone server profile, one or more enterprise archive files, scripts, and other files that help customize the resulting installation.

daemon. A program that runs unattended to perform continuous or periodic functions, such as network control.

dashboard. A Web page that can contain one or more viewers that graphically represent business data.

data grid. A system for accessing terabytes or petabytes of data.

DB2. A family of IBM licensed programs for relational database management.

deadlock. A condition in which two independent threads of control are blocked, each waiting for the other to take some action. Deadlock often arises from adding synchronization mechanisms to avoid race conditions.

demilitarized zone (DMZ). A configuration that includes multiple firewalls to add layers of protection between a corporate intranet and a public network, such as the Internet.

deploy. To place files or install software into an operational environment. In Java Platform, Enterprise Edition (Java EE), this involves creating a deployment descriptor suitable to the type of application that is being deployed.

deploy phase. See deployment phase.

deployment code. Additional code that enables bean implementation code written by an application developer to work in a particular EJB runtime environment. Deployment code can be generated by tools that the application server vendor supplies.

deployment descriptor. An XML file that describes how to deploy a module or application by specifying configuration and container options. For example, an EJB deployment descriptor passes information to an EJB container about how to manage and control an enterprise bean.

deployment directory. The directory where the published server configuration and Web application are located on the machine where the application server is installed.

deployment environment. A collection of configured clusters, servers, and middleware that collaborate to provide an environment to host software modules. For example, a deployment environment might include a host for message destinations, a processor or sorter of business events, and administrative programs.

deployment manager. A server that manages operations for a logical group or cell of other servers.

deployment phase. A phase that includes a combination of creating the hosting environment for your applications and the deployment of those applications. This includes resolving the application's resource dependencies, operational conditions, capacity requirements, and integrity and access constraints.

deployment policy. An optional way to configure an eXtreme Scale environment based on various items, including: number of systems, servers, partitions, replicas (including type of replica), and heap sizes for each server.

deployment topology. The configuration of servers and clusters in a deployment environment and the physical and logical relationships among them.

deprecated. Pertaining to an entity, such as a programming element or feature, that is supported but no longer recommended and that might become obsolete.

derivation. In object-oriented programming, the refinement or extension of one class from another.

deserialization. A method for converting a serialized variable into object data.

destination. An exit point that is used to deliver documents to a back-end system or a trading partner.

digital certificate. An electronic document used to identify an individual, a system, a server, a company, or some other entity, and to associate a public key with the entity. A digital certificate is issued by a certification authority and is digitally signed by that authority.

dirty read. A read request that does not involve any locking mechanism. This means that data can be read that might later be rolled back resulting in an inconsistency between what was read and what is in the database.

distributed eXtreme Scale. A usage pattern for interacting with eXtreme Scale when servers and clients exist on multiple processes.

DMZ. See demilitarized zone.

DNS. See Domain Name System.

do-while loop. A loop that repeats the same sequence of activities as long as some condition is satisfied. Unlike a while loop, a do-while loop tests its condition at the end of the loop. This means that its sequence of activities always runs at least once.

document type definition (DTD). The rules that specify the structure for a particular class of SGML or XML documents. The DTD defines the structure with elements, attributes, and notations, and it establishes constraints for how each element, attribute, and notation can be used within the particular class of documents.

domain. An object, icon, or container that contains other objects representing the resources of a domain. The domain object can be used to manage those resources.

Domain Name System (DNS). The distributed database system that maps domain names to IP addresses.

downstream. Pertaining to the direction of the flow, which is from the first node in the process (upstream) toward the last node in the process (downstream).

drop-down. See pull-down.

DTD. See document type definition.

DTD document definition. A description or layout of an XML document based on an XML DTD.

dynamic cache. A consolidation of several caching activities, including servlets, Web services, and WebSphere commands into one service where these activities share configuration parameters and work together to improve performance.

dynamic cluster. A server cluster that uses weights to balance the workloads of its cluster members dynamically, based on performance information collected from cluster members.

EAR. See enterprise archive.

EAR project. See enterprise application project.

Eclipse. An open-source initiative that provides ISVs and other tool developers with a standard platform for developing plug-compatible application development tools.

edition. A successive deployment generation of a particular set of versioned artifacts.

editor area. In Eclipse and Eclipse-based products, the area in the workbench window where files are opened for editing.

EJB. See Enterprise JavaBeans.

EJB container. A container that implements the EJB component contract of the Java EE architecture. This contract specifies a runtime environment for enterprise beans that includes security, concurrency, life cycle management, transaction, deployment, and other services. (Sun)

EJB context. In enterprise beans, an object that allows an enterprise bean to invoke services provided by the container and to obtain information about the caller of a client-invoked method. (Sun)

EJB factory. An access bean that simplifies the creating or finding of an enterprise bean instance.

EJB home object. In Enterprise JavaBeans (EJB) programming, an object that provides the life cycle operations (create, remove, find) for an enterprise bean. (Sun)

EJB inheritance. A form of inheritance in which an enterprise bean inherits properties, methods, and method-level control descriptor attributes from another enterprise bean that resides in the same group.

EJB JAR file. A Java archive that contains an EJB module. (Sun)

EJB module. A software unit that consists of one or more enterprise beans and an EJB deployment descriptor. (Sun)

EJB object. In enterprise beans, an object whose class implements the enterprise bean remote interface (Sun).

EJB project. A project that contains the resources needed for EJB applications, including enterprise beans; home, local, and remote interfaces; JSP files; servlets; and deployment descriptors.

EJB query. In EJB query language, a string that contains an optional SELECT clause specifying the EJB objects to return, a FROM clause that names the bean collections, an optional WHERE clause that contains search predicates over the collections, an optional ORDER BY clause that specifies the ordering of the result collection, and input parameters that correspond to the arguments of the finder method.

EJB reference. A logical name used by an application to locate the home interface of an enterprise bean in the target operational environment.

EJB server. Software that provides services to an EJB container. An EJB server may host one or more EJB containers. (Sun)

embedded server. A catalog service or container server that resides in an existing process and is started and stopped within the process.

endpoint.

1. A JCA application or other client consumer of an event from the enterprise information system.
2. The system that is the origin or destination of a session.

endpoint listener. The point or address at which incoming messages for a Web service are received by a service integration bus.

enterprise application project (EAR project). A structure and hierarchy of folders and files that contain a deployment descriptor and IBM extension document as well as files that are common to all Java EE modules that are defined in the deployment descriptor.

enterprise archive (EAR). A specialized type of JAR file, defined by the Java EE standard, used to deploy Java EE applications to Java EE application servers. An EAR file contains EJB components, a deployment descriptor, and Web archive (WAR) files for individual Web applications. See also Web archive.

enterprise bean. A component that implements a business task or business entity and resides in an EJB container. Entity beans, session beans, and message-driven beans are all enterprise beans. (Sun) See also bean.

