

# Server Guide

JEUS 9.1

**TMAXSOFT**

# Copyright

Copyright 2025. TmaxSoft Co., Ltd. All Rights Reserved.

## Company Information

TmaxSoft Co., Ltd.

TmaxSoft Tower, 45, Jeongjail-ro, Bundang-gu, Seongnam-si, Gyeonggi-do, South Korea

Website: <https://www.tmaxsoft.com/en/>

## Restricted Rights Legend

All TmaxSoft Software (JEUS®) and documents are protected by copyright laws and international convention. TmaxSoft software and documents are made available under the terms of the TmaxSoft License Agreement and this document may only be distributed or copied in accordance with the terms of this agreement. No part of this document may be transmitted, copied, deployed, or reproduced in any form or by any means, electronic, mechanical, or optical, without the prior written consent of TmaxSoft Co., Ltd. Nothing in this software document and agreement constitutes a transfer of intellectual property rights regardless of whether or not such rights are registered) or any rights to TmaxSoft trademarks, logos, or any other brand features.

This document is for information purposes only. The company assumes no direct or indirect responsibilities for the contents of this document, and does not guarantee that the information contained in this document satisfies certain legal or commercial conditions. The information contained in this document is subject to change without prior notice due to product upgrades or updates. The company assumes no liability for any errors in this document.

## Trademarks

JEUS® is registered trademark of TmaxSoft Co., Ltd.

Java, Solaris are registered trademarks of Oracle Corporation and its subsidiaries and affiliates.

Microsoft, Windows, Windows NT are registered trademarks or trademarks of Microsoft Corporation.

HP-UX is a registered trademark of Hewlett Packard Enterprise Company.

AIX is a registered trademark of International Business Machines Corporation.

UNIX is a registered trademark of X/Open Company, Ltd.

Linux is a registered trademark of Linus Torvalds.

Noto is a trademark of Google Inc. Noto fonts are open source. All Noto fonts are published under the SIL Open Font License, Version 1.1. (<https://www.google.com/get/noto/>)

Other products and company names are trademarks or registered trademarks of their respective

owners.

The names of companies, systems, and products mentioned in this manual may not necessarily be indicated with a trademark symbol (<sup>TM</sup>, ®).

## Open Source Software Notice

Some modules or files of this product are subject to the terms of the following licenses: APACHE2.0, CDDL1.0, EDL1.0, OPEN SYMPHONY SOFTWARE1.1, TRILEAD-SSH2, Bouncy Castle, BSD, MIT, SIL OPEN FONT1.1

Detailed Information related to the license can be found in the following directory:  
\${INSTALL\_PATH}/license/oss\_licenses

## Document History

Product Version	Guide Version	Date	Remarks
JEUS 9.1	3.3.1	2025-12-10	-
JEUS 9	3.1.2	2025-03-24	-
JEUS 9	3.1.1	2024-12-24	-

# Table of Contents

1. Introduction .....	1
1.1. Components .....	1
1.1.1. Master Server (MASTER).....	2
1.1.2. Managed Server (MS).....	4
1.2. Class Loader Structure.....	9
1.3. Server Directory Structure .....	10
1.4. Launcher .....	11
2. JEUS Configuration .....	13
2.1. Overview .....	13
2.2. Adding Servers .....	13
2.2.1. Using the Console Tool .....	13
2.3. Server Configuration .....	15
2.3.1. Environment Variable Configuration.....	15
2.3.2. Basic Configuration .....	16
2.3.2.1. Dynamically Changing Basic Configuration .....	16
2.3.2.2. Action On Resource Leak .....	17
2.3.2.3. JVM Config .....	19
2.3.2.4. Classpath .....	21
2.3.3. Listener Configuration .....	22
2.3.4. Thread Pool Configuration .....	23
2.3.5. Lifecycle Invocation Configuration.....	25
2.3.6. Resource Reference Configuration .....	26
2.4. Server Template Configuration .....	27
3. Controlling and Monitoring JEUS Servers .....	28
3.1. Controlling and Monitoring Servers .....	28
3.1.1. Life Cycle of Managed Servers .....	28
3.1.2. Starting a Managed Server .....	29
3.1.3. Shutting Down a Managed Server .....	31
3.1.4. Suspending a Managed Server .....	32
3.1.5. Resuming a Managed Server .....	33
3.2. Configuring Server Engine .....	34
3.2.1. Engine Option .....	34
3.2.2. Engine Initialization Time .....	35
3.3. Controlling and Monitoring Threads.....	36
3.3.1. Thread Monitoring .....	36
3.3.2. Thread Control .....	40
3.4. Controlling and Monitoring Memory .....	43
3.4.1. Memory Monitoring .....	43

3.4.2. Memory Usage Control .....	43
4. JNDI Naming Server .....	45
4.1. Overview .....	45
4.2. Basic Concepts and Structure .....	45
4.2.1. Basic Concepts .....	45
4.2.2. Checking Bound Objects .....	46
4.2.3. JNDI Naming Server Architecture .....	46
4.2.4. JNDI Clustering .....	47
4.3. Configuring JNDI Naming Server .....	49
4.3.1. Configuring JNSServer .....	49
4.3.2. Configuring JNSClient .....	51
4.4. JNDI in Clustered Environment .....	51
4.5. JNDI Programming .....	52
4.5.1. Configuring the JEUS environment .....	52
4.5.2. Configuring properties for InitialContext .....	53
4.5.3. Lookup for Named Objects Using Context .....	54
4.5.4. Using a Named Object .....	55
4.5.5. Closing the Context .....	56
4.5.6. Creating a Clustered Context .....	56
4.5.7. Remote Lookup .....	57
5. External Resource .....	58
5.1. Resource Types .....	58
5.2. Resource Configuration .....	59
5.2.1. Configuring a Data Source .....	59
5.2.2. Configuring a Mail Source .....	59
5.2.3. Configuring a URL Source .....	60
5.2.4. Configuring a Custom Resource .....	60
5.2.5. Configuring an External Resource .....	63
5.2.6. Configuring an External Resource .....	65
6. DB Connection Pool and JDBC .....	68
6.1. Overview .....	68
6.2. Data Sources and JDBC Connection Pooling .....	68
6.2.1. JDBC Driver .....	69
6.2.2. JDBC Connection Pool .....	69
6.2.3. Data Source .....	70
6.2.4. Cluster Data Source .....	71
6.2.4.1. Failover .....	71
6.2.4.2. Data Source Affinity .....	72
6.2.4.3. Cluster Data Sources Associated with ONS .....	73
6.3. Management of Data Sources and Connection Pools .....	74
6.4. Data Source Configuration .....	76

6.4.1. Connection Pool Configuration .....	79
6.5. Cluster Data Source Configuration .....	86
6.5.1. Configuring a Cluster Data Source .....	86
6.5.2. Configuring a Component Data Source in a Cluster Data Source .....	88
6.6. Dynamic Data Source Configuration .....	89
6.6.1. Adding a Data Source .....	90
6.6.2. Registering a Data Source on a Server .....	90
6.6.3. Removing a Data Source from a Server .....	92
6.6.4. Registering a Data Source in a Cluster .....	93
6.6.5. Removing Data Sources from a Cluster .....	94
6.6.6. Adding a Server to a Cluster .....	95
6.6.7. Removing a Server from a Cluster .....	96
6.6.8. Removing a Cluster .....	98
6.6.9. Removing a Data Source .....	99
6.6.10. Modifying Data Source Configuration .....	100
6.6.11. Checking Data Source Configuration .....	103
6.7. Dynamically Changing Cluster Data Source Configuration .....	104
6.7.1. Adding a Cluster Data Source .....	105
6.7.2. Registering a Cluster Data Source to a Server .....	106
6.7.3. Removing a Cluster Data Source from the Server .....	107
6.7.4. Registering a Data Source in a Cluster .....	108
6.7.5. Removing Data Sources from a Cluster .....	110
6.7.6. Adding a Server to a Cluster .....	111
6.7.7. Removing a Server from a Cluster .....	111
6.7.8. Removing a Cluster .....	111
6.7.9. Removing a Cluster Data Source .....	111
6.7.10. Changing Cluster Data Source Configuration .....	113
6.7.11. Checking the Cluster Data Source Configuration .....	114
6.8. Monitoring JDBC Connection Pool .....	115
6.8.1. Checking a JDBC Connection Pool List .....	116
6.8.2. Checking JDBC Connection Pool Information .....	118
6.9. Controlling JDBC Connection Pool .....	119
6.9.1. Creating a Connection Pool .....	119
6.9.2. Disabling a Connection Pool .....	120
6.9.3. Enabling a Connection Pool .....	121
6.9.4. Replacing a Connection in a Connection Pool .....	122
6.9.5. Minimizing the Number of Connections in a Connection Pool .....	122
6.9.6. Returning Connections to Connection Pool .....	123
6.9.7. Forcibly Destroying Connections in Connection Pool .....	124
6.10. JEUS JDBC Programming .....	125
6.10.1. Getting a Connection from a Data Source .....	125

6.10.2. Transaction Programming Rules .....	126
6.10.3. Getting a Connection Implementation Instance of the JDBC Driver .....	126
6.10.4. Connection Pool in a Standalone Client .....	126
6.11. Global Transaction (XA) and Connection Sharing .....	127
6.12. Connection Pooling Service Support for Various Users .....	127
7. Transaction Manager .....	129
7.1. Overview .....	129
7.1.1. Application .....	130
7.1.2. JEUS Transaction Manager .....	131
7.1.3. Resource Manager .....	132
7.2. Server Transaction Manager Configuration .....	132
7.2.1. Worker Thread Pool .....	133
7.2.2. Timeout Settings .....	134
7.2.3. Root Coordinator and Sub Coordinator .....	136
7.2.4. Transaction Join .....	137
7.3. Client Transaction Manager Configuration .....	137
7.3.1. Using a Transaction Manager .....	137
7.3.2. Transaction Manager Type .....	137
7.3.3. TCP/IP Port of the Transaction Manager .....	138
7.3.4. Worker Thread Pool .....	138
7.3.5. Timeout Settings .....	138
7.4. Transaction Application Programming .....	140
7.4.1. Local Transaction .....	140
7.4.2. Client-Managed Transaction .....	142
7.4.3. Bean-Managed Transaction .....	144
7.4.4. Container-Managed Transaction .....	146
7.4.5. Using a Transaction Manager .....	148
7.5. Transaction Recovery .....	148
7.5.1. Transaction Recovery Process .....	148
7.5.2. Recovery Log File .....	150
7.5.3. Recovery Related Configuration .....	151
7.5.4. Resource Manager Failure .....	152
7.6. Transaction Profile Function .....	152
7.7. Transaction Communication Problem between Servers with Different IP Bands .....	154
8. Logging .....	155
8.1. Overview .....	155
8.2. Basic JEUS Logger Structure .....	157
8.2.1. Overview .....	157
8.2.2. Launcher Logger .....	159
8.2.3. Server Logger .....	160
8.2.4. Access Logger .....	162

8.2.5. User Logger .....	163
8.2.6. Logger List .....	163
8.2.7. Log Messages with Module Names .....	167
8.3. Logging Configuration .....	169
8.3.1. Checking Logger Information .....	169
8.3.2. Dynamically Configuring a Logger .....	170
8.3.3. Standard Output and Standard Error Log Format Configuration .....	172
8.3.4. Logger Settings .....	174
8.3.5. Log File Rotation Configuration .....	179
8.3.6. Property Configuration .....	180
Appendix A: JEUS Server Ports .....	182
A.1. Server Ports .....	182
Appendix B: JDBC Data Source Configuration Examples .....	183
B.1. Overview .....	183
B.2. Oracle Thin (Type4) Configuration Example .....	183
B.2.1. Oracle Thin Connection Pool Data Source .....	183
B.2.2. Oracle Thin XA Data Source .....	184
B.2.3. Configuring java.util.Properties File with Oracle ASO .....	185
B.3. Oracle OCI (Type2) Configuration Example .....	186
B.3.1. Oracle OCI Connection Pool Data Source .....	186
B.4. DB2 Configuration Example .....	187
B.4.1. DB2 Type4 (JCC) Connection Pool Data Source .....	187
B.4.2. DB2 Type4 (JCC) XA Data Source .....	187
B.4.3. DB2 Type2 (JCC) XA Data Source .....	188
B.5. Sybase Configuration Example .....	189
B.5.1. Sybase jConnect 5.x Connection Pool Data Source .....	189
B.5.2. Sybase jConnect 6.x XA Data Source .....	189
B.6. Microsoft SQL Server Configuration Example .....	190
B.6.1. Microsoft SQL Server 2005 Connection Pool Data Source .....	190
B.7. Informix Configuration Example .....	191
B.7.1. Informix Connection Pool Data Source .....	191
B.8. Tibero Configuration Example .....	192
B.8.1. Tibero Connection Pool Data Source .....	192
B.9. MySQL 5.x Configuration Example .....	193
B.9.1. MySQL Connector/J Connection Pool Data Source .....	193



# 1. Introduction

This chapter describes components and services of JEUS.

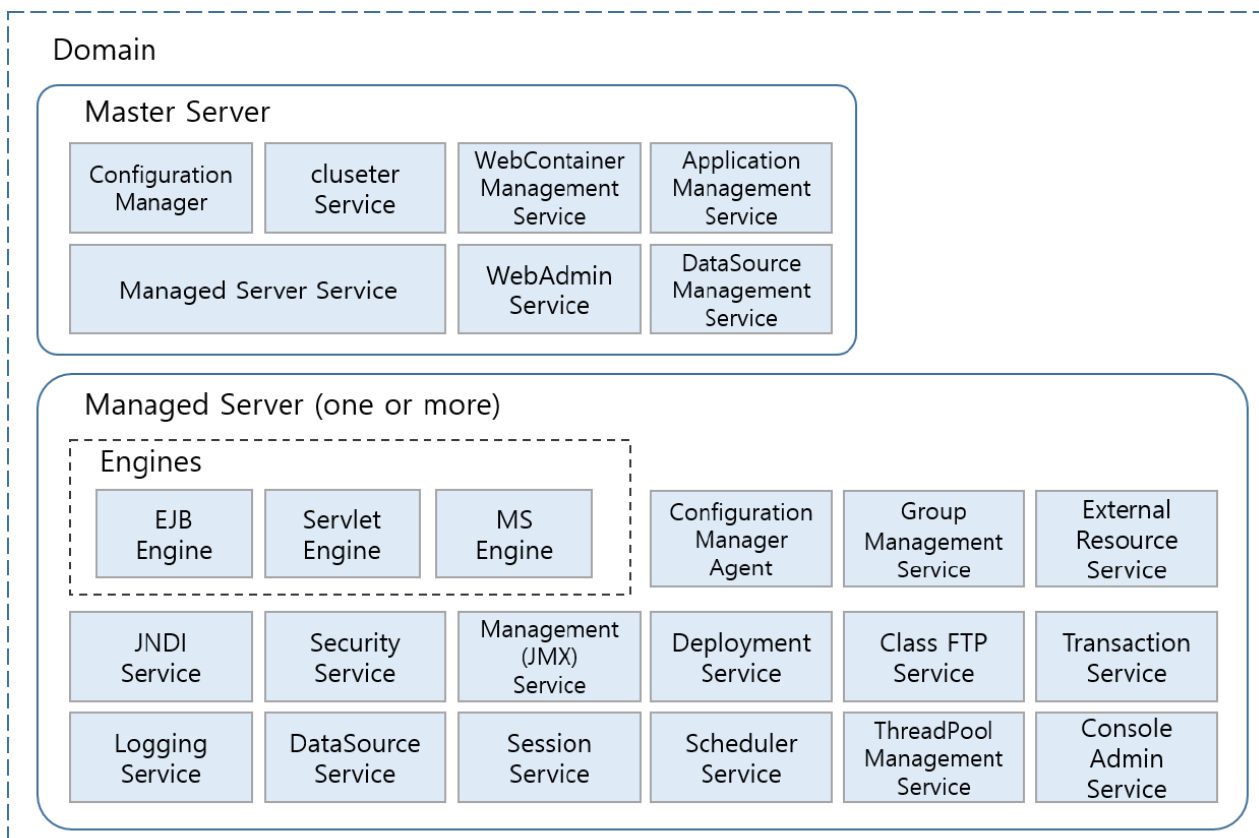
## 1.1. Components

JEUS servers can be centrally managed through the master server by grouping multiple servers in the same domain.



For more information on server startup and domain, see *JEUS Domain Guide*.

The following are featured JEUS components.



### JEUS Components

- **Domain**

Domain is a group that manages servers and clusters. A domain consists of a master server (hereafter MASTER) and one or more managed servers (hereafter MS). For more information about domains, refer to *JEUS Domain Guide*.

- **Master Server (MASTER)**

Only one MASTER can exist in a domain. It manages applications and configurations of a domain. It also manages multiple managed servers (MSs) in the domain.

The following are the main services of MASTER.

- Dynamic configuration reloading service
- Domain application management service
- Domain data source management service
- Cluster management service

- **Managed Server (MS)**

Multiple MSs can exist in a domain. An MS runs user-deployed applications, and provides services required by the applications.

The following are the main services of MS.

- EJB, WEB, and JMS engine service
- JNDI service
- Management service
- Security service
- HTTP session clustering service
- Transaction service
- Database (hereafter DB) or EIS (Enterprise Information System) connection service
- Scheduler and other services



For more information about domain configuration, refer to "Configuring a Domain" in *JEUS Domain Guide*.

### 1.1.1. Master Server (MASTER)

Master server (MASTER) is a server that manages a domain. Only one MASTER can exist in a domain. It manages domain configurations, and manages and controls applications running in the domain. It also manages MSs in the domain.



For a detailed explanation of the Master concept, see "Master Server(MASTER)" in *JEUS Domain Guide*.

The following services are the major services that can only be used in MASTER. These services can only be executed on MASTER because they have to be managed at the domain level. The following services can be used through the console tool (jeusadmin) by connecting to MASTER.

- **Dynamic configuration reloading service**

MASTER supports dynamic reloading of domain configurations on all servers in the domain.

Domain configurations can only be changed by MASTER.

The console tool (jeusadmin) can be used to execute dynamic configuration change commands. Some configuration changes can be applied dynamically, while others can only be applied after rebooting the server. This is the role of the dynamic configuration reloading service.

For more information about the service that governs changes to settings in a domain, see "Changing Domain Configuration" in *JEUS Domain Guide*.

- **Domain application management service**

MASTER manages all applications running in a domain by using the domain application management service. To run applications on MSs, the applications must be registered in the domain. Then, in MASTER, execute the **deploy** command by specifying the targets to deploy. The domain application management service synchronizes the applications' states between MASTER and MSs. If an application file is changed on MASTER, the service allows MSs to retrieve the changed file to sync with MASTER.

For more information on how to manage applications in a domain, see "Application Management in a Domain Environment" in *JEUS Applications & Deployment Guide*.

- **Domain data source management service**

In MASTER, the domain data source management service manages all data sources that are registered in a domain. MSs must use the data sources that are registered in the domain. For more information about how to register and manage data sources in the domain, refer to [DB Connection Pool and JDBC](#).

- **Cluster management service**

In MASTER, the cluster management service manages clusters. A cluster is a collection of multiple servers running the same services, and it supports load balancing and failover. For more information about clusters, refer to "JEUS Clustering" in *JEUS Domain Guide*.

- **Managed Server (MS) service**

MASTER can provide all services that are provided by MSs. In an environment where only one server is running services like in a small-sized production environment or a development environment, MASTER can also be used to service applications like an MS. For more information about MS services, refer to [Managed Server \(MS\)](#).



MASTER can be configured to run like an MS, but it is not recommended to do so. Except in an environment where only one server is required like in a small-sized production environment or a development environment, it is common to dedicate MASTER only for management and MSs for application services.

## 1.1.2. Managed Server (MS)

A Managed Server is a server instance that manages engines and services that are required to run the actual applications. Multiple MSs can be in a domain. MSs run user-deployed applications, and provide services and resources required by the applications.

The following are the key MS services.

- **Engine service**

MS provides servlet, EJB, and Jakarta™ Messaging (JMS) engines. For information about other engine services, refer to [Engine Service](#).