Enterprise JavaBeans (EJB). A component architecture defined by Sun Microsystems for the development and deployment of object-oriented, distributed, enterprise-level applications (Java EE).

enterprise service bus (ESB). A flexible connectivity infrastructure for integrating applications and services; it offers a flexible and manageable approach to service-oriented architecture implementation.

entity.

1. A simple Java class that represents a row in a database table or entry in a map.
2. In markup languages such as XML, a collection of characters that can be referenced as a unit, for example to incorporate often-repeated text or special characters within a document.

entity bean. In EJB programming, an enterprise bean that represents persistent data maintained in a database. Each entity bean carries its own identity. (Sun)

entry breakpoint. A breakpoint set on a component element that is hit before the component element is invoked.

environment. A named collection of logical and physical resources used to support the performance of a function.

environment variable. A variable that specifies how an operating system or another program runs, or the devices that the operating system recognizes.

error. A discrepancy between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition.

error log stream. A continuous flow of error information that is transmitted using a predefined format.

ESB. See enterprise service bus.

event.

1. A change to a state, such as the completion or failure of an operation, business process, or human task, that can trigger a subsequent action, such as persisting the event data to a data repository or invoking another business process.
2. A change to data in an enterprise information system (EIS) that is processed by the adapter and used to deliver business objects from the EIS to the endpoints (applications) that need to be notified of the change.

evictor. A component that controls the membership of entries in each BackingMap instance. Sparse caches can use evictors to automatically remove data from the cache without affecting the database.

exception. A condition or event that cannot be handled by a normal process.

exception handler. A set of routines that responds to an abnormal condition. An exception handler is able to interrupt and to resume the normal running of processes.

exclusive lock. A lock that prevents concurrently executing application processes from accessing database data. See also shared lock.

execution trace. A chain of events that is recorded and displayed in a hierarchal format on the Events page of the integration test client.

export. An exposed interface from a Service Component Architecture (SCA) module that offers a business service to the outside world. An export has a binding that defines how the service can be accessed by service requesters, for example, as a Web service.

export file.

1. A file created during the development process for inbound operations that contains the configuration settings for inbound processing.
2. The file containing data that has been exported.

expression. An SQL or XQuery operand or a collection of SQL or XQuery operators and operands that yields a single value.

Extensible Markup Language (XML). A standard metalanguage for defining markup languages that is based on Standard Generalized Markup Language (SGML).

eXtreme Scale grid. A pattern that is used to interact with eXtreme Scale when all of the data and clients are in one process.

factory. In object-oriented programming, a class that is used to create instances of another class. A factory is used to isolate the creation of objects of a particular class into one place so that new functions can be provided without widespread code changes.

failover. An automatic operation that switches to a redundant or standby system in the event of a software, hardware, or network interruption.

fine-grained. Pertaining to viewing an individual object in detail.

fire. In object-oriented programming, to cause a state transition.

firewall. A network configuration, typically both hardware and software, that prevents unauthorized traffic into and out of a secure network.

fix pack. A cumulative collection of fixes that is made available between scheduled refresh packs, manufacturing refreshes, or releases. It is intended to allow customers to move to a specific maintenance level. See also interim fix.

folder. A container used to organize objects.

for loop. A loop that repeats the same sequence of activities a specified number of times.

fork. A process element that makes copies of its input and forwards them by several processing paths in parallel.

garbage collection. A routine that searches memory to reclaim space from program segments or inactive data.

General Inter-ORB Protocol (GIOP). A protocol that Common Object Request Broker Architecture (CORBA) uses to define the format of messages.

generic object. An object that is used in API calls and XPATH expressions to refer to concepts, custom entities, or collections. For example, the XPATH expression `/WSRR/GenericObject` will retrieve all concepts from WebSphere Service Registry and Repository.

getter method. A method whose purpose is to get the value of an instance or class variable. This allows another object to find out the value of one of its variables.

GIOP. See General Inter-ORB Protocol.

global.

1. Pertaining to an element that is available to any process in the workspace. A global element appears in the project tree and can be used in multiple processes. Tasks, processes, repositories, and services can be either global (referenced by any process in the project) or local (specific to a single process).
2. Pertaining to information available to more than one program or subroutine.

global attribute. In XML, an attribute that is declared as a child of the schema element rather than as part of a complex type definition. Global attributes can be referenced in one or more content models using the `ref` attribute.

global element. In XML, an element that is declared as a child of the schema element rather than as part of a complex type definition. Global elements can be referenced in one or more content models using the ref attribute.

global instance identifier. A globally unique identifier that is generated either by the application or by the emitter and is used as a primary key for event identification.

global security. Pertains to all applications running in the environment and determines whether security is used, the type of registry used for authentication, and other values, many of which act as defaults.

global transaction. A recoverable unit of work performed by one or more resource managers in a distributed transaction environment and coordinated by an external transaction manager.

global variable. A variable that is used to hold and manipulate values assigned to it during translation and that is shared across maps and across document translations. One of the three types of variables supported by the Data Interchange Services mapping command language.

group.

1. A collection of users who can share access authorities for protected resources.
2. A set of related documents within an interchange. An interchange can contain zero to many groups.
3. In places, two or more people who are grouped for membership in a place.

HA. See high availability.

HA group. A collection of one or more members used to provide high availability for a process.

HA policy. A set of rules that is defined for an HA group that dictate whether zero (0), or more members are activated. The policy is associated with a specific HA group by matching the policy match criteria with the group name.

health. The general condition or state of the database environment.

heartbeat. A signal that one entity sends to another to convey that it is still active.

high availability (HA). Pertaining to a clustered system that is reconfigured when node or daemon failures occur, so that workloads can be redistributed to the remaining nodes in the cluster.

high availability manager. A framework within which core group membership is determined and status is communicated between core group members.

host.

1. A computer that is connected to a network and that provides an access point to that network. The host can be a client, a server, or both a client and server simultaneously.
2. In performance profiling, a machine that owns processes that are being profiled. See also server.

host name.

1. In Internet communication, the name given to a computer. The host name might be a fully qualified domain name such as mycomputer.city.company.com, or it might be a specific subname such as mycomputer.
2. The network name for a network adapter on a physical machine in which the node is installed.

host system. An enterprise mainframe computer system that hosts 3270 applications. In the 3270 terminal service development tools, the developer uses the 3270 terminal service recorder to connect to the host system.

HTTP over SSL (HTTPS). A Web protocol for secure transactions that encrypts and decrypts user page requests and pages returned by the Web server.

HTTPS.

1. See HTTP over SSL.
2. See Hypertext Transfer Protocol Secure.

Hypertext Transfer Protocol Secure (HTTPS). An Internet protocol that is used by Web servers and Web browsers to transfer and display hypermedia documents securely across the Internet.

IDE. See integrated development environment.

if-then rule. A rule in which the action (then part) is performed only when the condition (if part) is true.

IIOP. See Internet Inter-ORB Protocol.

import.

1. A development artifact that imports a service that is external to a module.
2. The point at which an SCA module accesses an external service, (a service outside the SCA module) as if it was local. An import defines interactions between the SCA module and the service provider. An import has a binding and one or more interfaces.

index. A set of pointers that are logically ordered by the values of a key. Indexes provide quick access to data and can enforce uniqueness of the key values for the rows in the table.

information center. A collection of information that provides support for users of one or more products, can be launched separately from the product, and includes a list of topics for navigation and a search engine.

inheritance. An object-oriented programming technique in which existing classes are used as a basis for creating other classes. Through inheritance, more specific elements incorporate the structure and behavior of more general elements.

installation package. An installable unit of a software product. Software product packages are separately installable units that can operate independently from other packages of that software product.

installation target. The system on which selected installation packages are installed.

instance. A specific occurrence of an object that belongs to a class.

instantiate. To represent an abstraction with a concrete instance.

integrated development environment (IDE). A set of software development tools, such as source editors, compilers, and debuggers, that are accessible from a single user interface.

interface. A collection of operations that are used to specify a service of a class or a component.

interim fix. A certified fix that is generally available to all customers between regularly scheduled fix packs, refresh packs, or releases. See also fix pack.

Internet Inter-ORB Protocol (IIOP). A protocol used for communication between Common Object Request Broker Architecture (CORBA) object request brokers.

Internet Protocol (IP). A protocol that routes data through a network or interconnected networks. This protocol acts as an intermediary between the higher protocol layers and the physical network.

invocation. The activation of a program or procedure.

IP. See Internet Protocol.

IP sprayer. A device that is located between inbound requests from the users and the application server nodes that reroutes requests across nodes.

iteration. See loop.

iterator. A class or construct that is used to step through a collection of objects one at a time.

JAAS. See Java Authentication and Authorization Service.

JAF. See JavaBeans Activation Framework.

JAR file. A Java archive file. See also Web archive, enterprise archive.

Java. An object-oriented programming language for portable interpretive code that supports interaction among remote objects. Java was developed and specified by Sun Microsystems, Incorporated.

Java API for XML (JAX). A set of Java-based APIs for handling various operations involving data defined through Extensible Markup Language (XML).

Java archive. A compressed file format for storing all of the resources that are required to install and run a Java program in a single file. See also Web archive, enterprise archive.

Java Authentication and Authorization Service (JAAS). In Java EE technology, a standard API for performing security-based operations. Through JAAS, services can authenticate and authorize users while enabling the applications to remain independent from underlying technologies.

Java class. A class that is written in the Java language.

Java Command Language. A scripting language for the Java environment that is used to create Web content and to control Java applications.

Java Connector security. An architecture designed to extend the end-to-end security model for Java EE-based applications to include enterprise information systems (EIS).

Java Database Connectivity (JDBC). An industry standard for database-independent connectivity between the Java platform and a wide range of databases. The JDBC interface provides a call level interface for SQL-based and XQuery-based database access.

Java EE. See Java Platform, Enterprise Edition.

Java EE application. Any deployable unit of Java EE functionality. This unit can be a single module or a group of modules packaged into an enterprise archive (EAR) file with a Java EE application deployment descriptor. (Sun)

Java EE Connector Architecture (JCA). A standard architecture for connecting the Java EE platform to heterogeneous enterprise information systems (EIS).

Java EE server. A runtime environment that provides EJB or Web containers.

Java file. An editable source file (with .java extension) that can be compiled into bytecode (a .class file).

Java Management Extensions (JMX). A means of doing management of and through Java technology. JMX is a universal, open extension of the Java programming language for management that can be deployed across all industries, wherever management is needed.

Java Message Service (JMS). An application programming interface that provides Java language functions for handling messages.

Java Naming and Directory Interface (JNDI). An extension to the Java platform that provides a standard interface for heterogeneous naming and directory services.

Java platform. A collective term for the Java language for writing programs; a set of APIs, class libraries, and other programs used in developing, compiling, and error-checking programs; and a Java virtual machine which loads and runs the class files. (Sun)

Java Platform, Enterprise Edition (Java EE). An environment for developing and deploying enterprise applications, defined by Sun Microsystems Inc. The Java EE platform consists of a set of services, application programming interfaces (APIs), and protocols that provide the functionality for developing multitiered, Web-based applications. (Sun)

Java Platform, Standard Edition (Java SE). The core Java technology platform. (Sun)

Java project. In Eclipse, a project that contains compilable Java source code and is a container for source folders or packages.

Java runtime environment. A subset of a Java developer kit that contains the core executable programs and files that constitute the standard Java platform. The JRE includes the Java virtual machine (JVM), core classes, and supporting files.

Java SE. See Java Platform, Standard Edition.

Java SE Development Kit (JDK). The name of the software development kit that Sun Microsystems provides for the Java platform.

Java Secure Socket Extension (JSSE). A Java package that enables secure Internet communications. It implements a Java version of the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols and supports data encryption, server authentication, message integrity, and optionally client authentication.

Java Specification Request (JSR). A formally proposed specification for the Java platform.

Java virtual machine (JVM). A software implementation of a processor that runs compiled Java code (applets and applications).

Java virtual machine Profiler Interface (JVMPi). A profiling tool that supports the collection of information, such as data about garbage collection and the Java virtual machine (JVM) API that runs the application server.

JavaBeans. As defined for Java by Sun Microsystems, a portable, platform-independent, reusable component model. See also bean.

JavaBeans Activation Framework (JAF). A standard extension to the Java platform that determines arbitrary data types and available operations and can instantiate a bean to run pertinent services.

Javadoc.

1. A tool that parses the declarations and documentation comments in a set of source files and produces a set of HTML pages describing the classes, inner classes, interfaces, constructors, methods, and fields. (Sun)

2. Pertaining to the tool that parses the declarations and documentation comments in a set of source files and produces a set of HTML pages describing the classes, inner classes, interfaces, constructors, methods, and fields.