- **JNDI service**

JNDI service provides the JNDI standard mechanism defined by Jakarta EE. It provides methods that find various objects registered in JEUS by name.

JEUS server calls the object that provides such service as JNDI naming server. In a clustered environment, JNDI servers on each MS exchange information and work together like a single JNDI naming server. For more information about JNDI services, refer to [JNDI Naming Server](#).

- **Management service**

Management service manages and monitors services, components, and applications by using Java Management Extensions (JMX).

It configures the management objects of the JEUS Manager, such as JMX Remote API Connector (RMI Connector/JMXMP Connector), HTML adapter, and SNMP adapter that are defined in JMX. The management objects provide the methods for accessing JEUS management and monitoring information. For more information about management service configurations and management objects, refer to *JEUS JMX Guide*.

- **Security service**

Security service responds to and authenticates security requests from applications and internal components.

For more information about security services, refer to *JEUS Security Guide*.

- **HTTP session clustering service**

HTTP session clustering service maintains HTTP sessions between servlet engines.

JEUS supports distributed HTTP session clustering. The service maintains HTTP sessions even if a servlet engine error occurs. While centralized HTTP session clustering is preferred for maintaining stable sessions during server operation, distributed HTTP session clustering provides a more memory-efficient approach to session distribution. For more information about HTTP session clustering services, refer to "Session Servers" in *JEUS Session Management Guide*.

The previously described service is automatically provided in a cluster setup. For non-clustered

servers, a limited service with certain constraints is provided that maintains sessions on all the servers in a domain. For more information, refer to "Domain-Wide Session Cluster Mode" in *JEUS Session Management Guide*.

- **Class FTP service**

In EJB 2.x, RMI stub classes are required to call an EJB from a remote client. But the required classes can be obtained through the FTP Class service without packaging the RMI stub classes on the remote client.

If FTP Class service is not used, the RMI stub classes must be in the classpath of the remote client. For more examples, refer to "EJB Client" in *JEUS EJB Guide*.



It is not FTP protocol but the HTTP protocol that actually sends the class files.

EJB 3.x does not use this service because it uses dynamic proxy. EJB 2.x also uses dynamic proxy by default, but it can be configured to use the RMI stub method.

- **Scheduler service**

The Scheduler executes specific tasks at times pre-determined by the user. For more information about the Scheduler service, see *JEUS Scheduler Guide*.

- **Logging service**

Logging service records events and errors that occur on the servers to a file. The logger level or the handler that are supported by Java Logging Technology can be configured in JEUS. For more information about logging services, refer to [Logging](#).

- **Database link service**

Servers provide support for applications to access databases using JDBC connection pools. For more information, refer to [DB Connection Pool and JDBC](#).

- **Transaction service**

Servers provide support for applications to use transactions through a transaction manager. For more information about transaction managers, refer to [Transaction Manager](#).

- **External Resources**

Server provides applications with connections to various types of external resources.

External Resource	Description
Data Source	Connects to a database. (Data source defined in JDBC standards)
Mail Source	Connects to a mail server.
URL Source	Connects to a URL source.

External Resource	Description
Message Bridge	Bridge between JMS destinations.
Custom Resource	Registers custom JavaBean resources to a JNDI storage.
External Source	Connects to an enterprise information system (EIS) such as TP Monitor(Tmax) and IBM MQ. (Separate from the method that deploys the JCA resource adapter.)
JAXR Source	XML registry source.

Users can add the connection information for external resources. The information will be registered on a JNDI naming server and applications can use it for JNDI lookup operation. Although external resources are configured in the domain, this service is classified as a server service since it is used by all servers in the domain.

In a cluster setup, all servers in a cluster share the same resources by using the JNDI naming server. For more information about resource configurations, refer to [External Resources](#).

- **Enterprise Information System (EIS) connection service**

EIS can be accessed by using the registration information of the JCA resources or external resources where the application is deployed. For more information about the JCA resource adapter, refer to *JEUS Jakarta Connectors Guide*.

## Engine Service

**An engine** corresponds to an EJB container or a web container that is defined in Jakarta EE, and it manages and runs user-deployed components. An engine is included as a service on the server and even without any configuration it automatically starts up using the default settings when the server starts up.

The engine configurations can be changed using the console tool while the server is running. Refer to the relevant documentation for information about the default and dynamic configurations of each engine.

The following are the three types of service engines provided by each server.

Engine	Description
EJB Engine	Acts as an EJB container that manages and executes EJB components. For more information about this, see <i>JEUS EJB Guide</i> .
Web engine (or servlet engine)	Acts as a web container that receives requests from web clients or web servers and creates dynamic web contents through user-deployed servlets (or JSP, JSF). For more information, refer to <i>JEUS Web Engine Guide</i> .
JMS Engine	Provides Jakarta Messaging (JMS). For more information, refer to <i>JEUS MQ Guide</i> .



Up to JEUS 6, engines could not run if the engine is not configured in the engine container, and this prevented applications from running services. If the web engine was not configured, the engine had to be added and JEUS had to be restarted to run web applications. But starting from JEUS 7, the engines automatically starts up when MSs start up. This way, any type of applications can be deployed and serviced even if the user does not configure the engine.

## INDEPENDENT Mode of Managed Server

In the INDEPENDENT mode, MSs boot up and operate without using MASTER. The INDEPENDENT mode is used when the URL information of MASTER doesn't exist during startup or when MASTER is in a failed state.

The URL of MASTER must be provided as an option when starting an MS. The launcher uses this URL to access the configuration files on MASTER. If the URL information is omitted, the launcher cannot receive configuration files from MASTER.

If the MASTER URL is not provided and configuration files are cached on the server's machine, the server will start in INDEPENDENT status using those configuration files. However, if no configuration files are available, the server cannot start.

Even if the configuration files exist on the server machine, the files may be outdated and out of sync with MASTER. Therefore, the MASTER URL should always be used to start a server.

There is only one case where the MASTER URL is not required: when MASTER and MS are on the same machine, so the domain configuration files do not need to be sent to the MS. If the MASTER URL is not provided even though it is not in a failed state, the server will run in the INDEPENDENT mode, but once the group management starts on the MS, it will communicate with MASTER and will be restored to the DEPENDENT mode.

The INDEPENDENT mode is triggered more often due to an error on MASTER rather than due to missing URL. When starting an MS, if an error occurs on MASTER, the launcher cannot receive the configuration files. After failback, the MSs that booted in the INDEPENDENT mode are switched back to the DEPENDENT mode. Their configuration files are synchronized with MASTER, and the applications that have failed to deploy are redeployed.



Even if MASTER is recovered and the configurations are synchronized, MS should be restarted if the non-dynamic configurations or running applications have been modified.

The following are the logs generated when MS starts in the INDEPENDENT mode.

```
JEUS_HOME/bin$ ./startManagedServer -domain jeus_domain -server server1 -u admin -p admin -masterurl
localhost:9736
*****
- JEUS Home           : /home/jeus
```

```

- Added Java Option :
*****
===== JEUS LICENSE INFORMATION =====
=== VERSION : JEUS 9.1 (9.1.0.0)
=== EDITION: Enterprise (Trial License)
=== NOTICE: This license restricts the number of allowed clients.
=== Max. Number of Clients: 5
=====
[2024.09.25 16:12:50][0] [launcher-1] [Launcher-0052] Receiving the configuration failed. Attempting
to start as INDEPENDENT.
<<__Exception__>>
java.io.IOException: Connection failed. host:localhost, port:9736, virtual id:FileTransfer
  at jeus.net.SocketProxy.getConnection(SocketProxy.java:69)
  at jeus.net.SocketProxy.getConnection(SocketProxy.java:25)
  at jeus.server.filetransfer.ConfigurationSynchronizer.connect(ConfigurationSynchronizer.java:116)
  at jeus.server.filetransfer.ConfigurationSynchronizer.connect(ConfigurationSynchronizer.java:99)
  at
  jeus.server.filetransfer.ConfigurationSynchronizer.checkConnection(ConfigurationSynchronizer.java:141
  )
  at
  jeus.server.filetransfer.ConfigurationSynchronizer.downloadConfigFileFromMasterServer(ConfigurationSy
  nchronizer.java:181)
  at
  jeus.launcher.ManagedServerLauncher.receiveConfigurationFromMasterServer(ManagedServerLauncher.java:2
  45)
  at jeus.launcher.ManagedServerLauncher.updateXmIs(ManagedServerLauncher.java:133)
  at jeus.launcher.ManagedServerLauncher.pullLatestDomainType(ManagedServerLauncher.java:67)
  at jeus.launcher.ManagedServerLauncher.initDescriptor(ManagedServerLauncher.java:53)
  at jeus.launcher.Launcher.start(Launcher.java:146)
  at jeus.launcher.ManagedServerLauncher.start(ManagedServerLauncher.java:73)
  at jeus.launcher.ManagedServerLauncher.main(ManagedServerLauncher.java:45)
  at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
  at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
  at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
  at java.lang.reflect.Method.invoke(Method.java:498)
  at jeus.server.Bootstrapper.callMainMethod(Bootstrapper.java:576)
  at jeus.server.Bootstrapper.callMain(Bootstrapper.java:583)
  at jeus.server.Bootstrapper.main(Bootstrapper.java:151)
  at jeus.server.ManagedServerLauncherBootstrapper.main(ManagedServerLauncherBootstrapper.java:10)
<<__Exception__>>
[2024.09.25 16:12:50][1] [launcher-1] [Config-0157] SecurityDomainsConfigServiceProvider is
jeus.service.descriptor.SecurityDomainsDescriptorFile.

...
[2024.09.25 16:12:53][0] [server1-1] [SERVER-0249] Successfully started the server INDEPENDENTLY
[2024.09.25 16:12:53][2] [server1-1] [SERVER-0248] The JEUS server is RUNNING.
[2024.09.25 16:12:53][2] [server1-1] [SERVER-0401] The elapsed time to start: 4079ms.
[2024.09.25 16:12:53][2] [launcher-21] [Launcher-0034] The server[server1] initialization completed
successfully[pid : 15332].
[2024.09.25 16:12:53][0] [launcher-1] [Launcher-0040] Successfully started the server. The server
state is now RUNNING.
JEUS_HOME/bin$

```

Once MASTER is restored, MS synchronizes the configurations with MASTER and redeploys any applications that have failed to deploy.

```

[2024.09.25 16:23:20][2] [server1-55] [Domain-0101] JEUS Master Server recovered. server1 is

```



communicating with the JEUS Master server.

[2024.09.25 16:23:20][2] [server1-55] [SERVER-0201] Successfully connected to the JEUS Master Server(192.168.14.63:9736).

[2024.09.25 16:23:20][2] [server1-55] [SERVER-0308] Resynchronized the configuration with JEUS Master Server.

## 1.2. Class Loader Structure

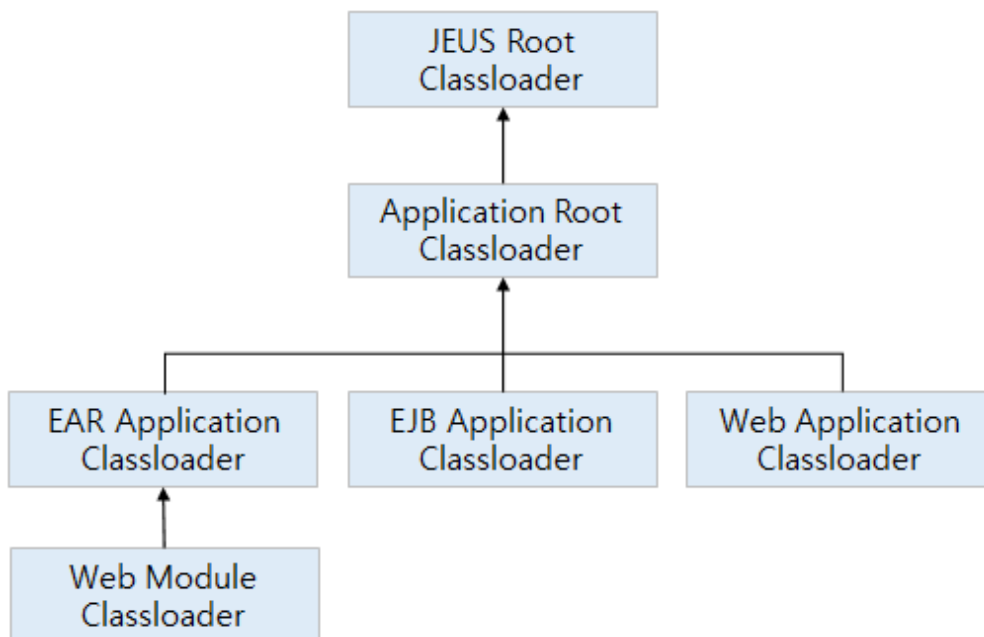
This section describes the Isolated Class Loader supported in JEUS.



By default, the Shared Class Loader (hereafter Shared) is used in JEUS 5, and it is also supported in JEUS 7 for backward compatibility. However, it needs to be manually configured, otherwise the Isolated method is used by default. This guide does not cover shared class loader.

Applications use a separate class loader to prevent duplicate classes in JEUS. This is called **Isolated Class Loader** (hereafter Isolated). JEUS complies with the Jakarta EE standards that recommend to use the Isolated Class Loader.

The following is the Isolated Class Loader hierarchy.



Isolated Class Loader Hierarchy

Each web module class loader of an application exists under an EJB class loader of the application. A class loader of an application does not request a class from a class loader of another application. When an application needs to use the interfaces of another application, it cannot reference the class loader of the application because Jakarta EE standards require that the interfaces be packaged together.

Since the classes are not shared, when deploying an application other applications do not need to be

redeployed. The proper use of this class loader requires the binding of related applications into an EAR file to create a single application.

## 1.3. Server Directory Structure

Each server has its own directory under the DOMAIN\_HOME folder. They exist in the servers directory under DOMAIN\_HOME. Each server has a directory with its server name and stores information that is required to start and operate the server. This directory is called SERVER\_HOME.

The following is the directory structure of SERVER\_HOME.

```
{DOMAIN_HOME}
|--servers
|   |--server1
|       |--.workspace
|           |--deployed
|           |--ejbtimerdb
|           |--tmlog
|           |--tmp
|       |--bin
|       |--lib
|       |--application
|       |--logs
|       |--nodemanager
```

### .workspace

Space used by each server in JEUS. Must not be modified by the user.

The following table describes each sub-directories.

Directory	Description
deployed	<p>Contains the unzipped image files of the applications deployed on the server and the files that were created when the applications were deployed. In this directory, directories are created by using application IDs to store the application files received from MASTER. Archived applications will be unzipped by using the application ID (deploy image).</p> <p>Under the deployed directory, a directory named "_generated_" is created to store the files created when the applications are deployed. In this directory, directories are created by using application IDs to store the files required for deployment.</p>
ejbtimerdb	<p>Directory that is used for Apache Derby (a file-based database that is embedded in JEUS) when the timer services of the EJB engine does not have database configuration. For more information about timer services, refer to "EJB Timer Service" in <i>JEUS EJB Guide</i>.</p>

Directory	Description
tmlog	<p>Directory that contains transaction logs required for transaction recovery.</p> <p>A sub directory is created with the name "_serverName_LOCATION_type_port_ipaddress_virtualport name". The following values are set for the type and virtualhost of the created directory name.</p> <ul style="list-style-type: none"> <li>◦ type: 0 for IPv4 and 1 for IPv6</li> <li>◦ virtualport: hash value of the server name</li> </ul> <p>For more information, refer to <a href="#">Recovery Log File</a>.</p>
tmp	<p>Directory that is used when temporarily saving files during server operation.</p> <p>There is no particular reason for users to access this directory.</p>

## bin

Contains the scripts for starting and stopping the server in SERVER\_HOME. The scripts execute the same functions as the scripts in 'JEUS\_HOME/bin', but the domain name and server name do not need to be specified.

The 'startMasterServer/stopserver' script is used for MASTER, and 'startManagedServer/stopserver' script is used for MS. For more information about the scripts, refer to "Starting and Ending JEUS Servers" in *JEUS Reference Guide*.

## lib/application

Contains application libraries that are applied to the server.

The files in this directory is used when a library conflicts with a domain-level library in the 'DOMAIN\_HOME/lib/application' directory. A warning indicating that there is a library conflict is displayed and the domin\_level library is ignored on the server. For more information about 'lib/application', refer to "lib/application Directory" in *JEUS Applications & Deployment Guide*.

## logs

Contains the launcher log, server log, and access log files. According to the rotation rule, each log file is created as "logname\_date.log00000" where "00000" represents a number between 1 and 99999. For more information, refer to [Logging](#).

## nodemanager

Contains configuration information and logs required for the node manager. For more information about node mangers, refer to *JEUS Node Manager Guide*.

# 1.4. Launcher

In JEUS, both MASTER and MS are started by a launcher. A launcher process is executed when starting a server by using a script or starting MS from MASTER. The Launcher is responsible for preparing the server for startup and for starting up the server.

The launcher performs the following tasks:

- Receive domain configuration files from MASTER.
- Start the server.

First, the launcher receives the domain configuration files from MASTER. The files are **domain.xml** and **security configuration files**. When the launcher receives the files, it uses them to start the server JVM. When starting up the server, it reads JVM options and classpaths from the files and uses the options to create the server JVM.

If the launcher process fails to receive the configuration files but there are domain configuration files that are cached on the machine, the launcher will use them to start the server JVM. If the launcher fails to receive the configuration files and there are no cached files, the server cannot be started.

Besides the 2 aforementioned tasks, the launcher process acts as a logger for the server boot logs.

The logs that are generated when the server starts up are logged in the launcher logs. The launcher process starts the server JVM and waits for the server to complete the boot process and records logs that have been generated when the server finishes starting up. If the server fails to boot before the server resets the logger, the launcher logs can be used to check for the cause of the failure.

Generally, the launcher process starts the server JVM and terminates when the server starts up successfully. Note that the server process started by the launcher operates as a background process. Since the server process is started as a child process of the launcher process, it does not have its own console screen.

Hence, to print the server logs on a console screen, the '-verbose' option should be used to start a server. If the '-verbose' option is used, the launcher process does not terminate and it outputs the logs on the screen until the server is terminated. For more information, refer to [Launcher Logger](#).

## 2. JEUS Configuration

This chapter describes the methods and required configurations for adding servers to a domain. It also describes how to change the server configuration.

### 2.1. Overview

In JEUS, you can use the console tool to change server configurations. If applications that run on the server engine do not require security authentication and authorization checks, the security functions should be disabled on JEUS.

In addition, tuning of the sub-components must be checked separately. For more information, refer to the relevant chapter for each component.

### 2.2. Adding Servers

This section describes how to add servers to a domain by using the console tool. It also describes the minimum required configurations for adding a server. For more information, refer to [Server Configuration](#).

#### 2.2.1. Using the Console Tool

A server can be added through the **add-server** command in the console tool. Since only some configurations can be added through this command, other commands must be used to add additional configurations after the server has been added. To change the engine configuration, use the web engine, EJB engine, and JMS engine commands.



For information about commands to modify web engine settings ("Web Engine Commands"), EJB engine settings ("EJB Engine Commands"), and JMS engine settings ("JMS Engine Commands"), see the respective section in *JEUS Reference Guide*.

The following is an example of adding a server using jeusadmin.

```
[MASTER]domain1.adminServer>server-info
Information about Domain (domain1)
```

=====									
Server	Status	Node Name	PID	Clu ster	Latest Start Time / Shutdown Time	Need to Restart	Listen Ports	Running Engines	

adminS	RUNNING	nod	572	N/A	2024-09-25	false	base-0.0.	jms,
server	(00:18:2	e1	02		(Thu) PM		0.0:9736	web, ejb
(*)	0)				01:28:24		http-serv	
					KST		er-0.0.0.0	
							:8088	
+-----+-----+-----+-----+-----+-----+-----+-----+-----+								
server1	RUNNING	nod	589	N/A	2024-09-25	false	base-0.0.	jms,
	(00:00:0	e1	25		(Thu) PM		0.0:9836	web, ejb
	8)				01:46:36		http-serv	
					KST		er-0.0.0.0	
							:8188	
+-----+-----+-----+-----+-----+-----+-----+-----+-----+								

```
[MASTER]domain1.adminServer>add-server server2 -addr 192.168.15.59 -port 9936 -node node1 -jvm "-Xmx512m -XX:MaxPermSize=128m"
```

Successfully performed the ADD operation for server (server2).