JavaMail API. A platform and protocol-independent framework for building Java-based mail client applications.

JavaScript. A Web scripting language that is used in both browsers and Web servers. (Sun)

JavaScript Object Notation. A lightweight data-interchange format that is based on the object-literal notation of JavaScript. JSON is programming-language neutral but uses conventions from languages that include C, C++, C#, Java, JavaScript, Perl, Python.

JavaServer Pages (JSP). A server-side scripting technology that enables Java code to be dynamically embedded within Web pages (HTML files) and run when the page is served, in order to return dynamic content to a client.

JAX. See Java API for XML.

JCA. See Java EE Connector Architecture.

JDBC. See Java Database Connectivity.

JDK. See Java SE Development Kit.

JMS. See Java Message Service.

JMS data binding. A data binding that provides a mapping between the format used by an external JMS message and the Service Data Object (SDO) representation used by a Service Component Architecture (SCA) module.

JMX. See Java Management Extensions.

JNDI. See Java Naming and Directory Interface.

join.

1. A process element that recombines and synchronizes parallel processing paths after a decision or fork. A join waits for input to arrive at each of its incoming branches before permitting the process to continue.

2. An SQL relational operation in which data can be retrieved from two tables, typically based on a join condition specifying join columns.

3. The configuration on an incoming link that determines the behavior of the link.

JSP. See JavaServer Pages.

JSP file. A scripted HTML file that has a .jsp extension and allows for the inclusion of dynamic content in Web pages. A JSP file can be directly requested as a URL, called by a servlet, or called from within an HTML page.

JSP page. A text-based document using fixed template data and JSP elements that describes how to process a request to create a response. (Sun)

JSR. See Java Specification Request.

JSSE. See Java Secure Socket Extension.

JVM. See Java virtual machine.

JVMPI. See Java virtual machine Profiler Interface.

Jython. An implementation of the Python programming language that is integrated with the Java platform.

key.

1. A cryptographic mathematical value that is used to digitally sign, verify, encrypt, or decrypt a message.
2. Information that characterizes and uniquely identifies the real-world entity that is being tracked by a monitoring context.

keyword. One of the predefined words of a programming language, artificial language, application, or command.

LDAP. See Lightweight Directory Access Protocol.

LDAP directory. A type of repository that stores information on people, organizations, and other resources and that is accessed using the LDAP protocol. The entries in the repository are organized into a hierarchical structure, and in some cases the hierarchical structure reflects the structure or geography of an organization.

library.

1. A collection of model elements, including business items, processes, tasks, resources, and organizations.
2. A project that is used for the development, version management, and organization of shared resources. Only a subset of the artifact types can be created and stored in a library, such as business objects and interfaces.

life cycle. One complete pass through the four phases of software development: inception, elaboration, construction and transition.

Lightweight Directory Access Protocol (LDAP). An open protocol that uses TCP/IP to provide access to directories that support an X.500 model and that does not incur the resource requirements of the more complex X.500 Directory Access Protocol (DAP). For example, LDAP can be used to locate people, organizations, and other resources in an Internet or intranet directory.

Lightweight Third Party Authentication (LTPA). A protocol that uses cryptography to support security in a distributed environment.

listener. A program that detects incoming requests and starts the associated channel.

listener port. An object that defines the association between a connection factory, a destination, and a deployed message-driven bean. Listener ports simplify the administration of the associations between these resources.

load balancing. The monitoring of application servers and management of the workload on servers. If one server exceeds its workload, requests are forwarded to another server with more capacity.

loader. A component that reads data from and writes data to a persistent store.

local.

1. Pertaining to a device, file, or system that is accessed directly from a user system, without the use of a communication line.
2. Pertaining to an element that is available only in its own process.

local database. A database that is located on the workstation in use.

lock. A means of preventing uncommitted changes made by one application process from being perceived by another application process and for preventing one application process from updating data that is being accessed by another process. A lock ensures the integrity of data by preventing concurrent users from accessing inconsistent data.

logging. The recording of data about specific events on the system, such as errors.

long name. The property that specifies the logical name for the server on the z/OS platform.

loop. A sequence of instructions performed repeatedly.

LTPA. See Lightweight Third Party Authentication.

maintenance mode. A state of a node or server that an administrator can use to diagnose, maintain, or tune the node or server without disrupting incoming traffic in a production environment.

Managed Bean (MBean). In the Java Management Extensions (JMX) specification, the Java objects that implement resources and their instrumentation.

map.

1. A data structure that maps keys to values.
2. A file that defines the transformation between sources and targets.
3. In the EJB development environment, the specification of how the container-managed persistent fields of an enterprise bean correspond to columns in a relational database table or other persistent storage.

MBean. See Managed Bean.

MBean provider. A library containing an implementation of a Java Management Extensions (JMX) MBean and its MBean Extensible Markup Language (XML) descriptor file.

memory leak. The effect of a program that maintains references to objects that are no longer required and therefore need to be reclaimed.

method. In object-oriented programming, an operation that an object can perform. An object can have many methods.

metric. A holder for information, typically a business performance measurement, in a monitoring context.

namespace. A logical container in which all the names are unique. The unique identifier for an artifact is composed of the namespace and the local name of the artifact.

node.

1. A logical grouping of managed servers.
2. Any item on a tree control, including a simple element, compound element, mapping command, comment, or group node.
3. In XML, the smallest unit of valid, complete structure in a document.
4. The fundamental shapes that make up a diagram.

node agent. An administrative agent that manages all application servers on a node and represents the node in the management cell.

object. In object-oriented design or programming, a concrete realization (instance) of a class that consists of data and the operations associated with that data. An object contains the instance data that is defined by the class, but the class owns the operations that are associated with the data.

Object Request Broker (ORB). In object-oriented programming, software that serves as an intermediary by transparently enabling objects to exchange requests and responses.

object-oriented programming. A programming approach based on the concepts of data abstraction and inheritance. Unlike procedural programming techniques, object-oriented programming concentrates not on how something is accomplished but instead on what data objects comprise the problem and how they are manipulated.

ObjectGrid. A grid-enabled, memory database for applications that are written in Java. ObjectGrid can be used as an in-memory database or to distribute data across a network.

ODBC. See Open Database Connectivity.

Open Database Connectivity (ODBC). A standard application programming interface (API) for accessing data in both relational and nonrelational database management systems. Using this API, database applications can access data stored in database management systems on a variety of computers even if each database management system uses a different data storage format and programming interface.

open source. Pertaining to software whose source code is publicly available for use or modification. Open source software is typically developed as a public collaboration and made freely available, although its use and redistribution might be subject to licensing restrictions. Linux is a well known example of open source software.

operation. An implementation of functions or queries that an object might be called to perform.

ORB. See Object Request Broker.

organization. An entity where people cooperate to accomplish specified objectives, such as an enterprise, a company, or a factory.

package.

1. In Java programming, a group of types. Packages are declared with the package keyword. (Sun)
2. The wrapper around the document content that defines the format used to transmit a document over the Internet, for example, RNIF, AS1, and AS2.
3. To assemble components into modules and modules into enterprise applications.

partitioning facility. A programming framework and a system management infrastructure that supports the concept of partitioning for enterprise beans, HTTP traffic, and database access.

Performance Monitoring Infrastructure (PMI). A set of packages and libraries assigned to gather, deliver, process, and display performance data.

permission. Authorization to perform activities, such as reading and writing local files, creating network connections, and loading native code.

persist. To be maintained across session boundaries, typically in nonvolatile storage such as a database system or a directory.

persistence.

1. A characteristic of data that is maintained across session boundaries, or of an object that continues to exist after the execution of the program or process that created it, typically in nonvolatile storage such as a database system.
2. In Java EE, the protocol for transferring the state of an entity bean between its instance variables and an underlying database. (Sun)

persistent data store. A nonvolatile storage for event data, such as a database system, that is maintained across session boundaries and that continues to exist after the execution of the program or process that created it.

pessimistic locking. A locking strategy whereby a lock is held between the time that a row is selected and the time that a searched update or delete operation is attempted on that row.

plug-in. A separately installable software module that adds function to an existing program, application, or interface.

PMI. See Performance Monitoring Infrastructure.

point-to-point. Pertaining to a style of messaging application in which the sending application knows the destination of the message.

policy. A set of considerations that influence the behavior of a managed resource or a user.

port. As defined in a Web Services Description Language (WSDL) document, a single endpoint that is defined as a combination of a binding and a network address.

port number. In Internet communications, the identifier for a logical connector between an application entity and the transport service.

primary key.

1. An object that uniquely identifies an entity bean of a particular type.
2. In a relational database, a key that uniquely identifies one row of a database table.

primitive type. In Java, a category of data type that describes a variable that contains a single value of the appropriate size and format for its type: a number, a character, or a Boolean value. Examples of primitive types include byte, short, int, long, float, double, char, boolean.

process.

1. A progressively continuing procedure consisting of a series of controlled activities that are systematically directed toward a particular result or end.
2. The sequence of documents or messages to be exchanged between the Community Managers and participants to run a business transaction.

profile. Data that describes the characteristics of a user, group, resource, program, device, or remote location.

program temporary fix (PTF). For System i[®], System p[®], and System z[®] products, a fix that is tested by IBM and is made available to all customers. See also fix pack.

prompt. A component of an action that indicates that user input is required for a field before making a transition to an output screen.

property. A characteristic of an object that describes the object. A property can be changed or modified. Properties can describe an object name, type, value, or behavior, among other things.

protocol binding. A binding that enables the enterprise service bus to process messages independently of the communication protocol.

proxy. An application gateway from one network to another for a specific network application such as Telnet or FTP, for example, where a firewall proxy Telnet server performs authentication of the user and then lets the traffic flow through the proxy as if it were not there. Function is performed in the firewall and not in the client workstation, causing more load in the firewall.

proxy cluster. A group of proxy servers that distributes HTTP requests across the cluster.

proxy peer access point. A means of identifying the communication settings for a peer access point that cannot be accessed directly.

proxy server.

1. A server that acts as an intermediary for HTTP Web requests that are hosted by an application or a Web server. A proxy server acts as a surrogate for the content servers in the enterprise.
2. A server that receives requests intended for another server and that acts on behalf of the client (as the client's proxy) to obtain the requested service. A proxy server is often used when the client and the server are incompatible for direct connection. For example, the client is unable to meet the security authentication requirements of the server but should be permitted some services.

PTF. See program temporary fix.

public.

1. In object-oriented programming, pertaining to a class member that is accessible to all classes.
2. In the Java programming language, pertains to a method or variable that can be accessed by elements residing in other classes. (Sun)

QoS. See quality of service.

qualifier. A simple element that gives another generic compound or simple element a specific meaning. Qualifiers are used in mapping single or multiple occurrences. A qualifier can also be used to denote the namespace used to interpret the second part of the name, typically referred to as the ID.

quality of service (QoS). A set of communication characteristics that an application requires. Quality of Service (QoS) defines a specific transmission priority, level of route reliability, and security level.

query.

1. A request for information from a database based on specific conditions: for example, a request for a list of all customers in a customer table whose balances are greater than USD1000.
2. A reusable request for information about one or more model elements

read-through cache. A sparse cache that loads data entries by key as they are requested. When data cannot be found in the cache, the missing data is retrieved with the loader, which loads the data from the back-end data repository and inserts the data into the cache.

recursion. A programming technique in which a program or routine calls itself to perform successive steps in an operation, with each step using the output of the preceding step.

refresh pack. A cumulative collection of fixes that contains new functions. See also fix pack, interim fix.

region. A contiguous area of virtual storage that has common characteristics and that can be shared between processes.

replica. A server that contains a copy of the directory or directories of another server. Replicas back up servers in order to enhance performance or response times and to ensure data integrity.

replication. The process of maintaining a defined set of data in more than one location. Replication involves copying designated changes for one location (a source) to another (a target) and synchronizing the data in both locations.

resource.

1. A discrete asset, for example application suites, applications, business services, interfaces, endpoints, and business events.
2. A facility of a computing system or operating system required by a job, task, or running program. Resources include main storage, input/output devices, the processing unit, data sets, files, libraries, folders, application servers, and control or processing programs.
3. A person, piece of equipment, or material that is used to perform a task or a project. Each resource is a particular occurrence or example of a resource definition.

role.

1. A description of a function to be carried out by an individual or bulk resource, and the qualifications required to fulfill the function. In simulation and analysis, the term role is also used to refer to the qualified resources.
2. A job function that identifies the tasks that a user can perform and the resources to which a user has access. A user can be assigned one or more roles.
3. A logical group of principals that provides a set of permissions. Access to operations is controlled by granting access to a role.
4. In a relationship, a role determines the function and participation of entities. Roles capture structure and constraint requirements on participating entities and their manner of participation. For example, in an employment relationship, the roles are employer and employee.

root. The user name for the system user with the most authority.

run time. The time period during which a computer program is running.

runtime topology. A depiction of the momentary state of the environment.

scalability. The ability of a system to expand as resources, such as processors, memory, or storage, are added.

scope.