NOTICE : base-addr [192.168.15.59] base-port [9936] http-port [8088]

Check the results using "list-servers or add-server".

```
[MASTER]domain1.adminServer>modify-system-thread-pool server2 -max 200
```

Successfully performed the MODIFY operation for the system thread pool of the server (server2), but all changes were non-dynamic. They will be applied after restarting.

Check the results using "modify-system-thread-pool server2 or show-system-thread-pool server2".

```
[MASTER]domain1.adminServer>server-info
```

Information about Domain (domain1)

Server	Status	Node	PID	Clu	Latest	Need	Listen	Running
		Name		ster	Start Time	to	Ports	Engines
					/	Restart		
					Shutdown			
					Time			
+-----+-----+-----+-----+-----+-----+-----+-----+-----+								
adminS	RUNNING	nod	572	N/A	2024-09-25	false	base-0.0.	jms,
server	(00:38:1	e1	02		(Thu) PM		0.0:9736	web, ejb
(*)	1)				01:28:24		http-serv	
					KST		er-0.0.0.0	
							:8088	
+-----+-----+-----+-----+-----+-----+-----+-----+-----+								
server1	RUNNING	nod	589	N/A	2024-09-25	false	base-0.0.	jms,
	(00:19:5	e1	25		(Thu) PM		0.0:9836	web, ejb
	9)				01:46:36		http-serv	
					KST		er-0.0.0.0	
							:8188	
+-----+-----+-----+-----+-----+-----+-----+-----+-----+								
server2	SHUTDOWN	nod	N/A	N/A	2024-09-25	N/A	N/A	N/A
		e1			(Thu) PM			
					02:04:50			
					KST			
+-----+-----+-----+-----+-----+-----+-----+-----+-----+								

## 2.3. Server Configuration

You can change server configuration using the console tool in JEUS.

### 2.3.1. Environment Variable Configuration

This section describes the environment variables necessary for the operation of various console tools provided by the server and JEUS. Environment variables cannot be configured by using console tools.

The environment variables required for JEUS operation are automatically set as default values in the JEUS\_HOME/bin/jeus.properties file for UNIX during the installation process. The files are written in a shell script supported by the OS. If necessary, the user can modify the file to add or edit environment variables.

The following describes the key environment variables used in JEUS.

Environment Variable	Description
JEUS_HOME	JEUS installation directory. (Example: /home/jeus)
JEUS_LIBPATH	JEUS library file path. (Example: /home/jeus/lib/system)
VM_TYPE	Virtual machine type. (Example: hotspot or old)
JEUS_USERNAME	Administrator account ID.
JEUS_PASSWORD	Administrator account password.
JAVA_HOME	JDK installation directory. (Example: /usr/jdk17)
JAVA_ARGS	JDK parameters.
JAVA_VENDOR	JDK vendor. (Example: Sun, IBM, or HP)

The environment variables specified in jeus.properties apply to all servers and tools that reference the script.

To specify different environment variables for each server, proceed as follows:

1. Create an environment variable file or a shell script file in the following path.

```
JEUS_HOME/bin/<DOMAIN_NAME>.<SERVER_NAME>.properties
```

2. Modify the file created in step 1 to set environment variables by referring to the jeus.properties or jeus.properties.cmd file. When specifying an environment variable, remove the comment prefix ('#') before it.

The following is a sample environment variable file.

```
#####
```

```
# This part is for booting JEUS automatically. #
# BE CAREFUL!! THIS IS ONLY FOR TEST AND DEVELOPMENT ENVIRONMENT. #
#####

# Set up administrator name
# JEUS_USERNAME=

# Set up administrator password
# JEUS_PASSWORD=
```

The following is an example of starting the server.

```
JEUS_HOME/bin$startMasterServer -server adminServer
```

## 2.3.2. Basic Configuration

This section describes the basic configurations. For configurations that are not described here, refer to the relevant guides.

### 2.3.2.1. Dynamically Changing Basic Configuration

Among the basic settings of the server, **Class FTP**, **Use MEJB**, and **Log Stdout To Raw Format** can be changed dynamically. The settings that are applied dynamically can be checked by executing the **modify-server** command through the **help** command in the console tool. For detailed information on each command, see *JEUS Reference Guide*.

### Using the Console Tool

The following is an example of dynamically changing the basic configurations by using the console tool.

```
[MASTER]domain1.adminServer>modify-server server2
Shows the current configuration.
server (server2)
=====
+-----+-----+
| Node           | node1           |
| JVM Configs    | -Xmx512m -XX:MaxPermSize=128m |
| Action On Resource Leak | WARNING        |
| Stdout to Raw Format | true           |
| MEJB           | false           |
| Class FTP      | false           |
| Server Log Home Directory | none           |
+-----+-----+
=====

[MASTER]domain1.adminServer>modify-server server2 -logStdoutToRawFormat false -mejB true -classFtp
true
Successfully performed the MODIFY operation for server (server2).
Check the results using "list-servers server2 or modify-server server2"
```



```
[MASTER]domain1.adminServer>modify-server server2
Shows the current configuration.
server (server2)
=====
+-----+-----+
| Node                | node1                |
| JVM Configs         | -Xmx512m -XX:MaxPermSize=128m |
| Action On Resource Leak | WARNING              |
| Stdout to Raw Format  | false                |
| MEJB                | true                 |
| Class FTP           | true                 |
| Server Log Home Directory | none                 |
+-----+-----+
=====
```

### 2.3.2.2. Action On Resource Leak

The 'Action On Resource Leak' setting allows the user to know when there is resource leak and performs the configured operations by checking whether the resources that are used on the server are closed. This server function is called the **invocation manager**.

On the server, the invocation manager tracks the external resources like JDBC Connections and Webt Con nections that are used during a call of stateless methods such as Servlet/JSP, EJB Stateless Session Bean, and MDB. It leaves a log or returns the resource depending on the mode configured among the following three modes:

Mode	Description
NoAction	No action is taken even if there are unreturned resources.
Warning	Logs about unreturned resources after the component call (default value). Add an SMTP handler to receive email notifications.
AutoClose	Logs about unreturned resources and closes them after the component call. Add an SMTP handler to receive email notifications.

The following is an example of changing the 'Action On Resource Leak' setting by executing the modify-server command in the console tool.

1. Check the current server configurations that need to be modified by using the **modify-server** command before booting the server.

```
[MASTER]domain1.adminServer>modify-server server2
Shows the current configuration.
server (server2)
=====
+-----+-----+
| Node                | node1                |
| JVM Configs         | -Xmx512m -XX:MaxPermSize=128m |
| Action On Resource Leak | WARNING              |
+-----+-----+
=====
```

Stdout to Raw Format	false
MEJB	true
Class FTP	true
Server Log Home Directory	none

=====

- Set the 'Action On Resource Leak' setting of server2 to 'AutoClose'.

```
[MASTER]domain1.adminServer>modify-server server2 -actionOnResourceLeak AutoClose
Successfully performed the MODIFY operation for server (server2).
Check the results using "list-servers server2 or modify-server server2".
```



When options are not dynamically applied, they must be modified before starting the server. If static configurations are modified at runtime, the server must be restarted to apply the changes.

- Verify that the modified configurations have been applied correctly by using the **modify-server** or **list-servers** command.

```
[MASTER]domain1.adminServer>modify-server server2
Shows the current configuration.
server (server2)
=====
+-----+-----+
| Node           | node1           |
| JVM Configs    | -Xmx512m -XX:MaxPermSize=128m |
| Action On Resource Leak | AUTO_CLOSE      |
| Stdout to Raw Format | false          |
| MEJB           | true            |
| Class FTP       | true            |
| Server Log Home Directory | none           |
+-----+-----+
=====
```

- Start the server 'server2'.

```
[MASTER]domain1.adminServer>start-server server2
The server(server2) was successfully started. The server is [RUNNING]
```



To use the start-server command in the console tool, node configuration is required. For more information about the node manager, refer to *JEUS Node Manager Guide*.

- Change the 'Action On Resource Leak' setting of server2 from 'AUTO\_CLOSE' to 'WARNING'.

This configuration cannot be applied while the server is running. To apply the modified configuration, restart the server.

```
[MASTER]domain1.adminServer>modify-server server2 -actionOnResourceLeak Warning
Successfully performed the MODIFY operation for server (server2), but all changes were non-
dynamic. They will be applied after restarting.
Check the results using "list-servers server2 or modify-server server2".
```

6. Verify that the modified configurations have been applied correctly by using the **modify-server** or **list-servers** command.

```
[MASTER]domain1.adminServer>modify-server server2
Shows the current configuration.
server (server2)
=====
+-----+-----+
| Node                | node1                |
| JVM Configs         | -Xmx512m -XX:MaxPermSize=128m |
| Action On Resource Leak | WARNING              |
| Stdout to Raw Format  | false                |
| MEJB                | true                 |
| Class FTP            | true                 |
| Server Log Home Directory | none                 |
+-----+-----+
=====
```



For detailed explanations of the console tool commands used above, see "Server Management Commands" in *JEUS Reference Guide*.

### 2.3.2.3. JVM Config

**JVM Config** is used to define parameters that are added to an individual JVM to start a server. Before starting the server with the launcher process, values specified in this section are read and added as parameters to create the server JVM. Refer to "Server System Properties" for the list of available JEUS parameters. The standard JVM parameters can also be configured.

JVM Config also includes JVM options that are applied to the server, system properties, and the system properties provided by JEUS. JVM memory or options are usually configured with values that are suitable for the server operating environment. For more information about JVM Config, refer to "Changing JVM Configuration in Server" in *JEUS Domain Guide*.



JVM config values cannot be applied while a server is running because they are not dynamic configurations. The server must be restarted in order to apply the changes.

## Using the Console Tool

You can change the JVM configurations on the server using the **modify-server**, **add-jvm-option**, **modify-jvm-option**, and **remove-jvm-option** commands of the console tool. For more information about the commands, see JEUS Domain Guide's "Changing JVM configuration in Server", and JEUS Reference Guide's "Server Management Commands".

The following is an example of changing the JVM configurations of a server by using the **modify-server** command.

1. Check the current server configurations that need to be modified by using the **modify-server** command before booting the server.

```
[MASTER]domain1.adminServer>modify-server server1
Shows the current configuration.
server (server1)
=====
+-----+-----+
| Node                                | node1 |
| Action On Resource Leak             | WARNING |
| Stdout to Raw Format                 | true  |
| MEJB                                | false |
| Class FTP                           | false |
| Server Log Home Directory           | none  |
+-----+-----+
=====
```

2. Add the JVM option to server1.

Set the maximum heap memory of JVM to 512MB and the maximum permanent memory to 128MB.

```
[MASTER]domain1.adminServer>modify-server server1 -jvmOptions "-Xmx512m -XX:MaxPermSize=128m"
Successfully performed the MODIFY operation for server (server1).
Check the results using "list-servers server1 or modify-server server1".
```



Options that cannot be dynamically applied must be changed before starting the server. When the configuration is changed while the server is running, the server must be restarted to apply the changes.

3. Check that the configurations changed by executing **modify-server** or **list-servers** have been applied correctly.

```
[MASTER]domain1.adminServer>modify-server server1
Shows the current configuration.
server (server1)
=====
+-----+-----+
| Node                                | node1 |
+-----+-----+
```

JVM Configs	-Xmx512m -XX:MaxPermSize=128m
Action On Resource Leak	WARNING
Stdout to Raw Format	true
MEJB	false
Class FTP	false
Server Log Home Directory	none
+-----+-----+	
=====	

#### 4. Add the JVM option to server1.

An option that creates a Heap Dump file has been added for when `OutOfMemoryError` occurs on the server. Since this configuration cannot be applied to the server while the server is running, the server must be restarted to apply the changes.

```
[MASTER]domain1.adminServer>modify-server server1 -jvmOptions "-XX:+HeapDumpOnOutOfMemoryError"
Successfully performed the MODIFY operation for server (server1).
Check the results using "list-servers server1 or modify-server server1".
```

#### 5. Verify that the modified configurations have been applied correctly by using the **modify-server** or **list-servers** command.

```
[MASTER]domain1.adminServer>modify-server server1
Shows the current configuration.
server (server1)
=====
+-----+-----+
| Node           | node1           |
+-----+-----+
| JVM Configs    | -Xmx512m -XX:MaxPermSize=128m, |
|                | -XX:+HeapDumpOnOutOfMemoryError |
+-----+-----+
| Action On Resource Leak | WARNING         |
+-----+-----+
| Stdout to Raw Format  | true           |
+-----+-----+
| MEJB             | false          |
+-----+-----+
| Class FTP         | false          |
+-----+-----+
| Server Log Home Directory | none           |
+-----+-----+
=====
```

### 2.3.2.4. Classpath

To add the classpath to the server, you need to change the configuration in `domain.xml`.

```
<server>
...
<user-interceptor>
```

```

    <jeus-classloader-append-class-path>/home/jeus/lib/mylib.jar</jeus-classloader-append-class-
path>
    <jeus-classloader-append-dirs>/home/jeus/lib/append</jeus-classloader-append-dirs>
    <boot-classloader-append-class-path>/home/jeus/boot/classes:/home/jeus/boot/a.jar</boot-
classloader-append-class-path>
    </user-interceptor>
</server>

```

The following describes each setting item. If you have multiple items to add to the classpath, separate each item using a path separator.

Item	Description
jeus-classloader-append-class-path	Specifies items to add to the class path of the JEUS root class loader (JEUS Root Classloader).
jeus-classloader-append-dirs	Adds all classpath entries belonging to the specified directory to the classpath of the JEUS root classloader (JEUS Root Classloader).
boot-classloader-append-class-path	Specifies items to add to the system classloader classpath of the server JVM.

### 2.3.3. Listener Configuration

This section describes the network listener configuration that is used on a server.

The network listener configuration is referenced by system services or various types of engines. Since port-integrated services are applied to this listener, it can be shared and used by different services or engines. As a simplified example, a basic listener can be used to provide all services.



Since this configuration cannot be applied to the server while the server is running, the server must be restarted to apply the changed configuration.

### Using the Console Tool

You can set a default listener using the **add-listener** command in jeusadmin.

```

[MASTER]domain1.adminServer>add-listener -server server1 -name TestListener -addr 192.168.14.145
-port 8188
Executed successfully, but some configurations were not applied dynamically. It might be necessary to
restart the server.
Check the result using 'list-server-listeners -server server1 -name TestListener.'

```

## 2.3.4. Thread Pool Configuration

This section describes how to configure common thread pools used on a server.

If the services used on a server are not configured with a dedicated thread pool, then they all use the common thread pool. The transaction service, JNDI service, and scheduler service use the common thread pool. These services can either use the common thread pool or a dedicated thread pool depending on the configuration. If the common thread pool is used, the allowed minimum number of threads for use can be assigned in advance.



The engine that handles requests from applications does not use a common thread pool. For more information about how to use a dedicated thread pool for a service, refer to [JNDI Naming Server](#), [Transaction Manager](#), and [JEUS Scheduler Guide](#).

### Using the Console Tool

This section shows how to check and modify a common thread pool by using the console tool.



For more information about the commands used in the examples, see "Thread Management Commands" in *JEUS Reference Guide*. For details on how to configure the thread pool dedicated to a service, refer to relevant documentation of each service.

The following is an example of changing the 'Max' from 100 to 200 and 'Keep Alive Time' from 5 minutes to 10 minutes for the thread pool.

```
[MASTER]domain1.adminServer>show-system-thread-pool server1
```

Shows the current configuration.

the system thread pool of the server (server1)

=====		
+-----+-----+		
Min	0	
Max	100	
Keep-Alive Time	300000	
Queue Size	4096	
Max Stuck Thread Time	3600000	
Action On Stuck Thread	IGNORE_AND_REPLACE	
Stuck Thread Check Period	300000	
Reserved Threads for the Service transaction	0	
Reserved Threads for the Service namingserver	0	
+-----+-----+		
=====		

```
[MASTER]domain1.adminServer>modify-system-thread-pool server1 -max 200 -keep 600000
```

Successfully performed the MODIFY operation for the system thread pool of the server (server1), but all changes were non-dynamic. They will be applied after restarting.

Check the results using "modify-system-thread-pool server1 or show-system-thread-pool server1".

```
[MASTER]domain1.adminServer>show-system-thread-pool server1
```

Shows the current configuration.

the system thread pool of the server (server1)

```
=====
```

+-----+-----+		
Min	0	
Max	200	
Keep-Alive Time	600000	
Queue Size	4096	
Max Stuck Thread Time	3600000	
Action On Stuck Thread	IGNORE_AND_REPLACE	
Stuck Thread Check Period	300000	
Reserved Threads for the Service transaction	0	
Reserved Threads for the Service namingserver	0	
+-----+-----+		
=====		

The following is an example of configuring the thread to be preallocated in the common thread pool of the JNDI service.

```
[MASTER]domain1.adminServer>show-system-thread-pool server1
```

Shows the current configuration.

the system thread pool of the server (server1)

```
=====
```

+-----+-----+		
Min	0	
Max	200	
Keep-Alive Time	600000	
Queue Size	4096	
Max Stuck Thread Time	3600000	
Action On Stuck Thread	IGNORE_AND_REPLACE	
Stuck Thread Check Period	300000	
Reserved Threads for the Service transaction	0	
Reserved Threads for the Service namingserver	0	
+-----+-----+		
=====		

```
[MASTER]domain1.adminServer>modify-system-thread-pool server1 -service namingserver -r 10
```

Successfully performed the MODIFY operation for The namingserver thread pool of the server

(server1)., but all changes were non-dynamic. They will be applied after restarting.

Check the results using "show-system-thread-pool server1 -service namingserver or modify-system-thread-pool server1 -service namingserver".

```
[MASTER]domain1.adminServer>show-system-thread-pool server1 -service namingserver
```

Shows the current configuration.

the system thread pool of the server (server1)

```
=====
```

+-----+-----+		
Min	0	
Max	200	
Keep-Alive Time	600000	
Queue Size	4096	
Max Stuck Thread Time	3600000	
Action On Stuck Thread	IGNORE_AND_REPLACE	
Stuck Thread Check Period	300000	
Reserved Threads for the Service transaction	0	
Reserved Threads for the Service namingserver	10	
+-----+-----+		
=====		



```

+-----+-----+
=====

[MASTER]domain1.adminServer>modify-system-thread-pool server1 -service namingserver
Shows the current configuration.
The namingserver thread pool of the server (server1).
=====
+-----+-----+
| Reserved Threads for the Service namingserver                | 10 |
+-----+-----+
=====

```

## 2.3.5. Lifecycle Invocation Configuration

JEUS offers Lifecycle Invocation to allow users to perform tasks in accordance with the server lifecycle. The server invokes the user-configured event when the computer starts up or shuts down.



If you change the class name while the server is running, the server must be restarted to apply the changes. Other invocation settings are applied dynamically.

The following is an example of registering a typical Java class for Lifecycle Invocation.

Class for Lifecycle Invocation: <LifeCycleTester.java>

```

package lifecycle;

public class LifeCycleTester {
    public void boot() {
        System.out.println("Boot");
        // do somethig
    }

    public void beforeDeploy() {
        System.out.println("Before Deploy");
        // do somethig
    }

    public void afterDeploy() {
        System.out.println("After Deploy");
        // do somethig
    }

    public void ready() {
        System.out.println("Ready");
        // do somethig
        try {
            System.out.println("Sleeping for 15 seconds ....");
            Thread.sleep(15000L);
        } catch (Exception e) {
            //ignored
        }
    }
}

```

```

public void beforeUndeploy() {
    System.out.println("Before Undeploy");
    // do somethig
}

public void afterUndeploy() {
    System.out.println("After Undeploy");
    // do somethig
}
}

```

## Using the Console Tool

The following describes the process of configuring the Lifecycle Invocation class of the server by using the previous example.