1. A specification of the boundary within which system resources can be used.
2. In Web services, a property that identifies the lifetime of the object serving the invocation request.

script. A series of commands, combined in a file, that carry out a particular function when the file is run. Scripts are interpreted as they are run.

scripting. A style of programming that reuses existing components as a base for building applications.

SDK. See software development kit.

Secure Sockets Layer (SSL). A security protocol that provides communication privacy. With SSL, client/server applications can communicate in a way that is designed to prevent eavesdropping, tampering, and message forgery.

security administrator. The person who controls access to business data and program functions.

security token. A representation of a set of claims that are made by a client that can include a name, password, identity, key, certificate, group, privilege, and so on.

serialization. In object-oriented programming, the writing of data in sequential fashion to a communications medium from program memory.

serializer. A method for converting object data to another form such as binary or XML.

servant region. A contiguous area of virtual storage that is dynamically started as load increases and automatically stopped as load eases.

server. A software program or a computer that provides services to other software programs or other computers. See also host.

server cluster. A group of servers that are typically on different physical machines and have the same applications configured within them, but operate as a single logical server.

service level agreement (SLA). A contract between a customer and a service provider that specifies the expectations for the level of service with respect to availability, performance, and other measurable objectives.

servlet. A Java program that runs on a Web server and extends the server functions by generating dynamic content in response to Web client requests. Servlets are commonly used to connect databases to the Web.

session.

1. A logical or virtual connection between two stations, software programs, or devices on a network that allows the two elements to communicate and exchange data.
2. A series of requests to a servlet originating from the same user at the same browser.
3. In Java EE, an object used by a servlet to track user interaction with a Web application across multiple HTTP requests.

session affinity. A method of configuring applications in which a client is always connected to the same server. These configurations disable workload management after an initial connection by forcing a client request to always go to the same server.

setter method. A method whose purpose is to set the value of an instance or class variable. This capability allows another object to set the value of one of its variables.

shard. An instance of a partition. A shard can be a primary or replica.

shared lock. A lock that limits concurrently running application processes to read-only operations on database data.

shell script. A program, or script, that is interpreted by the shell of an operating system.

signer certificate. The trusted certificate entry that is typically in a truststore file.

silent installation. An installation that does not send messages to the console but instead stores messages and errors in log files. A silent installation can use response files for data input.

silent mode. A method for installing or uninstalling a product component from the command line with no GUI display. When using silent mode, you specify the data required by the installation or uninstallation program directly on the command line or in a file (called an option file or response file).

skeleton. Scaffolding for an implementation class.

SLA. See service level agreement.

software development kit (SDK). A set of tools, APIs, and documentation to assist with the development of software in a specific computer language or for a particular operating environment.

SQL. See Structured Query Language.

SQL query. A component of certain SQL statements that specifies a result table.

SSL. See Secure Sockets Layer.

SSL channel. A type of channel within a transport chain that associates a Secure Sockets Layer (SSL) configuration repertoire with the transport chain.

stack. An area in memory that typically stores information such as temporary register information, values of parameters, and return addresses of subroutines and is based on the principle of last in, first out (LIFO).

stand-alone. Independent of any other device, program, or system. In a network environment, a stand-alone machine accesses all required resources locally.

stand-alone server. A catalog service or container server that is managed from the operating system that starts and stops the server process.

static. A Java programming language keyword that is used to define a variable as a class variable.

string. In programming languages, the form of data used for storing and manipulating text.

Structured Query Language (SQL). A standardized language for defining and manipulating data in a relational database.

subclass. In Java, a class that is derived from a particular class, through inheritance.

subquery. In SQL, a subselect used within a predicate, for example, a select-statement within the WHERE or HAVING clause of another SQL statement.

synchronize. To add, subtract, or change one feature or artifact to match another.

synchronous process. A process that starts by invoking a request-response operation. The result of the process is returned by the same operation.

synchronous replica. A shard that receives updates as part of the transaction on the primary shard to guarantee data consistency, which can increase the response time compared with an asynchronous replica.

syntax. The rules for the construction of a command or statement.

systems analyst. A specialist who is responsible for translating business requirements into system definitions and solutions.

TCP. See Transmission Control Protocol.

TCP channel. A type of channel within a transport chain that provides client applications with persistent connections within a local area network (LAN).

TCP/IP. See Transmission Control Protocol/Internet Protocol.

TCP/IP monitoring server. A runtime environment that monitors all requests and responses between a Web browser and an application server, as well as TCP/IP activity.

thin application client. A lightweight, downloadable Java application run time capable of interacting with enterprise beans.

thin client. A client that has little or no installed software but has access to software that is managed and delivered by network servers that are attached to it. A thin client is an alternative to a full-function client such as a workstation.

thread. A stream of computer instructions that is in control of a process. In some operating systems, a thread is the smallest unit of operation in a process. Several threads can run concurrently, performing different jobs.

thread contention. A condition in which a thread is waiting for a lock or object that another thread holds.

threshold. A setting that applies to an interrupt in a simulation that defines when a process simulation should be halted based on a condition existing for a specified proportion of occurrences of some event.

throughput. The measure of the amount of work performed by a device, such as a computer or printer, over a period of time, for example, number of jobs per day.

time to live. The time interval in seconds that an entry can exist in the cache before that entry is discarded.

timeout. A time interval that is allotted for an event to occur or complete before operation is interrupted.

timer. A task that produces output at certain points in time.

timing constraint. A specialized validation action used to measure the duration of a method call or a sequence of method calls.

Tivoli Performance Viewer. A Java client that retrieves the Performance Monitoring Infrastructure (PMI) data from an application server and displays it in various formats.

token.

1. A marker used to track the current state of a process instance during a simulation run.
2. A particular message or bit pattern that signifies permission or temporary control to transmit over a network.

topology. The physical or logical mapping of the location of networking components or nodes within a network. Common network topologies include bus, ring, star, and tree.

transaction. A process in which all of the data modifications that are made during a transaction are either committed together as a unit or rolled back as a unit.

Transmission Control Protocol (TCP). A communication protocol used in the Internet and in any network that follows the Internet Engineering Task Force (IETF) standards for internetwork protocol. TCP provides a reliable host-to-host protocol in packet-switched communication networks and in interconnected systems of such networks.

Transmission Control Protocol/Internet Protocol (TCP/IP). An industry-standard, nonproprietary set of communication protocols that provides reliable end-to-end connections between applications over interconnected networks of different types.

truststore file. A key database file that contains the public keys for a trusted entity.

type.

1. In Java programming, a class or interface.
2. In a WSDL document, an element that contains data type definitions using some type system (such as XSD).

UDDI. See Universal Description, Discovery, and Integration.

Uniform Resource Identifier (URI).

1. A compact string of characters for identifying an abstract or physical resource.
2. A unique address that is used to identify content on the Web, such as a page of text, a video or sound clip, a still or animated image, or a program. The most common form of URI is the Web page address, which is a particular form or subset of URI called a Uniform Resource Locator (URL). A URI typically describes how to access the resource, the computer that contains the resource, and the name of the resource (a file name) on the computer.

Uniform Resource Locator (URL). The unique address of an information resource that is accessible in a network such as the Internet. The URL includes the abbreviated name of the protocol used to access the information resource and the information used by the protocol to locate the information resource.

Uniform Resource Name (URN). A name that uniquely identifies a Web service to a client.

Universal Description, Discovery, and Integration (UDDI). A set of standards-based specifications that enables companies and applications to quickly and easily find and use Web services over the Internet.

Universally Unique Identifier (UUID). The 128-bit numerical identifier that is used to ensure that two components do not have the same identifier.

UNIX System Services. An element of z/OS that creates a UNIX environment that conforms to XPG4 UNIX 1995 specifications and that provides two open-system interfaces on the z/OS operating system: an application programming interface (API) and an interactive shell interface.

upgradeable lock. A lock that identifies the intent to update a cache entry when using a pessimistic lock.

upstream. Pertaining to the direction of the flow, which is from the start of the process (upstream) toward the end of the process (downstream).

URI. See Uniform Resource Identifier.

URL. See Uniform Resource Locator.

URL scheme. A format that contains another object reference.

URN. See Uniform Resource Name.

UUID. See Universally Unique Identifier.

variable. A representation of a changeable value.

version. A separately licensed program that typically has significant new code or new function.

virtual host. A configuration enabling a single host machine to resemble multiple host machines. Resources associated with one virtual host cannot share data with resources associated with another virtual host, even if the virtual hosts share the same physical machine.

virtual machine. An abstract specification for a computing device that can be implemented in different ways in software and hardware.

virtualization. A technique that encapsulates the characteristics of resources from the way in which other systems interact with those resources.

waiter. A thread waiting for a connection.

WAR. See Web archive.

WCCM. See WebSphere Common Configuration Model.

Web archive (WAR). A compressed file format, defined by the Java EE standard, for storing all the resources required to install and run a Web application in a single file. See also enterprise archive.

Web browser. A client program that initiates requests to a Web server and displays the information that the server returns.

Web component. A servlet, JavaServer Pages (JSP) file, or a HyperText Markup Language (HTML) file. One or more Web components make up a Web module.

Web container. A container that implements the Web component contract of the Java EE architecture. (Sun)

Web container channel. A type of channel within a transport chain that creates a bridge in the transport chain between an HTTP inbound channel and a servlet or JavaServer Pages (JSP) engine.

Web crawler. A type of crawler that explores the Web by retrieving a Web document and following the links within that document.

Web server. A software program that is capable of servicing Hypertext Transfer Protocol (HTTP) requests.

Web server plug-in. A software module that supports the Web server in communicating requests for dynamic content, such as servlets, to the application server.

Web server separation. A topology where the Web server is physically separated from the application server.

Web site. A related collection of files available on the Web that is managed by a single entity (an organization or an individual) and contains information in hypertext for its users. A Web site often includes hypertext links to other Web sites.

WebSphere. An IBM brand name that encompasses tools for developing e-business applications and middleware for running Web applications.

WebSphere Common Configuration Model (WCCM). A model that provides for programmatic access to configuration data.

what you see is what you get (WYSIWYG). A capability of an editor to continually display pages exactly as they will be printed or otherwise rendered.

while loop. A loop that repeats the same sequence of activities as long as some condition is satisfied. The while loop tests its condition at the beginning of every loop. If the condition is false from the start, the sequence of activities contained in the loop never runs.

WLM. See Workload Manager.

workload management. The optimization of the distribution of incoming work requests to the application servers, enterprise beans, servlets and other objects that can effectively process the request.

Workload Manager (WLM). A component of z/OS that provides the ability to run multiple workloads at the same time within one z/OS image or across multiple images.

workspace.

1. A directory on disk that contains all project files, as well as information such as preferences.
2. A temporary repository of configuration information that administrative clients use.
3. In Eclipse, the collection of projects and other resources that the user is currently developing in the workbench. Metadata about these resources resides in a directory on the file system; the resources might reside in the same directory.

write-behind cache. A cache that asynchronously writes each write operation to the database using a loader.

write-through cache. A cache that synchronously writes each write operation to the database using a loader.

WYSIWYG. See what you see is what you get.

X/Open XA. The X/Open Distributed Transaction Processing XA interface. A proposed standard for distributed transaction communication. The standard specifies a bidirectional interface between resource managers that provide access to shared resources within transactions, and between a transaction service that monitors and resolves transactions.

XA. A bidirectional interface between one or more resource managers that provide access to shared resources and a transaction manager that monitors and resolves transactions.

XML. See Extensible Markup Language.

z/OS. An IBM mainframe operating system that uses 64-bit real storage.

zone-based support. A function that enables rules-based shard placement to improve grid availability by placing shards across different data centers, whether on different floors or even in different buildings or geographies.

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 259
 - WebSphere Application Server 283
- Administration API 276
- API 274
- availability
 - setting 259

B

- back-end 119
- backing map
 - lock strategy 85
 - plug-ins 81
 - session 81
- benefits 114
- best practices 254
- build definition file
 - creating
 - CIP 42
 - IIP 45

C

- CA Wily Introscope 329
- caching 118
- caching support 114
- caching supportloaderloader
 - transaction 114
- capacity planning 29
- catalog server
 - clustering 16
 - enabling logs 335, 363
 - enabling trace 335, 363
 - starting 259
 - stopping 259
- catalog service
 - best practices for 266
 - starting
 - in a WebSphere Application Server Environment 263
 - in an environment that is not running WebSphere Application Server 263
- catalog service process
 - starting in WebSphere Application Server 283
- client 92
- client authorization
 - creator only access 346
 - custom 346
 - JAAS 346
 - permissions
 - checking period 346
- client invalidation 97
- client properties 236
- client-server security
 - secure sockets layer (SSL) 349
 - TCP/IP 349
 - transport layer security (TLS) 349

- command line 65
- configuration 228
 - deployment
 - distributed 7
 - local 7
- configuring 68, 77, 190
- Configuring 92
- configuring after install 54
- container processes
 - starting 267
- container server
 - enabling logs 335, 363
 - enabling trace 335, 363
 - starting 259
 - stopping 259
- containers 16
- CPU sizing 32, 33
- customization definitions
 - generating 75
- customized jobs
 - running 76
 - uploading 76
- customizing 73