In the console tool, you can add a Lifecycle Invocation of the server by using the **add-lifecycle-invocation** command, and add the library reference and Invocation with the **add-invocation-library** and **add-invocation** commands. For detailed information, see "add-lifecycle-invocation", "add-invocation-library", and "add-invocation" in *JEUS Reference Guide*.

```

[MASTER]domain1.adminServer>add-lifecycle-invocation lifecycle.LifecycleTester -s adminServer -m boot
-type BOOT

```

Successfully performed the ADD operation for Lifecycle Invocation Class [lifecycle.LifecycleTester] and Invocation [boot](Invocation ID = 0), but all changes were non-dynamic. They will be applied after restarting.

Check the results using "list-lifecycle-investigations".

```

[MASTER]domain1.adminServer>list-lifecycle-investigations

```

List of Lifecycle investigations

```

=====
+-----+-----+-----+-----+
| Target | Lifecycle Invocation Class | Invocation | Invocation |
|         |                             | Library Ref |             |
+-----+-----+-----+-----+
| [Server]admin | test.lifecycle.invocation.Lif | lib1 | [1]boot |
| Server        | eCycleInvocation              |      |         |
+-----+-----+-----+-----+
| [Server]admin | lifecycle.LifecycleTester     |      | [0]boot |
| Server        |                               |      |         |
+-----+-----+-----+-----+

```

```

=====
Use the "lifecycle-invocation-info" command for more information about Lifecycle invocation.

```

## 2.3.6. Resource Reference Configuration

Configure the common mapping information for the resources used by the applications on the server.

The names of the resources that are used in the applications are registered in the JNDI storage of the server where each application is serviced. When resource mapping is configured, regardless of the registered name, the applications can always use the resource by using the same name.



When a server is in a cluster, the value configured in the cluster is applied first to the resource reference setting.

You can configure resource references by editing domain.xml as follows:

```
<server>
  <name>server1</name>
  ...
  <res-ref>
    <jndi-info>
      <ref-name>/jdbc/DB1</ref-name>
      <export-name>db1</export-name>
    </jndi-info>
  </res-ref>
</server>
```

## 2.4. Server Template Configuration

Common configurations of the servers to be used when creating a cluster can be stored as a server template. By doing so, multiple servers with a common configuration can be automatically created when creating a cluster, which allows for easy creation of a server cluster environment.

### Using the Console Tool

A server template can be added by using the console tool. Add a server template by using the **add-server-template** command, and then configure detailed items via various options. For more information about add-server-template, refer to "add-server-template" in *JEUS Reference Guide*.

```
[MASTER]domain1.adminServer>add-server-template template1 -m true
Successfully performed the ADD operation for server template (template1).
NOTICE : base-addr [0.0.0.0] base-port [9736] http-port [8088]
Check the results using "list-servers or show-server-template or add-server-template".
```

```
[MASTER]domain1.adminServer>show-server-template
Shows the current configuration.
Server template list
=====
+-----+-----+
| server templates                | template1 |
+-----+-----+
=====
```

# 3. Controlling and Monitoring JEUS Servers

This chapter describes how to control and monitor JEUS servers. Only managed servers (MSs) are covered in this chapter with the assumption that the Master Server (MASTER), which manages the servers, is under normal operation.



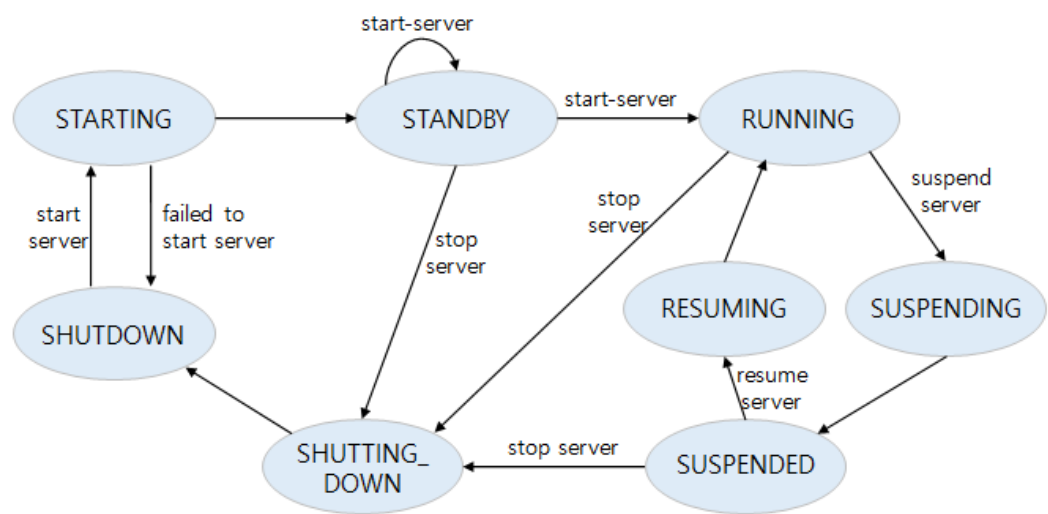
For more information about server execution scripts and console tools, see "JEUS Server Startup and Shutdown" and "Local Commands" in *JEUS Reference Guide*.

## 3.1. Controlling and Monitoring Servers

This section describes how to control and monitor servers.

### 3.1.1. Life Cycle of Managed Servers

The following shows the possible state transitions of an MS. MASTER or the user can change the state of an MS.



State Transitions of a Managed Server (MS)

Division	Description
SHUTDOWN	Server has not been started or has shut down normally.
STARTING	Server has received a start command and is in the process of starting up.  During startup, the server creates the engines (WEB, EJB, and JMS), starts JEUS services (security and SCF), and deploys registered applications, etc.
STANDBY	After the required JEUS services have started on the server, if applications that are registered on the server fail to be deployed or distributed, the server goes into the STANDBY state.

Division	Description
RUNNING	A server has started and the registered applications are running normally. If a server that was in the STANDBY state is started by the force option, not all registered applications may be running.
SUSPENDING	A running server is in the process of executing the suspend command.  Except for listeners used for application services, all running services on all applications are stopped in this state.
SUSPENDED	All applications that are deployed on the server are stopped. No services are running except for the listeners used for application services.
RESUMING	A server in the SUSPENDED state is in the process of executing the resume command.  All stopped applications are restarted and ready to provide services. After the execution of the resume command, the server goes into the RUNNING state.
SHUTTING_DOWN	A server is in the process of shutting down. The deployed applications are undeployed, and all JEUS services that were started at startup are terminated.  When a server is shutting down, a timeout setting is used to allow the currently running services to be processed. This is called a 'graceful shutdown'.



1. For more information about the lifecycle of MSs managed by MASTER, refer to "Monitoring Server Lifecycle Status" in *JEUS Domain Guide*.
2. For more information about graceful shutdowns, refer to [Shutting Down a Managed Server](#).

### 3.1.2. Starting a Managed Server

By default, MASTER and MS are started by a launcher. Server startup commands via a script or MASTER is used to start the launcher, and the launcher prepares for the server startup and starts the server. For more information about the launcher, refer to [Launcher](#). For information about how to start a server using the console tool, refer to [Starting a Managed Server \(MS\)](#) in *JEUS Domain Guide*. Note that this section only provides examples of starting servers by using scripts.

#### Example

A launcher terminates after starting JVM of a server and confirming that the server has started.

In the following example, the launcher log is used to confirm that the server has been started successfully.

```
JEUS_HOME/bin$ ./startManagedServer -domain jeus_domain -server server1 -u jeus -p jeus -masterurl
```

```
localhost:9736
*****
- JEUS Home      : /home/jeus
- Java Vendor    : Sun
- Added Java Option :
*****

...
[2024.09.25 22:35:04][2] [launcher-1] [SERVER-0201] Successfully connected to the JEUS Master
Server(localhost:9736).
[2024.09.25 22:35:04][2] [launcher-1] [Launcher-0058] All local configurations are up-to-date.
...
[2024.09.25 22:35:07][0] [server1-1] [SERVER-0242] Successfully started the server.
[2024.09.25 22:35:07][2] [server1-1] [SERVER-0248] The JEUS server is RUNNING.
[2024.09.25 22:35:07][2] [launcher-22] [Launcher-0034] The server[server1] initialization completed
successfully[pid : 81719].
[2024.09.25 22:35:07][0] [launcher-1] [Launcher-0040] Successfully started the server[server1]. The
server state is now RUNNING.
JEUS_HOME/bin$
```

Problems may occur when an MS deploys its application files during startup. In such cases, an MS remains in the STANDBY state. The application files that are deployed during startup have been executed successfully one or more times. Hence, if there is a problem, then it is likely that the files have been modified incorrectly.

If the XML tag of the application in the .workspace directory, an internal JEUS directory, is modified incorrectly, the server may remain in the STANDBY state as shown in the following example. For more information about deployment of the domain structure, refer to "Application Management in the Domain Environment" in *JEUS Applications & Deployment Guide*.



Since applications are generally managed by MASTER, the .workspace directory of the MS must not be accessed directly.

```
JEUS_HOME/bin$ ./startManagedServer -domain jeus_domain -server server1 -u jeus -p jeus -masterurl
localhost:9736
...
[2024.09.25 23:20:07][2] [launcher-1] [SERVER-0201] Successfully connected to the JEUS Master
Server(localhost:9736).
...
[2024.09.25 23:20:09][2] [server1-1] [Deploy-0095] Distributing the application[healthcheck].
java.lang.RuntimeException: [was class com.ctc.wstx.exc.WstxParsingException] Unexpected close tag
</urlpattern>; expected </url-pattern>.
[2024.09.25 23:20:09][0] [server1-1] [SERVER-0522] An exception occurred while processing
[SERVER_HOME/.workspace/deployed/healthcheck/1657613257716/healthcheck_war___/WEB-INF/web.xml].
<<__Exception__>>
...
<<__!Exception__>>
[2024.09.25 23:20:09][2] [server1-1] [SERVER-0248] The JEUS server is STANDBY.
[2024.09.25 23:20:09][0] [server1-1] [SERVER-0250] Starting server (server1) failed. Staying in
STANDBY.
[2024.09.25 23:20:09][2] [server1-1] [SERVER-0401] The elapsed time to start: 1775ms.
[2024.09.25 23:20:09][2] [launcher-22] [Launcher-0034] The server[server1] initialization completed
successfully[pid : 86118].
[2024.09.25 23:20:09][0] [launcher-1] [Launcher-0042] Completed starting the server but the server
```

```
state is still STANDBY.
JEUS_HOME/bin$
```

The **server-info** command can be used to verify that an MS is in the STANDBY state as shown in the following example.

```
[MASTER]jeus_domain.adminServer>server-info
Information about Domain (jeus_domain)
=====
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Server | Status | Node | PID | Cluster | Latest Start Time / Shutdown Time | Need to Restart | Listen Ports | Running Engines |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| adminServer (*) | RUNNING | node1 | 814 | N/A | 2024-09-25 (Thu) PM 10:34:03 KST | false | base-0.0.0.0:9736 http-server-0.0.0.0:8088 | jms, web, ejb |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| server1 | STANDBY | N/A | 861 | N/A | 2024-09-25 (Thu) PM 11:20:08 KST | false | base-0.0.0.0:9836 http-server-0.0.0.0:8188 | jms, web, ejb |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
=====
```

### 3.1.3. Shutting Down a Managed Server

An MS can be shut down using the **jeusadmin** command. It can be shut down using a local-shutdown command in jeusadmin that is connected to the MS. It can also be shut down using a MASTER command in the console tool. For more information about how to shut down an MS, refer to "Shutting Down a Managed Server (MS)" in *JEUS Domain Guide*.



Since MASTER is set to manage an MS, it is recommended to use MASTER to stop an MS.

#### Example

There is a function called graceful shutdown that guarantees the completion of the requests currently being processed before an MS shuts down. New requests are not processed. The server stops accepting requests after it receives the stop command, but the requests that are currently being processed are allowed to finish.

The following is an example of setting the wait time in the timeout option of the MS stop command.

```
[MASTER]jeus_domain.adminServer>stop-server server1 -to 60000
Stop server message to server [server1] was successfully sent.
```

It is also possible to set a wait time when the console tool is used to directly connect to an MS.

```
jeus_domain.server1>local-shutdown -to 60000
Executing this command affects the service. Do you want to continue? (y/n)y
The server [server1] has been shut down successfully.
offline>
```

After the server shuts down successfully, the following message is displayed.

```
...
[2024.09.25 23:39:28][2] [server1-25] [SERVER-0248] The JEUS server is SHUTDOWN.
[2024.09.25 23:39:28][2] [server1-24] [NET-0214] Closing http-server.
[2024.09.25 23:39:28][0] [server1-25] [SERVER-0265] The JEUS server has exited.
[2024.09.25 23:39:28][0] [server1-1] [SERVER-0099] The server[server1] has been shut down.
[2024.09.25 23:39:28][0] [server1-20] [SERVER-0565] The JVM process is shutting down.
[2024.09.25 23:39:28][0] [server1-20] [SERVER-0566] The JVM process will be terminated.
[2024.09.25 23:39:28][2] [server1-22] [NET-0214] Closing base.
```

### 3.1.4. Suspending a Managed Server

Shutting down a Managed Server is to terminate the Managed Server JVM after shutting down all running application services. A separate function is provided to suspend all applications by applying graceful timeout so that the requests currently being processed can be completed. To suspend applications, use the **suspend-server** command in the console tool.

The following is an example of suspending a server using the console tool.

```
[MASTER]jeus_domain.adminServer>suspend-server -servers server1
Executing this command affects the service. Do you want to continue? (y/n)y
Successfully suspended server(s).
```

The **server-info** command can be used to verify that the server has been suspended.

```
[MASTER]jeus_domain.adminServer>server-info
=====
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Server | Status | Node | PID | Clu | Latest Start | Need | Listen | Runni |
|         |         | Name |     | ster | Time /       | to   | Ports  | ng    |
|         |         |      |     |     | Shutdown Time| Restart|        | Engines|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| admin | RUNNING | nod | 880 | N/A | 2024-09-25   | false | base-0.0. | jms,  |
| Server | (00:22:1 | e1  | 29  |     | (Thu) PM     |      | 0.0:9736  | web,  |
| (*)   | 4)      |     |     |     | 11:38:36 KST |      | http-serv | ejb   |
|         |         |     |     |     |              |      | er-0.0.0.0|       |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```



							:8088		
serve	SUSPEND	N/A	891	N/A	2024-09-25	false	base-0.0.	jms,	
r1	ED(00:08		46		(Thu) PM		0.0:9836	web,	
	:01)				11:52:49 KST		http-serv	ejb	
							er-0.0.0.0		
							:8188		

The following message is displayed when a server has been successfully suspended.

```
[2024.09.25 00:15:16][2] [server1-25] [SERVER-0248] The JEUS server is SUSPENDING.
[2024.09.25 00:15:16][2] [server1-25] [JMS-7375] The persistence store manager for the JMS server
'server1' has been shut down.
[2024.09.25 00:15:16][2] [server1-25] [SERVER-0248] The JEUS server is SUSPENDED.
```



Even when a server is not managed by MASTER, an MS can be suspended by directly accessing the server through the console tool.

### 3.1.5. Resuming a Managed Server

It is possible to resume the applications of the suspended server.

Even when an MS is not managed by MASTER, the suspended MS can be resumed by directly accessing the server through the console tool.

The following is an example of resuming a server using the console tool.

```
[MASTER]jeus_domain.adminServer>resume-server -servers server1
Successfully resumed the servers.
```

The **server-info** command can be used to check that the server is in the RUNNING state.

```
[MASTER]jeus_domain.adminServer>server-info
```

Server	Status	Node	PID	Clu	Latest	Need	Listen	Running
		Name		ster	Start Time	to	Ports	Engines
					/ Shutdown	Restart		
					Time			
adminS	RUNNIN	nod	121	N/A	2024-09-25	false	base-0.0.	jms,
erver	G(00:04	e1	360		(Fri) AM		0.0:9736	web, ejb
(*)	:21)				11:30:11 KST		http-serv	
							er-0.0.0.0	
							:8088	

server1	RUNNING	N/A	122	N/A	2024-09-25	false	base-0.0.	jms,
	G(00:01		223		(Fri) AM		0.0:9836	web, ejb
	:07)				11:33:26 KST		http-serv	
							er-0.0.0.0	
							:8188	

The following message is displayed when the application services of the server have been resumed successfully.

```
[2024.09.25 11:34:29][2] [server1-26] [SERVER-0248] The JEUS server is RESUMING.
[2024.09.25 11:34:29][2] [server1-26] [SERVER-0248] The JEUS server is RUNNING.
```

## 3.2. Configuring Server Engine

Internally, JEUS has engines that run applications. There are separate engines for web application services, EJB application services, and JMS services. Option to use each engine can be configured during initialization, but the server must be restarted in order to apply the configuration.

### 3.2.1. Engine Option

Each engine of JEUS can be configured for use. When an engine is configured for use, it is initialized either when the server boots or when the first application that needs the engine is deployed. If an engine is not configured for use, it does not get initialized even after the server starts, and application deployment to the engine will fail.



Previously deployed applications will fail to be deployed if the server is restarted after setting the engine for non-use. The server state also changes to **STANDBY** instead of **RUNNING**. The administrator must manually change the engine and the application states.

The following example shows how to configure each engine on the server by using the console tool.

```
[MASTER]jeus_domain.adminServer>disable-engines adminServer -all
[adminServer]
Change Engine to Disabled: Web EJB JMS

Applying configuration ...
=====
+-----+-----+
|                                     Result                                     |
+-----+-----+
| Successfully changed only the JEUS Domain Configuration.                    |
| Restart the server to apply the changes.                                     |
+-----+-----+
```

```
...
[MASTER]jeus_domain.adminServer>enable-engines adminServer -all
[adminServer]
Change Engine to Enabled: Web EJB JMS
```

Applying configuration ...

Result
Successfully changed only the JEUS Domain Configuration. Restart the server to apply the changes.

```
...
[MASTER]jeus_domain.adminServer>disable-engines adminServer -web -ejb
[adminServer]
Change Engine to Disabled: Web EJB
```

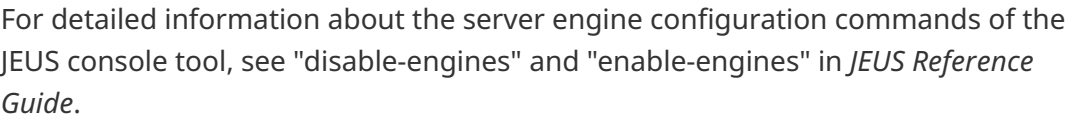
Applying configuration ...

Result
Successfully changed only the JEUS Domain Configuration. Restart the server to apply the changes.

```
...
[MASTER]jeus_domain.adminServer>enable-engines adminServer -web -ejb
[adminServer]
Change Engine to Enabled: Web EJB
```

Applying configuration ...

Result
Successfully changed only the JEUS Domain Configuration. Restart the server to apply the changes.



Each engine of JEUS can be configured for use. When an engine is configured for use, it is initialized either when the server boots or when the first application that needs the engine is deployed. If an engine is not configured for use, it does not get initialized even after the server starts, and

application deployment to the engine will fail. Option to initialize an engine can be configured by using the console tool.

The following example shows how to initialize each engine on the server by using the console tool.

```
[MASTER]jeus_domain.adminServer>disable-engine-init-on-boot adminServer,server1
EngineInitOnBoot was successfully disabled.
Applying configuration ...
```

```
=====
+-----+
|                                     |
|                                     |
+-----+
|                                     |
|                                     |
+-----+
|                                     |
|                                     |
+-----+
=====
```

```
[MASTER]jeus_domain.adminServer>enable-engine-init-on-boot adminServer
EngineInitOnBoot was successfully enabled.
Applying configuration ...
```

```
=====
+-----+
|                                     |
|                                     |
+-----+
|                                     |
|                                     |
+-----+
|                                     |
|                                     |
+-----+
=====
```



For detailed information about the server engine configuration commands of the JEUS console tool, see "disable-engine-init-on-boot" and "enable-engine-init-on-boot" in *JEUS Reference Guide*.

## 3.3. Controlling and Monitoring Threads

JEUS provides functions to monitor servlet and EJB remote threads, and system pools. JEUS also provides a function that checks thread stacks as well as sends interrupt signals to specific threads.

### 3.3.1. Thread Monitoring

JEUS provides a function to monitor threads including servlet request threads, EJB remote request threads, and system thread pools.

By using the functionality to monitor servlet threads or EJB remote request threads, it is possible to check the thread ID, thread name, thread status, and processing time. Using the thread pool monitoring function makes it possible to know the core size, max size, and keep alive time of the system thread pool as well as the information about the threads in the system thread pool.

## Thread Information Check

The following shows how to check thread information using the console tool.

```
[MASTER]jeus_domain.adminServer>thread-info -server adminServer
```

Thread information for the server [adminServer]

There are no EJB RMI threads for the server [adminServer].

There is no batch thread pool in server [adminServer].

=====  
The web engine threads for 'adminServer\_web-WebContainerThreadPool'.

tid	name	state	elapsed	uri
50	adminServer_web-WebContainerThreadPool-1	waiting	20755	
59	adminServer_web-WebContainerThreadPool-10	waiting	20745	
51	adminServer_web-WebContainerThreadPool-2	waiting	20403	
52	adminServer_web-WebContainerThreadPool-3	waiting	20395	
53	adminServer_web-WebContainerThreadPool-4	waiting	9505	
54	adminServer_web-WebContainerThreadPool-5	waiting	15399	
55	adminServer_web-WebContainerThreadPool-6	waiting	20386	
56	adminServer_web-WebContainerThreadPool-7	waiting	4507	
57	adminServer_web-WebContainerThreadPool-8	waiting	20765	
58	adminServer_web-WebContainerThreadPool-9	waiting	23346	

elapsed: Elapsed time (ms)

=====  
Thread statistics for the 'adminServer\_web-WebContainerThreadPool'.

	total	active	idle	blocked	reconn
The number of threads.	10	0	10	0	0

total = active + idle, reconn: reconnecting

=====  
The threads for the 'jeus.ejb.asyncpool' thread pool.

tid	name	thread state	active thread
(No data available)			

=====  
The statistics for the 'jeus.ejb.asyncpool' thread pool.

pool name	minimum	maximum	current	work	remaining work
	pool size	pool size	pool size	queue size	queue size

```

+-----+-----+-----+-----+-----+-----+
| jeus.ejb.a |      0 |      30 |      0 |      4096 |      4096 |
| syncpool   |      |      |      |      |      |
+-----+-----+-----+-----+-----+
=====

```

=====

The threads for the 'EJBTimerService' thread pool.

```

+-----+-----+-----+-----+-----+
| tid |      name      | thread state | active thread |
+-----+-----+-----+-----+
| 48 | EJBTimerService-1 | WAITING      | false         |
| 49 | EJBTimerService-2 | WAITING      | false         |
+-----+-----+-----+-----+
=====

```

=====

The statistics for the 'EJBTimerService' thread pool.

```

+-----+-----+-----+-----+-----+-----+
| pool name | minimum | maximum | current | work | remaining work |
|           | pool size | pool size | pool size | queue size | queue size      |
+-----+-----+-----+-----+-----+-----+
| EJBTimer |      2 |      30 |      2 | 4096 |      4096 |
| Service  |      |      |      |      |      |
+-----+-----+-----+-----+-----+
=====

```

=====

The threads for the 'threadpool.System' thread pool.

```

+-----+-----+-----+-----+-----+
| tid |      name      | thread state | active thread |
+-----+-----+-----+-----+
| 72 | threadpool.System-1 | TIMED_WAITING | false         |
| 73 | threadpool.System-2 | TIMED_WAITING | false         |
| 83 | threadpool.System-3 | TIMED_WAITING | false         |
+-----+-----+-----+-----+
=====

```

=====

The statistics for the 'threadpool.System' thread pool.

```

+-----+-----+-----+-----+-----+-----+
| pool name | minimum | maximum | current | work | remaining work |
|           | pool size | pool size | pool size | queue size | queue size      |
+-----+-----+-----+-----+-----+-----+
| threadpool |      0 |      100 |      3 | 4096 |      4096 |
| l.System   |      |      |      |      |      |
+-----+-----+-----+-----+-----+
=====

```

=====

The threads for the 'LPQ-INTERNAL' thread pool.

```

+-----+-----+-----+-----+
| tid | name | thread state | active thread |
+-----+-----+-----+-----+

```

(No data available)

The statistics for the 'LPQ-INTERNAL' thread pool.

pool name	minimum pool size	maximum pool size	current pool size	work queue size	remaining work queue size
LPQ-IN TERNAL	0	50	0	4096	4096

The threads for the 'JMS-INTERNAL' thread pool.

tid	name	thread state	active thread
38	JMS-INTERNAL-3	WAITING	false
41	JMS-INTERNAL-6	WAITING	false
42	JMS-INTERNAL-7	WAITING	false
43	JMS-INTERNAL-8	WAITING	false
45	JMS-INTERNAL-10	WAITING	false
40	JMS-INTERNAL-5	WAITING	false
44	JMS-INTERNAL-9	WAITING	false
37	JMS-INTERNAL-2	WAITING	false
39	JMS-INTERNAL-4	WAITING	false
36	JMS-INTERNAL-1	WAITING	false

The statistics for the 'JMS-INTERNAL' thread pool.

pool name	minimum pool size	maximum pool size	current pool size	work queue size	remaining work queue size
JMS-IN TERNAL	10	20	10	4096	4096

## Checking the Stack Trace of a Thread

The stack trace of a desired thread can also be checked in the console tool. The thread stack trace is usually checked when a blocked thread is detected by using the ti command of the console tool.

```
[MASTER]jeus_domain.adminServer>print-stack-trace -server server1 45
servlet thread [tid=45] Stack trace of server1_web-WebContainerThreadPool-1 tid=45
java.lang.Thread.State: WAITING
    at sun.misc.Unsafe.park(Native Method)
    at java.util.concurrent.locks.LockSupport.park(LockSupport.java:175)
```

```
at
java.util.concurrent.locks.AbstractQueuedSynchronizer$ConditionObject.await(AbstractQueuedSynchronizer.java:2039)
at jeus.util.pool.auto.LinkedBlockingQueue.take(LinkedBlockingQueue.java:450)
at jeus.util.pool.ThreadPoolExecutor.getTask(ThreadPoolExecutor.java:1418)
at jeus.util.pool.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:1340)
at jeus.servlet.thread.WebThreadPoolExecutor$WebRequestWorker.run(WebThreadPoolExecutor.java:273)
at java.lang.Thread.run(Thread.java:748)
```



For detailed information about the **ti** and **strace** commands of the JEUS console tool, see "Local Commands" in *JEUS Reference Guide*.

### 3.3.2. Thread Control

JEUS provides a function that sends an InterruptSignal to a specific thread to trigger an exception to stop the thread execution.



This function is provided experimentally. If a thread is interrupted, the thread will not be stopped right away. Instead, it checks the interrupt status and throws an exception to stop any further processing. The exception must be handled in the user application, and the user assumes responsibility for issues that occur from unhandled exceptions.

#### Thread Interrupt

Interrupting a thread means sending a signal to induce an exception to stop further processing of the thread. It is usually used when a request is blocked or delayed longer than expected.

When a thread is interrupted, the thread is neither forcibly stopped nor stopped immediately.

JEUS supports a function that can send the InterruptSignal to a servlet thread, EJB remote request thread, and running thread in the system thread pool. A thread can be interrupted if it is processing a JNDI remote connection, EJB, JDBC, or IO operation.

- **Servlet Thread**

An exception is thrown when a servlet thread is interrupted in the following cases. The exception must be handled in the user application.

- When connecting to a remote JNDI
- When processing a JNDI remote operation
- When calling an EJB
- When getting a JDBC connection or calling a method of the connected object



An interrupt signal can be issued directly through the console tool, or through a function that checks the processing time of a servlet thread. To configure this function, refer to "Configuring Automatic Thread Pool Management (Thread Status Notification)" in *JEUS Web Engine Guide*.

#### • EJB Remote Request Thread

An EJB issues a `jakarta.ejb.EJBException` by checking the interrupt status in the following cases.

- When an EJBHome method like 'create' is called in the EJB 2.x way (in EJB 3.0, create is processed internally in the EJB container when the user looks up an EJB).
- When calling an EJB business method

The exception must be handled in the user application. If the interrupt signal is issued after the EJB business method is called, an exception may not occur. However, the thread may be interrupted when the method calls another EJB or uses a JDBC or JNDI operation.

#### • JDBC

For JDBC operations, if an interrupt signal is received while executing the following operations, a `java.sql.SQLException` will occur. The user application must handle the exception.

- When getting a connection from the connection pool via `DataSource#getConnection()`.
- When using the `java.sql.Connection` from the connection pool.
- When using the `executeQuery` operation after setting the `use-sql-trace` or `stmt-caching-size` option for the JDBC connection pool. (For more information about the options, refer to [Connection Pool Configuration](#))

#### • JNDI

A JNDI operation can sometimes be interrupted. If the interrupt signal is issued while executing the following JNDI operations, a `javax.naming.NamingException` will occur. The user application must handle the exception.

- When connecting to a remote JNDI (when a `javax.naming.InitialContext` is created).
- When a remote message is sent while executing a JNDI operation.

The operation can be interrupted during JNDI operations such as looking up, binding, rebinding, and renaming, but not when accessing a local registry or cache.

#### • IO Operation

The IO operation can sometimes be interrupted depending on the situation.

- `java.nio.channels.SocketChannel`

An operation can be interrupted while using the `java.nio.channels.SocketChannel`. Connecting via the `java.nio.channels.SocketChannel` is performed in three steps.

At the beginning step, check whether the interrupt status of the thread is set, and close the

channel if it is set. During the actual connection step, if the Interrupt Status is set, the connect() method is not called. During the end step, a java.nio.channels.ClosedByInterruptedException occurs if the Interrupt Status is set (Interrupt Status gets cleared afterwards).

- java.nio.channels.SocketChannel#read()/java.nio.channels.SocketChannel#write()

The java.nio.channels.SocketChannel#read() or java.nio.channels.SocketChannel#write() is performed in three steps.

At the beginning step, check whether the interrupt status of the thread is set, and close the channel if it is set. In the second step, if the interrupt status is set, read or write operation is not executed. During the end step, a java.nio.channels.ClosedByInterruptedException occurs if the Interrupt Status is set (Interrupt Status gets cleared afterwards).

After the java.nio.channels.ClosedByInterruptedException has been thrown, if a read or write is attempted by using the previously created channel, a java.nio.channels.ClosedChannelException is thrown.

- Using java.net.Socket

When using the java.net.Socket, the Socket IO operation can be interrupted depending on the JVM compile options in HP-UX and Solaris.

A read or write operation that uses a socket stream can be interrupted and a java.net.SocketException is thrown. (Interrupt Status gets cleared afterwards). This does not immediately close the socket, but it stops any subsequent operations.

A thread that is executing a socket IO operation is not interrupted in IBM AIX and LINUX.

- **Object#wait() / Thread#sleep() / Thread#join()**

When a thread that is executing Object#wait(), Thread#sleep(), or Thread#join(), the thread can be interrupted by throwing a java.lang.InterruptedException. (Interrupt Status gets cleared afterwards).



Refer to the Javadoc API of each class.

## **Sending the Interrupt Signal**

JEUS provides a command to send an interrupt signal to a thread by using the console tool.

The following is an example of sending a thread interrupt signal by using the console tool. For more information about the **interrupt-thread** command in the JEUS console tool, refer to "interrupt-thread" in *JEUS Reference Guide*.

```
[MASTER]jeus_domain.adminServer>interrupt-thread -server server1 45
Sent an interrupt hint signal to the thread [tid=45] on the server server1.
```

## 3.4. Controlling and Monitoring Memory

JEUS monitors the memory usage and controls the server when it exceeds the specified free memory.

### 3.4.1. Memory Monitoring

The **memory-info** command can be used to monitor memory. For more information about the memory-info command, see "Server Management Commands" in *JEUS Reference Guide*.

```
[MASTER]jeus_domain.adminServer>memory-info -servers adminServer
Domain [jeus_domain] Memory Information
Memory Information
=====
+-----+-----+-----+
|  Server  | Total Amount of Memory | The Current Amount of Memory |
+-----+-----+-----+
| adminServer |          721420288 |          315030816 |
+-----+-----+-----+
=====
```

### 3.4.2. Memory Usage Control

Since this function is not commonly used in JEUS, memory related operations can only be controlled through the current system properties.

Property	Description
jeus.server.memorymonitor.enabled	Configures whether to use MemoryMonitorService. If the configuration is not enabled, then the rest of the properties do not have any meaning.  (Default value: false)
jeus.server.memorymonitor.ratio	Configures the memory overflow standard.  If 0.8 is configured, then if 80% of memory is used (based on the max memory), then it shall be considered as overflow. (Default value: 0.8)
jeus.server.memorymonitor.interval	Configures the cycle which measures memory usage.  (Default Value: 2000, unit: milliseconds)
jeus.server.memorymonitor.duration	Configures the time which determines memory overflow. For example, if jeus.server.memorymonitor.ratio is 0.8 and jeus.server.memorymonitor.duration is 1 minute, then in MemoryMonitorService, when over 80% of memory is continued for over 1 minute, then it is considered as memory overflow. (Default value: 60000, Unit: milliseconds)
jeus.server.enable.restart.in.memory.shortage	When determined as memory overflow, then determines whether the server will be automatically restarted. (Default value: true).



1. When a memory overflow occurs, a heap dump is executed only once (created in the 'SERVER\_HOME/logs' directory).
2. For IBM JDK, a heap dump is created in the directory where Java is running. In order to configure a different path, use the '-Xdump:heap:file' option. For information about other IBM heap dump options, refer to the relevant website.
3. A thread dump is written in log before a heap dump is created.

## 4. JNDI Naming Server

This chapter describes basic concepts and terminologies of JEUS JNDI. It also describes how to set the environment configuration and develop applications.

### 4.1. Overview

Java Naming and Directory Interface™ (JNDI) is a standard API used by Java applications to find and get an object in the network by using the logical name of the object. From the user perspective, it provides an environment where applications can find and use objects more easily than in the previous enterprise environment.

JEUS JNDI is compatible with JNDI 1.2 API and supports Sun Microsystems' JNDI API standards. It also provides JNDI Service Provider Interface (SPI) for compatibility with the enterprise environment. This means that any product that implements JNDI SPI can use the objects of JEUS JNDI Tree.

The JEUS JNDI service is used throughout the JEUS system and can be seen whenever the EJB, servlet/JSP, JMS, and JDBC are used.

### 4.2. Basic Concepts and Structure

JEUS JNDI has a unique architecture that binds and looks up objects. This section will first describe the basic concepts and structure of JEUS JNDI.

#### 4.2.1. Basic Concepts

When a managed server (MS) starts, JEUS automatically prepares JNDI Service that is provided by the JNDI naming server. When the service starts, the JEUS Naming Service Server (JNSServer) starts internally. JEUS Naming Service Client (JNSClient) acts as a client that communicates with the JNSServer.

When a JNSClient is connected to a JNSServer, objects are first bound and looked up in the JNSClient and then in the JNSServer. A JNSServer is connected to more than one JNSClient. A JNSServer is also connected to other JNSServers (especially in the clustering environment) in a tree-like structure. This naming repository structure is called the **JNDI Tree**.

The JNDI Tree is used to bind or look up objects. All objects are looked up and bound to the JNDI Tree through the server. When an application requests to bind an object to a JNSClient, the object is sent and bound to the JNDI Tree. A JNSClient in the application can look up the bound objects.

The objects in the JNDI tree can only be accessed by using **InitialContext**. Therefore, an application must create the InitialContext object to use JNDI. The InitialContext object provides access to other objects through the JNDI Tree. In addition, it binds or looks up objects as well as gets and removes the object list.

Bound objects can be checked by using the console tool.

### 4.2.2. Checking Bound Objects

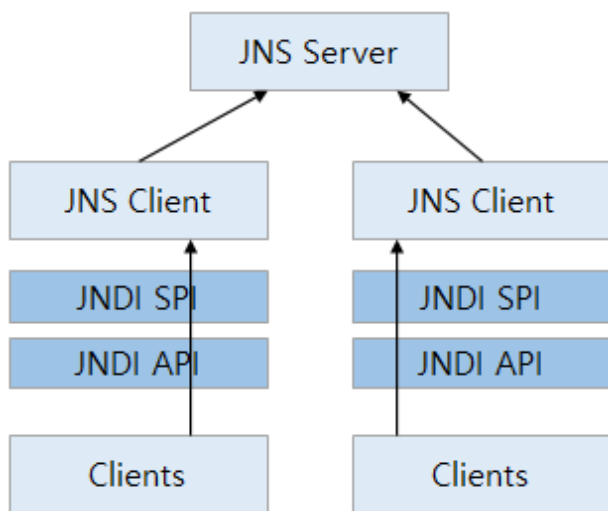
The bound objects can be checked by using the console tool. For information on how to use the console tool to check bound objects, refer to "jndi-info" in *JEUS Reference Guide*.

### 4.2.3. JNDI Naming Server Architecture

This section describes the operation of the JNDI tree structure.

The JNDI Tree consists of JNSServers and JNSClients. A JNSServer is on the JVM of the MS, and a JNSClient is on the JVM of an MS or a client. As the main element of the JEUS JNDI architecture, the JNSServer creates and manages the JNDI Tree. A JNSClient resides below the JNSServer and is managed by the JNSServer.

The following shows the relationship between a JNSServer and JNSClients.



Relationship Between JNSServer and JNSClient

To access the entire JNDI Tree, a JNSClient must send a request to the JNSServer.

A JNSServer is connected to other JNSServers on other MSs to form a cluster. A JNSClient interacts with a JNSServer on an MS and handles client access requests. The client does not directly access the JNSServer, but instead binds and looks up objects through a JNSClient.

#### JNSServer

A JNSServer manages the JNDI Tree. It is an independent naming server that allows a JNSClient to access the JNDI Tree. Multiple JNSServers can be connected to expand the JNDI Tree. This is because a JNSServer can be directly connected to other JNSServers on other MSs. In JEUS, after an MS starts, the JNSServer automatically waits for JNSClient access requests.

#### JNSClient

The basic function of a JNSClient is to send application requests to a JNSServer and return the result from the JNSServer. On each JVM, there is one JNSClient singleton instance per server. Since only one JNSClient is used to lookup an object, it can be used effectively with an EJB or a servlet in the enterprise environment.

The following is a list of important JNSClient features.

- Accessing a JNDI Tree

Provides a way to access the JNDI Tree that is managed by the MS accessing the JNSServer. The objects that are bound and looked up can be shared by the entire JNDI Tree or only by specific clients depending on the client configuration.

- Caching looked up objects

A JNSClient caches frequently used objects for faster access. A JNSClient caches the objects while communicating with a JNSServer.

- Managing JNSServer connections

A JNSClient receives a request from a client and sends it to JNSServer and returns the result from the server. Since a JNSClient is on the JVM where the client exists, communication can be made efficiently without incurring additional IOs.

The following table describes the two types of JNS clients distinguished by where the JNSServer is.

JNSClient	Description
Server-side	<p>Used by EJB Beans, servlets, and JSPs on each engine of the MS.</p> <p>To use a server-side JNSClient, configure the <code>java.naming.factory.initial</code> property as <code>jeus.jndi.JNSContextFactory</code>. This value is usually configured when the MS starts, and it does not need to be configured separately.</p>
Client-side	<p>Used by applets, applications, and clients that are not running on the MS.</p> <p>To use a client-side JNSClient, configure the <code>java.naming.factory.initial</code> property as <code>jeus.jndi.JNSContextFactory</code>.</p> <p>A client-side JNSClient manages resources efficiently by closing any connections that are idle for a specified time period and reconnecting if necessary. The applications running in the client container do not need to configure this value since it is set as a system property when the client container starts.</p>

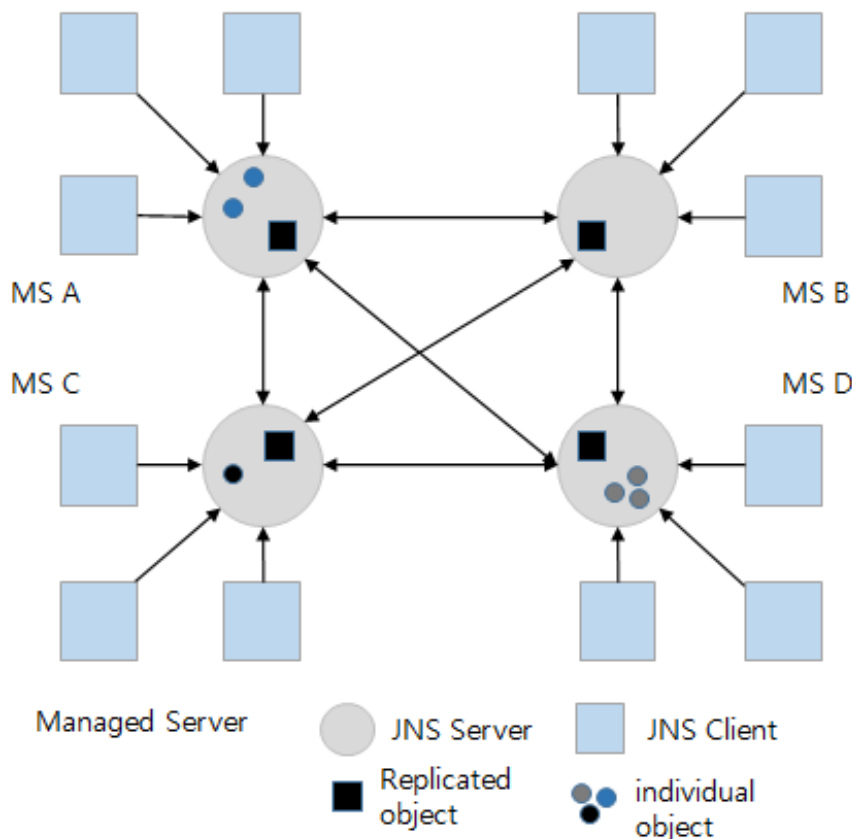
## 4.2.4. JNDI Clustering

The JNDI tree consists of connections between a JNSServer and JNSClients. This structure allows a JNSClient to let other JNSClients know about modified objects and supports multiple MSs (clustering). In other words, individual JNDIs form an extensive JNDI tree through clustering. When there are any modified objects in a JNDI Tree, the changes are propagated to other JNDIs. Therefore, in the JNDI

Naming clustered environment, objects bound to a naming server can be looked up from other naming servers.

For example, as shown in the following figure, a cluster environment is composed of four types of MSs (A,B,C,D). If an object called obj1 on MS A is bound as objName1, then this object is sent to all the other MSs (B,C,D). As a result, objName1 can also be looked up on MS B, C, and D to obtain obj1.

Not all bound objects can be replicated to other MSs in the cluster. Only the objects bound to the cluster can be replicated. For example, objects like data sources that are configured to a specific MS can only be bound to the JNSServer of each MS.



JEUS JNDI Architecture in Clustered Environment

Each MS is responsible for managing each JNSServer. Each JNSServer starts when JEUS system starts and is connected to JNSServers on other MSs. When each JEUS engine gets an InitialContext object, a JNSClient connects to the JNSServer. When a client looks up a JNDI tree object, the request is sent to the JNSClient and then to the JNSServer on the MS. As a result, the client obtains the requested object.

## Remote Lookup in a Clustered Environment

If no separate settings are made, JNDI Lookup is performed for the JEUS JNDI clustering area in which the application is included. However, when the application looks up the contents of another MS's JNDI server that is not in the same cluster, the provider URL (see "JNDI System Properties" in JEUS Reference Guide) must be specified to create a context, or a name using the `jh` (JEUS Host) protocol created by JEUS must be used to perform a lookup. For more information, refer to [JNDI Programming](#).



## JNSServer Replication

Each JNSServer of JEUS is connected to other servers and waiting to interact with them. When a new MS joins an existing cluster, its JNSServer sends a notification to other connected JNSServers. Each JNSServer also sends its data to the new JNS server so that the existing bound objects can be looked up on the new JNS server.

Owing to such scalability, a JNS server that is restarted due to an error can continue to operate normally by receiving the JNDI Tree information from the other JNSServers.

## 4.3. Configuring JNDI Naming Server

A JNDI naming server consists of a JNSServer and a JNSClient, each with different configurations.

A JNSServer needs configurations to accept JNSClient connection requests and access other JNSServers. A JNSClient needs configurations to access the JNSServer and modify the JNDI tree.

This section describes how to configure the JNSServer and JNSClient.

### 4.3.1. Configuring JNSServer

#### Configuring a Common Thread Pool

A thread pool to share server-wide can be configured using the console tool. For basic information about thread pools, refer to [Thread Pool Configuration](#).

- **Using the console tool**

A common thread pool can be configured using the console tool.