D

- data grid 29
- deployment policy
 - deploymentPolicy.xsd file 194
 - descriptor XML 190
 - XML configuration 194
- distributing changes
 - peer JVMs 184
- dynamic cache
 - configuring 170

E

- entity metadata
 - emd.xsd file 222
 - XML configuration 217, 222
- event listener 187
- evictors
 - configuring 177
 - plug-in 179
 - TTL evictor 177
- extension files 73

F

- failed updates 119
- failover
 - configuring 13
- First steps console 54

G

- grid authorization 351

- grid security
 - JSSE 342
 - token manager 342

H

- HTTP session manager 147
 - parameters for configuring 158
 - with WebSphere Application Server Community Edition 154
 - with WebSphere Virtual Enterprise 150
- Hyperic HQ 333

I

- IBM Installation Factory
 - build definition file 41
- IBM Tivoli Monitoring 323
- IBM Update Installer for WebSphere
 - uninstalling
 - CIP 44
- IBM Update Installer for WebSphere Software 67
- index
 - configuration 182
 - HashIndex 182
- Installation Factory
 - CIP
 - maintenance 44
- Installation Factory plug-in
 - build definition file
 - modify 47
 - installing
 - CIP 42
 - IIP 46
- installing 68
 - customized installation package 48
- IBM Installation Factory
 - CIP 41
 - IIP 41
 - maintenance 67
 - Network Deployment 39
 - server 37
 - silently 48, 64, 65
 - stand-alone 37
 - WebSphere Application Server 39
- Introscope 329
- invalidation 187

J

- Java Authentication and Authorization Service
 - JAAS 343
- Java message service 183
- Java Persistence API 127
- Java Persistence API (JPA) 124
 - cache plug-in
 - configuration 132

- Java Persistence API (JPA) *(continued)*
 - cache plug-in *(continued)*
 - introduction 128
 - cache topology
 - embedded 128, 132, 134, 142
 - embedded partitioned 128, 132, 134, 142
 - Hibernate 134
 - OpenJPA 142
 - remote 128, 132, 134, 142
 - Hibernate plug-in
 - configuration 134
 - OpenJPA plug-in
 - configuration 142
- Java virtual machine 12
- JMS 187
- JMX securityaccess control
 - authentication 352
 - JAAS support 352
 - secure transport 352
- jobs 73
- JVM 12

L

- loader 119
 - preloading 108
- loader transaction 119
- loaders
 - JPA 124
- locking
 - configuring programmatically 87
 - configuring with XML 87
 - no 87
 - optimistic 87
 - pessimistic 87
- log element 184
- log sequence 184
- logs
 - overview 335, 363

M

- managed bean 313
- Managed Beans 312
- manageprofiles command 53, 56
- map 78
- MBean 312, 313
- MBeans
 - accessing programmatically 314
 - accessing with wsadmin 314
- messages 369
- metrics 323, 333
- migrate 36
- monitor 333
- monitoring 299
 - agent 323
 - with the statistics API 295
 - with vendor tools 323
- Monitoring applications
 - with Introscope 329

N

- network 10
- Network Deployment 56

- network ports 9

O

- object request broker 239
- Object Request Broker 11, 38, 68
- ObjectGrid
 - XML configuration 212
- ObjectGrid descriptor XML 196
- ObjectGrid interface 78
- objectGrid.xsd file 212
- ObjectGridManager interface
 - enabling trace with 335, 363
- objectGridSecurity.xsd file 228
- operating systems 10
- operational checklist 26
- ORB 11
- orb.properties 239
- orb.properties file 38
- overviewprogrammatically
 - using a loader 106

P

- parallel transactions 33
- parameters 64, 65
- partition 29
- peer 183
- per partition 32
- performance monitoring
 - infrastructure 299, 304
- Performance Monitoring
 - Infrastructure 300
- Performance Monitoring Infrastructure (PMI) 291
- placement strategy 29
- planning 7, 9, 10, 26
 - configuration
 - options 9
 - requirements 9
- plug-in 78
- PMI i, 304
 - See also* performance monitoring
 - infrastructure
 - MBean 291
- profile
 - augmenting 53, 55
 - creating 53, 54
- Profile Management Tool 73, 75
- Profile Management Tool plug-in 53, 54, 55
- profiles
 - augment 56
 - create 56
 - non-root user 63
- properties 229
 - client 236
 - server 230

Q

- quorum
 - container behavior 18
 - xsadmin 18

R

- real time 254
- release notes 369
- replication 183, 187
- response file 64
- response time 254

S

- security 228, 358
 - authentication
 - creating an authenticator 344
 - LDAP 344
 - Tivoli access manager 344
 - WebSphere Application
 - Server 344
 - credential 344
 - integration 341, 357
 - introduction 341
 - local 343
 - plug-ins 343
 - single sign-on (SSO) 344
 - WebSphere Application Server 357
 - XML configuration 225, 354
 - server properties 230
 - session 78
 - session manager 150
 - session state
 - J2EE 156
 - WebSphere Application Server
 - Community Edition 156
 - sessions 147
 - shard 29
 - silently 71
 - SIP
 - session 153
 - session management 153
 - Spring
 - descriptor XML 247
 - extension beans 242
 - framework 242
 - namespace support 242
 - native transactions 242
 - objectgrid.xsd file 252
 - packaging 242
 - shard scope 242
 - webflow 242
 - XML configuration 252
 - Sprint extension beans 243
 - stand-alone 68, 262
 - start server
 - programmatically 276
 - starting
 - catalog server 270
 - container server 270
 - starting servers 262
 - startOgServer
 - options 270
 - statistics 295
 - statistics API 291
 - statistics modules 297
 - stop server
 - programmatically 276
 - stopOgserver 281
 - stopping servers 280
 - support 114, 369

T

- time-based data updater 127
- Tivoli 323
- trace
 - options for configuring 337, 365
 - overview 335, 363
- transaction 78
 - callback 108
 - ID 108
- troubleshooting 363
 - messages 369
 - release notes 369
- tuning 9, 10, 11, 12

U

- uninstalling 71

W

- wasprofile command 53, 55
- WebSphere Application Server 55, 56, 67
- WebSphere Configuration Tools 73
- WebSphere Customization Tools 73, 75
- Wily 329
- Wily Introscope 329
- write-behind 114, 118, 119
 - failed updates
 - handling 121

X

- XML 190
- xsadmin sample utility 319



Printed in USA