```
[MASTER]domain1.adminServer>modify-system-thread-pool adminServer -service namingserver -r 10
Successfully performed the MODIFY operation for The namingserver thread pool of
the server (adminServer)., but all changes were non-dynamic. They will be applie
d after restarting.
Check the results using "show-system-thread-pool adminServer -service namingserv
er or modify-system-thread-pool adminServer -service namingserver"

[MASTER]domain1.adminServer>modify-system-thread-pool adminServer -service namingserver
show the current configuration.
The namingserver thread pool of the server (adminServer)
=====
+-----+-----+-----+
| Reserved Threads for the Service namingserver                | 10  |
+-----+-----+-----+
=====
```

#### Configuring service thread pool

Configure the service thread pool by using the console tool.

## • Using the console tool

The service thread pool can also be configured using the console tool.

```
[MASTER]domain1.adminServer>modify-service-thread-pool server1 -service namingserver -min 0 -max 20
```

Successfully performed the MODIFY operation for The namingserver thread pool of the server (server1)., but all changes were non-dynamic. They will be applied after restarting.

Check the results using "show-service-thread-pool server1 -service namingserver or modify-service-thread-pool server1 -service namingserver"

```
[MASTER]domain1.adminServer>modify-service-thread-pool server1 -service namingserver
```

Shows the current configuration.

The namingserver thread pool of the server (server1).

```
=====
+-----+-----+
| Min                | 0      |
| Max                | 20     |
| Keep-Alive Time    | 60000  |
| Queue Size         | 4096   |
| Max Stuck Thread Time | 3600000 |
| Action On Stuck Thread | NONE   |
| Stuck Thread Check Period | 300000 |
+-----+-----+
=====
```

The following describes the configuration items related to **stuck thread handling**.

Item	Description
Max Stuck Thread Time	Used to detect a stuck thread.  If a thread remains occupied for longer than this time, the thread is considered as stuck. [Default value: 3600000 (1 hour), Unit: milliseconds]
Action On Stuck Thread	Action to perform when a thread is identified as stuck.  Select one of the following options. <ul style="list-style-type: none"><li>• None: No action is taken.</li><li>• Interrupt: Sends an interrupt signal.</li><li>• IgnoreAndReplace: Replaces the stuck thread with a new thread.</li></ul>
Stuck Thread Check Period	Interval for checking the status of a stuck thread.  At this interval, thread status is checked to see if the thread is stuck. (Default value: 300000 (5 minutes), Unit: milliseconds)



Since the thread pool configuration can be changed dynamically, the server does not need to be restarted. However, the server must be restarted in order to change from a common thread pool to a dedicated thread pool.

## 4.3.2. Configuring JNSClient

The configuration of the JNSClient depends on where JNSClient is running.

- Server-side JNSClient configuration

A server-side JNSClient executes on an MS and follows the JNS server configuration.

- Client-side JNSClient Configuration

A client-side JNSClient can access JNS servers on other JVMs. It is connected to a JNSServer and there is a thread that applies the JNDI tree information. The JEUS JNDI uses a thread pool to manage the threads. The thread pool is configured as a JEUS property, but the default value can be used.

The following are the properties of a client-side JNSClient.

- JNSContext.RESOLUTION
- JNSContext.CONNECT\_TIMEOUT
- JNSContext.CONNECTION\_DURATION



If JNDI is used in server-side objects like an EJB, servlet, or JSP, a server-side JNSClient is used to bind or lookup the objects. Therefore, it is not necessary to configure these properties. For more information about the properties, refer to "JNDI System Properties" in *JEUS Reference Guide*.

## 4.4. JNDI in Clustered Environment

When a clustered environment is configured for the MSs, each JNSServer becomes automatically clustered. For more information about how to cluster MSs, refer to "JEUS Clustering" in *JEUS Domain Guide*.

By default, a JNSClient connects to a local JNSServer. The JNSClient recognizes the address of the MS's JNSServer as Context.PROVIDER\_URL. Each MS address in the cluster must be added to Context.PROVIDER\_URL as in the following.

JNSClients perform load balancing and failover by using the provided Context.PROVIDER\_URL value.

```
Hashtable ht = new Hashtable();  
ht.put(Context.PROVIDER_URL, "host1:9736,host2:9736"); //host1 and host2 are clustered
```

When an MS that is assigned to Context.PROVIDER\_URL fails, the JNSClient detects the failure when it tries to execute the JNSServer's JNDI operation on the failed MS and performs failover to another JNSServer. The JNSClient regularly checks the failed MS to see if it is in the RUNNING state. This configuration can be set by using the '-Djeus.jndi.cluster.recheckto' option (Default value: 5, Unit: minutes).

If a JNSClient runs on an MS managed by the Master Server (MASTER), use the cluster name that was configured on MASTER as in the following.

```
Hashtable ht = new Hashtable();  
ht.put(Context.PROVIDER_URL, "jeus://cluster1"); //cluster1 is the name of the cluster configured in  
the MASTER
```

By using the cluster name, the latest status information of the MS can always be obtained from MASTER to configure the cluster regardless of the "jeus.jndi.cluster.recheckto" setting on the JNSClient. In other words, if an MS in the cluster is in a FAILED state or a new MS is added, the server is updated before performing JNDI operation on the JNSClient. **Therefore, it is recommended to use the cluster name when the JNSClient is running on an MS that is managed by MASTER.**

## Lookup

Any client in a cluster can look up any object that is bound to a JNDI Tree. When a client has an object bound to a JNSClient, the object is shared by all the MSs that are clustered through a JNSServer. Any data changes or deletions are also applied to the JNSClient of each MS.

Depending on the characteristics of each object, some objects are shared across the entire cluster (objects such as export name for lookup) while some objects are only visible on its own MS (MS-dependent objects such as data sources).

## 4.5. JNDI Programming

This section describes how to use JEUS JNDI for programming.

Java clients can access a JNDI Tree by using the InitialContext object. The InitialContext object uses the Standard JNDI standard properties and JEUS properties. First, configure the JNDI environment, and then look up objects by using the InitialContext. Next, get the object reference. Finally, close the InitialContext after use.

The following are the steps used by a Java client to use the JEUS JNDI.

1. Configure the JEUS environment.
2. Configure properties for the InitialContext.
3. Use the context to look up the named object.
4. Use the named object to get the object reference.
5. Close the context.

The following is a detailed description of each step.

### 4.5.1. Configuring the JEUS environment

Set the path for JEUS Client modules, JEUS\_CLIENT, in the classpath so that JNDI services can use the

classes.

```
-classpath ${JEUS_CLIENT};
```

## 4.5.2. Configuring properties for InitialContext

Java clients configure environment properties for the InitialContext before using JEUS JNDI services.

There are two configuration methods.

- Using the '-D' option of the JVM.
- Creating a hash table and sending it to the InitialContext constructor.

Using the '-D' option of the JVM is more preferable than creating a hash table.

The following are the properties that must be configured to create the InitialContext of JEUS JNDI.

- Context.INITIAL\_CONTEXT\_FACTORY (required)
- Context.URL\_PKG\_PREFIXES
- Context.PROVIDER\_URL
- Context.SECURITY\_PRINCIPAL
- Context.SECURITY\_CREDENTIALS



For more information about properties, refer to "JNDI System Properties" in *JEUS Reference Guide*.

Add these properties to the hash table and use them to create an InitialContext. Use the default InitialContext if the InitialContext is only used in the objects on the server (EJBs, servlets or JSPs).

The following are the client program configurations.

```
Context ctx = null;
Hashtable ht = new Hashtable();
ht.put(Context.INITIAL_CONTEXT_FACTORY, "jeus.jndi.JNSContextFactory");
ht.put(Context.URL_PKG_PREFIXES, "jeus.jndi.jns.url");
ht.put(Context.PROVIDER_URL, "<hostname>");
ht.put(Context.SECURITY_PRINCIPAL, "<username>");
ht.put(Context.SECURITY_CREDENTIALS, "<password>");

try {
    ctx = new InitialContext(ht);
    // use the context
} catch (NamingException ne) {
    // fail to get an InitialContext
} finally {
    try{
```

```

        ctx.close();
    catch (Exception e) {
        // an error occurred
    }
}

```

When JNDI is used in a cluster environment, additional environment properties for `jeus.jndi.JNSContext` can be used to create a JNDI Tree and effectively manage the internal operations of a `JNSClient`.

- Creating a context using the hash table

To configure an environment property of the `InitialContext`, the property name is assigned to a hash table (`java.util.Hashtable`) key and the property data is assigned to the hash table (`java.util.Hashtable`) value. The key value pair is passed to the `InitialContext` constructor as a parameter.

- Creating a Security Context

Multiple security-related environment properties can be used to allow a user of the JEUS security domain to access an execution thread. When the `InitialContext` is created, the following properties are configured for the Security Context.

- User name property: `java.naming.security.principal`
- Password property: `java.naming.security.credentials`

After a successful authentication, the authenticated user information is configured for an execution thread, and the user can access the resources that are managed by the JEUS Security Manager. If the authentication fails, the thread recognizes the user as a guest user.

Even when the Security Context is configured, the context is ignored when the JEUS security API is used to log on before the `InitialContext` is created. In this case, the already logged on subject is used for JNDI communication.



1. For more information about user information settings and JEUS Security Service, refer to *JEUS Security Guide*.
2. For more information on `InitialContext` related properties and environment properties, refer to "JNDI System Properties" in *JEUS Reference Guide*.

### 4.5.3. Lookup for Named Objects Using Context

Objects that are bound to a JNDI Tree can be located by using the `lookup()` method of the JNDI context. In the following example, 'countermod' is the module name of the object that is going to be looked up, and the 'Count' is the EJB name.

```

try {
    Context ctx = new InitialContext();

```

```

    Count count = (Count)ctx.lookup("java:global/countermod/Count");
    //    Count count = (Count)ctx.lookup("java:global/countermod/Count!my.ejb3.tx.Count");
    //    Count count = (Count)ctx.lookup("Count");
    // successfully got the object
} catch (NameNotFoundException ex) {
    // no such binding exists
} catch (NamingException e) {
    // an error occurred
}

```

When there are multiple business interfaces, append '[interface name]' to the name to look up.

```

Count count = (Count)ctx.lookup("java:global/countermod/Count!my.ejb3.tx.Count");

```

When there is an export name, use the export name to look up.

```

Count count = (Count)ctx.lookup("Count");

```

The following is an example of looking up an EJB 2.x object.

```

try {
    Context ctx = new InitialContext();
    CountHome countHome = (CountHome)ctx.lookup("Count");
    // successfully got the object
} catch (NameNotFoundException ex) {
    // no such binding exists
} catch (NamingException e) {
    // an error occurred
}

```

#### 4.5.4. Using a Named Object

In EJB 3.x, a looked-up object can be used immediately as in the following.

```

count.service();

```

For EJB 2.x, get the reference of the EJB remote object by using the create() method of the looked-up EJB home object. Execute the create() method to get the home reference of the object and use it to execute a method as in the following.

```

Count count = countHome.create();
count.service();

```

## 4.5.5. Closing the Context

When finished with using the context, close the context by executing the `close()` method as in the following.

```
try {
    cx.close();
} catch(Exception e) {
    // an error occurred
}
```

## 4.5.6. Creating a Clustered Context

To create a context that is usable in the clustered environment in JEUS, set multiple hosts in the `Context.PROVIDER_URL` property as in the following.

```
Hashtable ht = new Hashtable();
ht.put(Context.PROVIDER_URL, "host1:9736,host2:9736");
```

If the cluster name that is configured in the domain is known, then it can be configured for use as in the following.

```
'jeus://' + <cluster name> in Context.PROVIDER_URL
```

However, it is usable only on an MS in the cluster and cannot be used on a standalone client where the cluster information is unknown.

```
Hashtable ht = new Hashtable();
ht.put(Context.PROVIDER_URL, "jeus://cluster1");
```

In a context clustered environment, a policy can be configured for selecting an MS to look up.

`Jeus.jndi.clusterlink.selection-policy` property can be used to configure one of the following three available options. As well, these values can be configured with the `System` property, and in this case, configuration via `Hashtable` is given priority.

Division	Description
locallinkPreference	Use an object on the local MS.
roundrobin	For the first request, randomly select an MS to look up the object. Use the round robin method to select an MS for any subsequent requests.
random	Randomly select one of the clustered MSs.



The following is a configuration example.

```
Hashtable ht = new Hashtable();
ht.put(Context.PROVIDER_URL, "host1:9736,host2:9736");
ht.put("jeus.jndi.clusterlink.selection-policy", "random");
```

### 4.5.7. Remote Lookup

When an application looks up from a JNDI server on an MS that is not in the cluster, it must create a context by specifying the PROVIDER URL or by using the name that uses the jh (JEUS host) protocol created by JEUS.

The following syntax can be used to look up an object on a remote JEUS JNDI cluster from a clustered environment in JEUS.

```
jh:<remote JEUS host name>:<remote JEUS base port>/<export name>
("jh" = JEUS host)
```

The following shows how to implement a lookup. Add the following code to the JNDI context string.

```
try {
    Context ctx = new InitialContext();
    CountHome countHome = (CountHome)ctx.lookup("jh:dev:9736/Count");
    // successfully got the object
} catch (NameNotFoundException ex) {
    // no such binding exists
} catch (NamingException e) {
    // an error occurred
}
```

# 5. External Resource

This chapter describes a variety of external resources that can be used to build a system by integrating with JEUS, and how to configure them. Refer to each external resource guide for more information about configuring external resources and their usage.

## 5.1. Resource Types

External resources exist outside JEUS, and applications can access them through JEUS. A typical example is a database. Resources like these can be accessed by adding related configurations to JEUS.



If an external resource provides a resource adapter with JCA standard compatibility, it is recommended to deploy and use the provided resource adapter.

The following are the resources that can be configured in JEUS.

- **Data Source**

A client can directly access a data source without adding any configurations to JEUS. However, configuring a data source enables applications to more easily access databases by using JDBC connection pool through JNDI. Refer to [DB Connection Pool and JDBC](#) for more information about data source configurations.

- **Mail Resource**

A mail resource is used by a client application to send an e-mail by using a mail protocol like SMTP. In JEUS, a mail resource binds the e-mail host information to the JNDI export name and allows the client to indirectly gain access to the host. A mail resource of the jakarta.mail.Session type can be obtained through JNDI lookup.

- **URL Resource**

A URL resource allows applications to access the external URL objects through JNDI. If a URL changes, the URL configuration is modified and the applications can continue to use the resource without source modification.

A URL object of the java.net.URL type can be obtained through JNDI lookup.

- **Message Bridge**

A bridge for connection between destinations of different JMS vendors. An MQ that complies with the Jakarta Messaging 2.0 specification can be configured without any limitations. For more information, refer to "JEUS MQ Message Bridge" in JEUS MQ Guide.

- **Custom Resource**

A custom resource allows a Java bean resource to be bound to the JNDI repository. It is a typical resource that performs a lookup by using a service that is registered through the JNDI ObjectFactory.

- **External Resources - IBM MQ and Sonic MQ**

Denormalized resources that can be connected to JEUS. Usually, Jakarta Messaging products of IBM MQ or Sonic MQ and Tmax TP Monitor can be configured in JEUS. These resources can also be accessed directly via Java APIs without being configured in JEUS. However, to manage these resources from the transaction manager, they need to be configured in JEUS. For more information, refer to [Transaction Manager](#).

- **External Resource**

Resources running on JEUS. They are usually used by WebT, jTmax, or InfiniteCache that are integrated with JEUS. A class that implements the `jeus.external.ResourceBootstrapper` interface must be used to access the external resources in JEUS.

- **Concurrency Utilities Resource**

Defines resources related to Jakarta Concurrency. This allows you to define manageable tasks on the application server and maintain context and operate when the tasks are executed. For more information, see "JEUS Jakarta Concurrency Guide".

## 5.2. Resource Configuration

This section describes how to configure each resource.

Resources are configured within the scope of the domain and registered on a server when the server starts.

### 5.2.1. Configuring a Data Source

Data sources handle configurations that are related to the database. For more information, refer to [DB Connection Pool and JDBC](#).

### 5.2.2. Configuring a Mail Source

A mail source can be used to configure an SMTP host that is used by a client application to send an e-mail to JEUS. Refer to the Jakarta Mail specification for more information about e-mail hosts.

You can set the mail source by editing `domain.xml` as follows:

```
<servers>
  <server>
    <name>server1</name>
```

```

...
<resources>
  <mail-source>
    <mail-entry>
      <export-name>mailSource</export-name>
      <mail-property>
        <name>mail.smtp.auth</name>
        <value>true</value>
      </mail-property>
    </mail-entry>
  </mail-source>
</resource>
</server>
</servers>

```



Since the property names in the '**Mail Property**' section follow the Jakarta Mail specification, refer to the corresponding specification for each property configuration.

### 5.2.3. Configuring a URL Source

The following values are bound to the JNDI names, PRIMARY\_URL and SECONDARY\_URL respectively.

```
http://www.foo.com
```

```
http://www.bar.com
```

Edit domain1.xml as follows to set the URL source.

```

<resources>
  <url-source>
    <url-entry>
      <export-name>PRIMARY_URL</export-name>
      <url>http://www.foo.com</url>
    </url-entry>
    <url-entry>
      <export-name>SECONDARY_URL</export-name>
      <url>http://www.bar.com</url>
    </url-entry>
  </url-source>
</resources>

```

### 5.2.4. Configuring a Custom Resource

Custom resources enable the lookup and use of Java bean resources through JNDI ObjectFactory.

This section describes how to configure and register custom resources.

The following is an example of an object factory class that creates a Java bean resource class and resource instance. This class must exist in the 'SERVER\_HOME/lib/application' or 'DOMAIN\_HOME/lib/application' directory.

#### Factory Class Example that Creates Custom Resources

```
package dog;

import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.Name;
import javax.naming.spi.ObjectFactory;

public class DogFactory implements ObjectFactory {
    public DogFactory() {}

    public Object getObjectInstance(Object obj, Name name, Context nameCtx, Hashtable<?, ?>
environment)
        throws Exception {
        Dog dog = Dog.getInstance();
        System.out.println("Creating a dog whose name is " + dog.getName() + ",
and age is " + dog.getAge());
        return dog;
    }
}
```

#### Custom Resource Class Example

```
package dog;

public class Dog implements java.io.Serializable {
    public static final String DOG_NAME = "wangwang";
    public static final int DOG_AGE = 1;

    private static Dog instance = new Dog();

    private int age = DOG_AGE;
    private String name = DOG_NAME;

    public Dog() {}

    public static Dog getInstance() {
        return instance;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public String getName() {
        return name;
    }
}
```

```

    }

    public void setName(String name) {
        this.name = name;
    }

    public boolean equals(Object anObject) {
        if (this == anObject) {
            return true;
        }
        if (anObject instanceof Dog) {
            Dog anotherDog = (Dog) anObject;
            return (this.age == anotherDog.age && this.name.equals(anotherDog.name));
        }
        return false;
    }
}

```



To dynamically add a custom resource, the applicable classes must exist in the class path of the server class loader. If the class loader cannot load the classes, then the add command, which is normally executed dynamically, is processed as pending. In this case, add the custom resource classes to the 'SERVER\_HOME/lib/application' or 'DOMAIN\_HOME/lib/application' folder and restart the server.

Deleting a custom resource is not performed gracefully. In other words, even if there is a request in progress, the system does not wait until the request is complete which may cause errors to occur in the user applications.

## Using the Console Tool

A custom resource can be looked up and dynamically added or deleted using the console tool.

```

[MASTER]domain1.adminServer>add-custom-resource custom/dog -resource dog.Dog -factory dog.DogFactory
Successfully performed the ADD operation for A custom resource.
Check the results using "list-custom-resources or add-custom-resource"

```

```

[MASTER]domain1.adminServer>list-custom-resources
List of Custom Resources

```

```

=====
+-----+-----+-----+-----+
| Export Name | Resource Class | Factory Class | Properties |
+-----+-----+-----+-----+
| custom/dog | dog.Dog        | dog.DogFactory | [test=1,   |
|             |                 |                 | test1=2]   |
+-----+-----+-----+-----+
=====

```

```

[MASTER]domain1.adminServer>add-custom-resource-to-servers custom/dog -servers server1
Successfully performed the ADD operation for A custom resource.
Check the results using "list-custom-resources"

```

```
[MASTER]domain1.adminServer>remove-custom-resource custom/dog
Successfully performed the REMOVE operation for A custom resource.
Check the results using "list-custom-resources or remove-custom-resource"
```

```
[MASTER]domain1.adminServer>list-custom-resources
List of Custom Resources
```

```
=====
+-----+-----+-----+-----+
| Export Name | Resource Class | Factory Class | Properties |
+-----+-----+-----+-----+
(No data available)
=====
```

## 5.2.5. Configuring an External Resource

External sources are largely divided into JMS sources and Connectors.

### JMS Source

To add a JMS source, you need to edit domain.xml as follows:

```
<resources>
  <external-source>
    <jms-source>
      <vendor>ibmmq</vendor>
      <factory-class-name>FirstClassName</factory-class-name>
      <resource-type>QCF</resource-type>
      <export-name>exportName</export-name>
      <queue>MQ</queue>
      <queueManager>MQManager</queueManager>
    </jms-source>
  </external-source>
</resources>
```

The following describes the JMS Source configuration items.

Item	Description
Vendor	<p>JMS vendor. Select one of:</p> <ul style="list-style-type: none"> <li>◦ ibmmq: IBM product.</li> <li>◦ sonicmq: Sonic MQ.</li> <li>◦ others: other product.</li> </ul>
Factory Class Name	Factory class name of the JMS resource.

Item	Description
Resource Type	<p>JMS type.</p> <p>Select one of:</p> <ul style="list-style-type: none"> <li>◦ QCF</li> <li>◦ TCF</li> <li>◦ Q</li> <li>◦ T</li> <li>◦ XAQCF</li> <li>◦ XATCF</li> <li>◦ LOCALXAQCF</li> <li>◦ LOCALXATCF</li> </ul>
Export Name	Name to bind to JNDI. Can be used to obtain JMS ConnectionFactory and Destination.
Queue	Use only when the resource type is Q.
QueueManager	Use only when the resource type is Q.
Topic	Use only when the resource type is T.
Property	Necessary attributes for JMS resources. It has a name, a type, and a value.



Refer to the IBM MQ or the Sonic MQ manual for more information about each item.

## Adding a Connector

You can add a Connector by editing domain.xml as follows:

```
<resources>
  <external-source>
    <connector>
      <resource-adapter-module-id>connectorModuleId</resource-adapter-module-id>
      <worker-pool>
        <min>0</min>
        <max>5</max>
        <keep-alive-time>60000</keep-alive-time>
        <queue-size>4096</queue-size>
        <shutdown-timeout>-1</shutdown-timeout>
      </worker-pool>
    </connector>
  </external-source>
</resources>
```



For more information about the Connector, see "JEUS Jakarta Connectors Guide".

## 5.2.6. Configuring an External Resource

External resources can be configured so that JEUS can integrate with Tmax or Infinite Cache. For integration with Tmax, JEUS connects to Tmax to configure WebT and JTmax. WebT is an outbound service that uses Tmax transaction services, and JTmax is an inbound service that receives requests from Tmax. Refer to the JTmax Server Guide of the Tmax manual for more information.

This section describes how to implement and register the external resources.

The following is the `jeus.external.ResourceBootstrapper` interface. The class that implements this interface must reside in the 'SERVER\_HOME/lib/application' or 'DOMAIN\_HOME/lib/application' directory.

`jeus.external.ResourceBootstrapper`

```
package jeus.external;

import javax.naming.Context;
import java.util.Map;

/**
 * A bootstrapper to use external resources in JEUS.
 */
public interface ResourceBootstrapper {
    /**
     *
     * @param propertyMap A Map with resource settings.
     */
    void setProperties(Map propertyMap) throws InvalidPropertyException;

    /**
     * Binds a resource.
     * @param context
     */
    void initResources(Context context);

    /**
     * Returns information about available properties.
     * @return
     */
    ResourcePropertyInfo[] getPropertyInfo();

    /**
     *
     * @param propertyMap A map with properties to be modified.
     */
    void modifyProperties(Map propertyMap) throws InvalidPropertyException;

    /**
     * To be called when re-binding the resource.
     * @param context
     */
    void reconfigResources(Context context);
}
```

```

/**
 * Removes the resource.
 * @param context
 */
void destroyResources(Context context);
}

```



To dynamically add an external resource, the applicable classes must exist in the class path of the server class loader. If the class loader cannot load the classes, then the add command, which is normally executed dynamically, is processed as pending. In this case, add the external resource classes to the 'SERVER\_HOME/lib/application' or 'DOMAIN\_HOME/lib/application' folder and restart the server.

Deleting an external resource is not performed gracefully. This means that an incomplete request may cause errors to occur in the user applications.

## Using the Console Tool

An external resource can be looked up and dynamically added or deleted by using the console tool.

```

[MASTER]domain1.adminServer>add-external-resource test/ext -resource
test.ext.TestResourceBootstrapper
Successfully performed the ADD operation for A external resource.
Check the results using "list-external-resources or add-external-resource"

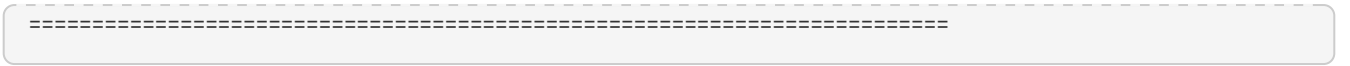
[MASTER]domain1.adminServer>list-external-resources
List of External Resources
=====
+-----+-----+-----+
| Name      | Resource Class                  | Properties |
+-----+-----+-----+
| test/ext  | test.ext.TestResourceBootstrapper | []         |
+-----+-----+-----+
=====

[MASTER]domain1.adminServer>add-external-resource-to-servers test/ext -servers server1
Successfully performed the ADD operation for A external resource.
Check the results using "list-external-resources"

[MASTER]domain1.adminServer>remove-external-resource test/ext
Successfully performed the REMOVE operation for A external resource.
Check the results using "list-external-resources or remove-external-resource"

[MASTER]domain1.adminServer>list-external-resources
List of External Resources
=====
+-----+-----+-----+
| Name      | Resource Class                  | Properties |
+-----+-----+-----+
(No data available)

```



## 6. DB Connection Pool and JDBC

This chapter describes the basic mechanism of JDBC connection pooling provided by JEUS, how to use the JDBC connection pool, and the data source management method used in the JEUS domain structure.

### 6.1. Overview

Web applications usually use a database (DB) to store data. Web application servers (WAS) like JEUS need to communicate with a DB in order to provide database dependent services, such as connection pooling, to applications. The Java Database Connectivity (JDBC) standard defines the interface between the DB clients and DB for an efficient and systematic communication.

The JDBC standard describes how to use DB connections in applications. It also describes how to perform SQL operations and provides related APIs. For more information about the JDBC standard, refer to Sun's JDBC web page ([Oracle JDBC](#)).

JEUS provides connection pooling and other additional services to applications based on JDBC 4.0.

### 6.2. Data Sources and JDBC Connection Pooling

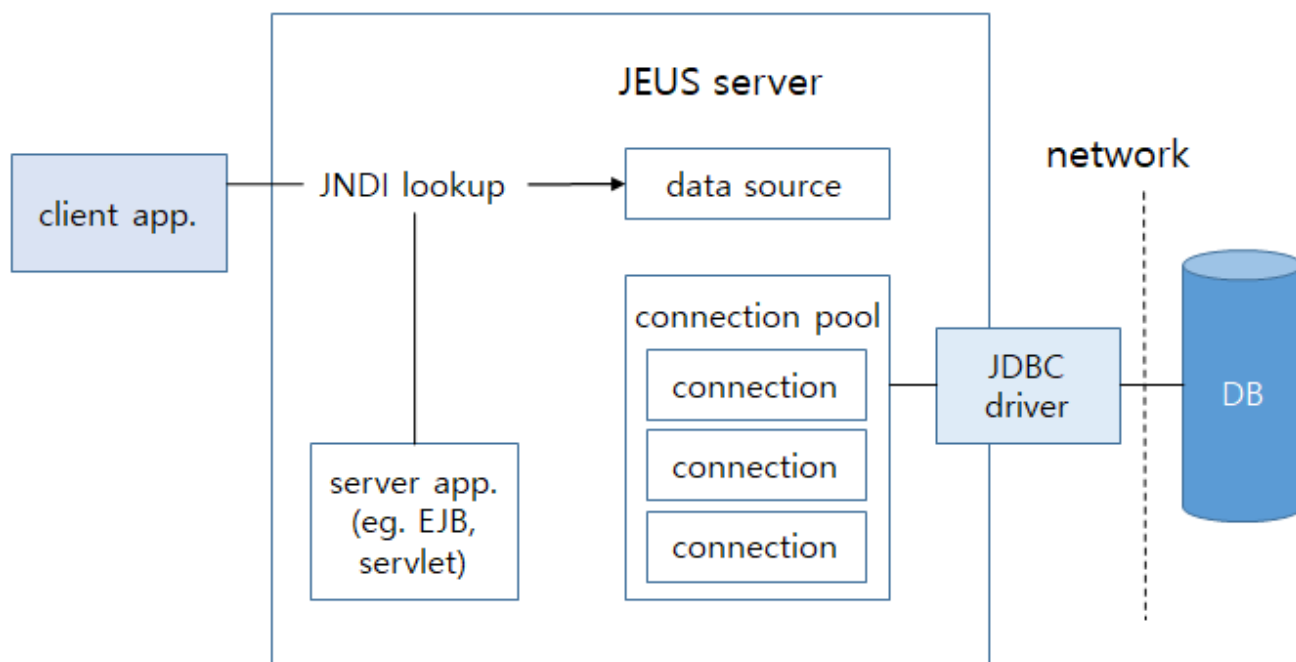
Data source is an object abstracted as a Factory that provides JDBC connections to applications. It internally constructs a JDBC connection pool and provides the connection pooling service.

JDBC connection pooling is a JDBC connection management service. JDBC connection pooling gets and stores a certain number of connections from the DB, and provides them whenever the applications need them. JDBC connection pooling service collects and reuses the connections after use.

Applications do not directly request for a connection to the DB. Instead, they use JNDI lookup to obtain a data source for the DB and requests for a connection through the data source. The data source that received the request sends a connection from its own connection pool to the application. This is how a connection request is processed.

After the application finishes using the connection, JEUS collects the connection and performs the required connection clean up operation. In order to reuse the connection, JEUS puts the connection back into the connection pool and manages it.

The following is the JDBC connection pooling mechanism used on JEUS server.



JEUS Connection Pooling



Instead of creating a connection pool and initializing it at server startup, JEUS creates and initializes a connection pool when a connection request to a data source occurs for the first time in an application.

### 6.2.1. JDBC Driver

Since the JDBC Driver is a collection of API implementations that are required for communicating with the DB, JEUS can provide connection pooling and other services by integrating with JDBC certified drivers. Certified drivers are provided through each vendor's website.

JEUS JDBC environment configuration can vary depending on the DB vendor. This is because the properties that each driver requires is different and the detailed properties that need to be configured must be checked in each vendor's JDBC driver manual.

### 6.2.2. JDBC Connection Pool

JDBC connection pool is a single framework that allows users to use and manage DB connections more efficiently and provides the connection pooling service.

The following are the benefits of using a connection pool.

- **Higher performance**

The system performance can be improved. Creating and deleting DB connections can put a burden on the system. By creating and reusing connections in the connection pool, overhead incurred from creating and deleting connections every time a connection is needed is significantly reduced.

- **Managing connections**

The number of concurrent connections can be controlled. Since the number of concurrent connections is limited to the configured value, excessive load is not put on the DB.

### 6.2.3. Data Source

A data source is an interface between applications and connection pools, and applications view a data source object as a DB Connection Factory. Data source connections provide more benefits than `java.sql.DriverManager` connections.

#### Data Source Types

There are three types of data sources.

- **Default data source**

`javax.sql.DataSource` type. This is the default data source type and cannot be used for connection pooling.

- **Connection pool data source**

`javax.sql.ConnectionPoolDataSource` type. JEUS provides a connection pool for this type.

It is usually used in the following situations.

- When creating applications that can access a database without using XA.
- When configuring auto-commit to "false" and manually controlling a local transaction.

```
import java.sql.Connection;

Connection conn = datasource.getConnection();
conn.setAutoCommit(false);
...DB access code... // Local transaction in JDBC driver
conn.commit();
```

- **XA data source**

`javax.sql.XADataSource` type. JEUS provides connection pool for this type. It supports connection pooling as well as integration with global transactions (hereafter XA).

It is usually used in the following situations.

- When a Jakarta EE component logic like servlet/EJB needs to access two or more DBs.
- When related logics need to be tied as a sequence of operations even when a Jakarta EE component accesses a single DB.

## XA Emulation of Connection Pool Data Sources

The local transaction optimization function can be used to increase XA processing performance. It emulates the connections obtained from connection pool data sources so that they can participate in XA transactions. In JEUS 6, XA emulation was enabled by using a local transaction data source type. In JEUS 9, you can set the XA emulation flag in the connection pool data source.

This function is usually used in the following situations.

- When the DB doesn't support XA, or the JDBC driver doesn't support the `javax.sql.XADataSource` implementation.
- When applications need to use XA, but don't want to use an XA data source due to performance problems (basically, when local transaction optimization is needed).

Currently, DBs or JDBC drivers that do not support XA are almost never used. Therefore, this function will usually be used for transaction optimization, but only one local XA data source can participate per XA transaction. A local XA data source is used in the following situations.

- When Jakarta EE component logics like servlet/EJB use only one DB which does not require the use of XA data source.
- When using one of multiple DBs used in the Jakarta EE component to process a local transaction to enhance performance.

Note that when multiple DBs are used, transaction recovery may not be properly executed since the connection pool data source that uses the XA emulation function does not support 2-phase commit (2PC).



The XA emulation of a connection pool data source is viewed as a local transaction with auto commit turned off. From the viewpoint of the DB, this is a different type of transaction than XA transaction that is managed by the JEUS transaction manager. Instead, JEUS uses emulation to allow a local transaction to participate in the XA transaction, so that the application can view the transaction as a single transaction.

### 6.2.4. Cluster Data Source

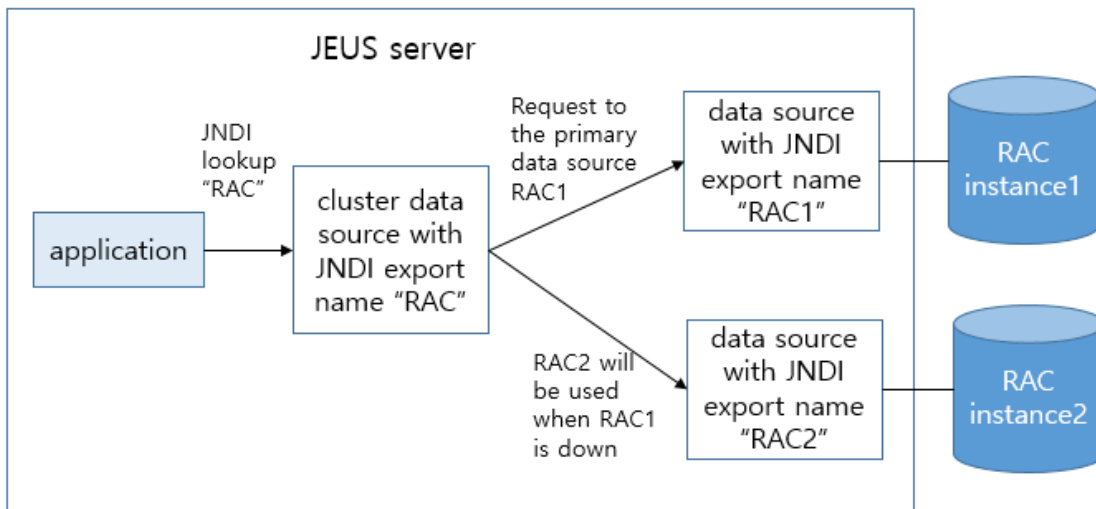
The following describes features of failover, Data Source affinity, and associating with ONS for cluster data sources.

#### 6.2.4.1. Failover

A cluster data source is provided to support failover and failback between RAC instances at the JEUS level. Real application cluster (RAC) is a DB clustering function provided by Oracle. For more information about RAC, refer to Oracle documentation.

A cluster data source is basically a data source instance with a single JNDI name, and it logically ties

multiple individual data sources and manages them. An individual data source (hereafter component data source) that belongs to a cluster data source is also an independent data source with its own individual JNDI name and must be configured as a data source of an RAC instance. A cluster data source operates by delegating connection requests it receives from applications to one of the component data sources. A connection request is delegated only to a component data source that has been verified to process requests successfully. This allows for transparent cluster data source failover in applications.



RAC Cluster Data Source Failover

It is recommended that JEUS cluster data sources be used instead of connect time failover (CTF) provided by Oracle's JDBC driver. Since Oracle CTF performs failover of each connection, recovering all connections in the pool may take a long time when there's a problem with an entire data source. However, JEUS cluster data sources detect problems in each data source and performs failover in data source unit which is more efficient. Furthermore, JEUS cluster data sources provide an automatic fallback function.

In the former method, since JEUS is responsible for failover, check-query and check-query-period have to be configured in order to check for component data source failure in each RAC instance. However, in the latter mechanism, since the driver is responsible for failover, check-query setting is not required. For more information about the cluster data source configuration, refer to [Cluster Data Source Configuration](#).

#### 6.2.4.2. Data Source Affinity

In RAC, global transactions should be processed in one RAC instance rather than in physically distributed multiple RAC instances for better performance. Also, only one RAC instance must be used when forcibly processing local transactions in RAC by using a cluster data source of which the member DataSource is ConnectionPoolDataSource with XA emulation configured.

In global or local transactions, a cluster data source keeps information that which component data source and RAC instance process the first connection request and then requests connections to the component data source. This is data source affinity that guarantees that global or local transactions are processed in one RAC instance.



When multiple cluster data sources are associated with one RAC instance, all their requests are processed in the RAC instance. When multiple RACs are associated, data source affinity is also guaranteed for each RAC. If a and b cluster data sources are associated with RAC A and B respectively, the RAC A and B guarantee data source affinity for connection requests of the a and b cluster data sources respectively. Data source affinity is not guaranteed between RAC A and B.

If data source affinity is set, failover, failback, and load balancing will be ignored, and the same data source as an RAC instance associated with a transaction will be used. If a transaction-associated data source is not available, a data source will be used according to a failover, failback, or load balancing policy.

#### **6.2.4.3. Cluster Data Sources Associated with ONS**

Oracle Notification Service (ONS) enables to share status information between RAC nodes. If ONS is set for RAC, each node in the RAC shares its status with all the other nodes. Non RAC server components can also get RAC node status if they participate in ONS as an ONS client. Through this, JEUS provides cluster data sources with enhanced features related to ONS.

Cluster data sources associated with ONS internally map each component data source to a relevant RAC instance. This allows to use RAC instance information obtained from ONS for relevant component data source management.

JEUS can get the following status information from ONS as an ONS client.

- Up & Down Notification of RAC instance  
Notifies when an RAC instance starts or ends.
- Available capacity (%) of each RAC instance (also called runtime load balancing advisory)  
Available capacity (%) of each RAC instance out of 100%.

#### **Functions of Cluster Data Sources Associated with ONS**

Cluster data sources associated with ONS provide the following functions.

- Efficient component data source status management  
To monitor DB status, polling was used. However, polling has the following issues; it lowers performance because executed during a request processing, and cannot monitor DB status when a network is disconnected. To resolve the issues, RAC instance up & down notification can be used. This not only resolves the polling issues but also efficiently detects whether component data sources are failed or recovered.

- Efficient load balancing by using runtime load balancing advisory

When available capacity of DB cannot be monitored, only simple round-robin can be used for load balancing. Now more efficient load balancing can be performed because available capacity of each RAC instance can be monitored through runtime load balancing advisory in real time. For

example, if component data sources A and B are associated with RAC instances A and B and have 60% and 40% of available capacity respectively, 60% of connection requests are transferred to A and 40% of them are transferred to B.

## ONS Settings

To operate JEUS as an ONS client, ONS library must be installed.

Download ons.jar from an Oracle support site to the `$JEUS_HOME/lib/datasources` directory. After configuring ONS-related options for cluster data sources, cluster data sources associated with ONS can be used. First, configure an IP address and a port number used by each RAC node in ONS for ONS communications. This configuration associates cluster data sources and ONS. Cluster data sources associated with ONS can efficiently detect whether component data sources are failed or recovered through ONS by using failover/failback or load balancing. Especially load balancing can be performed more efficiently through runtime load balancing advisory.

## 6.3. Management of Data Sources and Connection Pools

Since JEUS has been expanded to a domain structure, the management structures of data source and connection pools have been partially changed. After understanding the underlying concept of the expanded domain structure, it should not be difficult to grasp the knowledge of the changed data source and connection pool management structures.

The minimum unit for running services on JEUS is the **server** and depending on the circumstances, services can run on multiple servers grouped into a cluster. Servers and clusters can be grouped into a single management unit called a **domain**. A domain includes a set of servers and clusters that need to be managed in connection to each other. It manages the servers and clusters in the domain and the **Master Server** (MASTER) manages all services associated with the domain. Excluding MASTER, all servers in the domain are called **Managed Servers**(MSs). MASTER can function as both MASTER and MS simultaneously, but it is recommended to only use MS to provide services and only use MASTER to manage the domain. For more information about JEUS domain structure, refer to *JEUS Domain Guide*.

Now, let's look into how data sources and connection pools are managed in the domain structure.

By default, a data source is a resource that is exposed to a domain. This means that the servers and clusters in a domain (more accurately, servers in a cluster) reference the exposed data source configurations to create their own connection pool and provide the connection pool service. A cluster is an abstraction of a group of servers, but it isn't the actual subject that provides the services. Data sources which a cluster references are available on the servers in the cluster. Therefore, the servers in the cluster can reference the data sources that are referenced by the cluster. The servers in a cluster can also individually configure data source references. The server can reference the data sources configured both in the cluster and on itself.

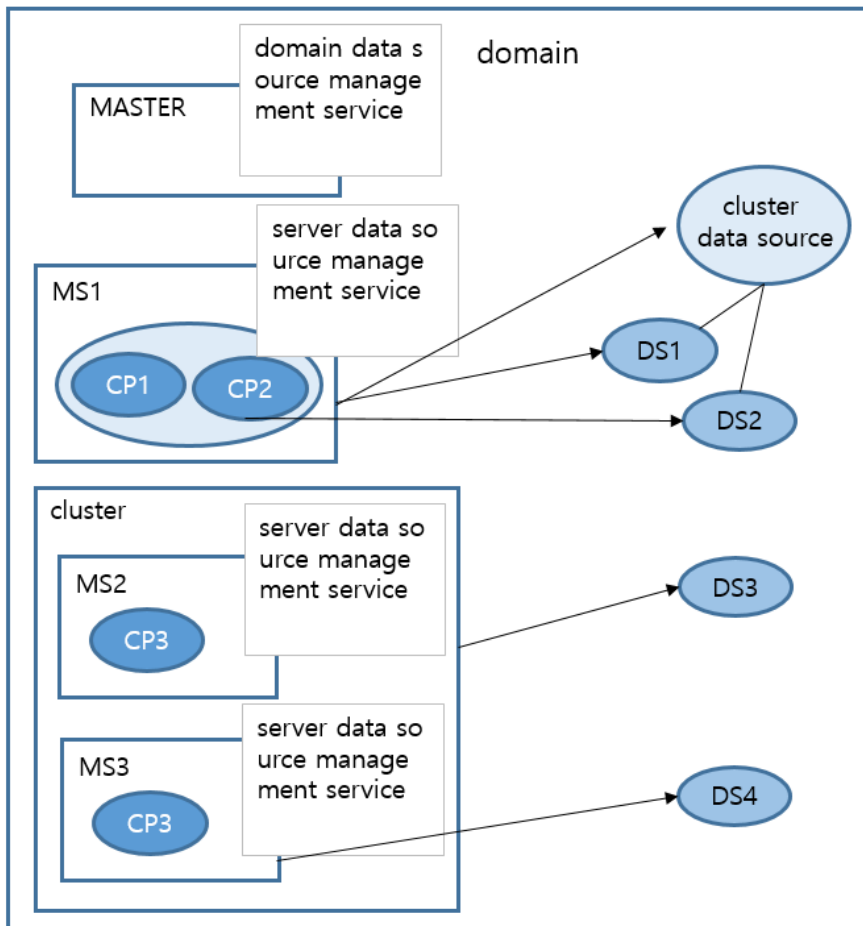
In order to reference a data source on a server or cluster, the data source ID must be registered on the server or cluster. Hence, when a server or a cluster registers a data source ID, the server and servers in the cluster read the registered data source configuration and prepare the information

needed to create a connection pool. The information is mapped to the JNDI name of the data source and bound to the server's JNDI repository. Once this process is completed, the application deployed on the server can create and use a connection pool by looking up the data source using its JNDI name.



Data sources are bound to JNDI at the server level. The same JNDI names can be used as long as different data sources are not bound to the JNDI on the same server. This means that data sources with the same JNDI names cannot be referenced simultaneously on the same server. Data sources are distinguished by the data source ID instead of JNDIs because different data sources can have the same JNDI names. Each data source ID must be set to a unique value at the domain level so that it can be used as an identifier.

The following shows a more detailed example of a domain that consists of MASTER and three MSs.



Data Source and Connection Pool Management in JEUS Domain Structure

In the previous example, MS1 is registering a cluster data source. Since the cluster data source's component data sources are DS1 and DS2, in order for MS1 to fully use the cluster data source, the component data sources (DS1 and DS2) of the cluster data source must be registered to MS1. With this configuration, after creating the connection pool with DS1 and DS2, MS1 can group them into a cluster and provide the cluster connection pool service.

MS2 and MS3 are also in the cluster. DS3 is registered to the cluster, and thus DS3 is available on

both MS2 and MS3 which are in the cluster. This means that MS2 and MS3 can each create a DS3 connection pool to provide the connection pool service.

In addition, MS3 registers DS4 to itself. Now, MS3 can provide connection pool service for both DS3 registered to the cluster and DS4 registered to itself.

Each MS runs the data sources and connection pool management services at the server level while MASTER runs them at the domain level. MASTER service is connected to each MS service. It manages all data sources and connection pools, and processes the data source and connection pool requests from the console tool.

## 6.4. Data Source Configuration

The first thing to do in using data sources and JDBC connection pools in JEUS is to check if the JDBC driver library exists in the '\$JEUS\_HOME/lib/datasource/' directory, and to configure the required data source settings. Data source configuration includes the basic configuration for JDBC driver and connection pool configurations.

This section briefly explains how to configure data sources using the console tool. For detailed instructions using the console tool, refer to "add-data-source" in *JEUS Reference Guide*.

```
[MASTER]domain1.adminServer>add-data-source -id ds1 -dst ConnectionPoolDataSource -dscn
oracle.jdbc.pool.OracleConnectionPoolDataSource -sn 61.77.153.4 -pn 1521 -dn orcl -user scott
-password tiger --property driverType:java.lang.String=thin
Successfully performed the ADD operation for data source [ds1] to domain.
Check the results using "add-data-source".
[MASTER]domain1.adminServer>addds
Shows the current configuration.
Data sources in domain
=====
+-----+-----+-----+-----+
| ds1 | common data source |
+-----+-----+-----+-----+
=====

[MASTER]domain1.adminServer>
```

### DataSourceAccountProvider Interface

By default, JEUS allows you to configure DB access account information in data sources and stores it in the JEUS configuration repository. Through this, users do not need to pass the DB access account information as parameters every time JDBC connection is requested by user applications.

Additionally, the DB access account information can be stored in the repository by users after being encrypted through integration with JEUS Security for secure management.

In addition, JEUS supports storing the DB access account information to be used in data sources in an external repository, rather than in the one configured in JEUS. For protecting DB access account information, JEUS also supports the use of other external security modules other than JEUS Security.

To facilitate such flexible integration with external systems, JEUS specifies the `jeus.jdbc.helper.DataSourceAccountProvider` interface. Users can manage DB access account information in various ways by appropriately implementing this interface to integrate with their desired external systems.

The following is the specification of `jeus.jdbc.helper.DataSourceAccountProvider` interface.

`jeus.jdbc.helper.DataSourceAccountProvider`

```
public interface DataSourceAccountProvider {  
    String getUser(Map<String, String> dataSourceConfigurationMap)  
        throws DataSourceAccountProviderException;  
    String getPassword(Map<String, String> dataSourceConfigurationMap)  
        throws DataSourceAccountProviderException;  
}
```

Method	Description
<code>getUser(Map&lt;String, String&gt; dataSourceConfigurationMap)</code>	Receives a Map containing data source configuration as a parameter from JEUS and returns plaintext user values for DB access.
<code>getPassword(Map&lt;String, String&gt; dataSourceConfigurationMap)</code>	Receives a Map containing data source configuration as a parameter from JEUS and returns plaintext password values for DB access to JEUS.

The Map passed as a parameter to each API of the `jeus.jdbc.helper.DataSourceAccountProvider` interface, which was introduced earlier, includes some of the JEUS data source configuration values. The implementations can obtain the values corresponding to following keys.

Key	Value
<code>DATA_SOURCE_ID</code>	Data source ID entered in the JEUS data source configuration.
<code>EXPORT_NAME</code>	Export name entered in the JEUS data source configuration.
<code>VENDOR</code>	Vendor entered in the JEUS data source configuration.
<code>DATA_SOURCE_CLASS_NAME</code>	Data source class name entered in the JEUS data source configuration.
<code>SERVER_NAME</code>	Server name entered in the JEUS data source configuration.
<code>PORT_NUMBER</code>	Port number entered in the JEUS data source configuration.
<code>DATABASE_NAME</code>	Database name entered in the JEUS data source configuration.
<code>USER</code>	User entered in the JEUS data source configuration.
<code>PASSWORD</code>	Password entered in the JEUS data source configuration.

The `DataSourceAccountProviderException` thrown by each API of the `jeus.jdbc.helper.DataSourceAccountProvider` must occur in the following situation.

Exception Name	Exception Occurs When
jeus.jdbc.datasource.DataSourceAccountProviderException	It fails to obtain the plaintext user or password value required to access the DB.

To use the implementation of the jeus.jdbc.helper.DataSourceAccountProvider interface, which was implemented as intended by the user, the class name must be specified as the following jvm-option to the JEUS server.

JVM Option Name	JVM Option Value
jeus.jdbc.config.data-source-account-provider-class-name	Full name of the jeus.jdbc.helper.DataSourceAccountProvider interface implementation.

If the users does not configure the above jvm-option, JEUS uses the default implementation that integrates with JEUS Security.

Based on the specification above, a customized DataSourceAccountProvider can be implemented as follows. In the following examples,the DB access account information is assumed to be 'scott/tiger'.

DataSourceAccountProvider Implementation Example 1 - Storing DB access account information in plaintext in JEUS configuration

```
public class MyDataSourceAccountProvider implements DataSourceAccountProvider {
    @Override
    public String getUser(Map<String, String> dataSourceConfigurationMap)
        throws DataSourceAccountProviderException {
        // Returns the user value retrieved by the JEUS configuration.
        return dataSourceConfigurationMap.get(USER); // Gets 'scott'.
    }

    @Override
    public String getPassword(Map<String, String> dataSourceConfigurationMap)
        throws DataSourceAccountProviderException {
        // Returns the password value retrieved by the JEUS configuration.
        return dataSourceConfigurationMap.get(PASSWORD); // Gets 'tiger'.
    }
}
```

DataSourceAccountProvider Implementation Example 2 - Storing DB access account information in a separate file

```
public class MyDataSourceAccountProvider implements DataSourceAccountProvider {
    @Override
    public String getUser(Map<String, String> dataSourceConfigurationMap)
        throws DataSourceAccountProviderException {
        // Gets 'user' by properly parsing the file with the SERVER_NAME retrieved from the JEUS
        configuration.
        File file = getFile(); // Gets the file containing the database access account information.
        // Internally gets the user value 'scott' by parsing the file.
        return getUserFromFileWithSID(file, dataSourceConfigurationMap.get(SERVER_NAME));
    }

    @Override
```

```

    public String getPassword(Map<String, String> dataSourceConfigurationMap)
        throws DataSourceAccountProviderException {
        // Gets the password by properly parsing the file with the SERVER_NAME value retrieved from
the JEUS configuration.
        File file = getFile(); // Gets the file containing the database access account information.
        // Internally gets the password value 'tiger' by parsing the file.
        return getPasswordFromFileWithSID(file, dataSourceConfigurationMap.get(SERVER_NAME));
    }
}

```

DataSourceAccountProvider Implementation Example 3 - Obtaining only DB access password by integrating with an external security module, ExternalSecurity.

```

public class MyDataSourceAccountProvider implements DataSourceAccountProvider {
    @Override
    public String getUser(Map<String, String> dataSourceConfigurationMap)
        throws DataSourceAccountProviderException {
        // Returns the user value retrieved from the JEUS configuration.
        return dataSourceConfigurationMap.get(USER); // Gets 'scott'.
    }

    @Override
    public String getPassword(Map<String, String> dataSourceConfigurationMap)
        throws DataSourceAccountProviderException {
        // Gets the data source setting values to be passed as parameters to an external security
module 'ExternalSecurity'
        // Gets the data source ID from the JEUS data source configuration.
        String dataSourceID = dataSourceConfigurationMap.get(DATA_SOURCE_ID);
        // Gets the server name from the JEUS data source configuration.
        String serverName = dataSourceConfigurationMap.get(SERVER_NAME);
        // Gets the user value from the JEUS data source configuration.
        String user = dataSourceConfigurationMap.get(USER);

        // Passes dataSourceID, serverName, and user as parameters to the external security module
'ExternalSecurity', and then
        //receives the corresponding password from ExternalSecurity.
        return ExternalSecurity.getPassword(dataSourceID, serverName, user); // Gets 'tiger'.
    }
}

```

### 6.4.1. Connection Pool Configuration

You can check the current data source settings with the list-data-sources command in jeusadmin.

```

[MASTER]domain1.adminServer>list-data-sources -id ds1
The configuration of the data source [ds1]
=====
+-----+-----+
| Configuration Name | Configuration Value |
+-----+-----+
| id                 | ds1                  |
| export-name        | ds1                  |
| data-source-class-name | oracle.jdbc.pool.OracleConnectionPoolDataSource |
| data-source-type    | ConnectionPoolDataSource |

```

```

server-name          | 61.77.153.4          |
port-number          |                      | 1521 |
database-name        | orcl                 |
user                 | scott                |
password             | tiger               |
login-timeout        |                      | 0    |
auto-commit          | DRIVER              |
stmt-query-timeout   |                      | 0    |
pool-destroy-timeout |                      | 10000|
property             | [driverType;java.lang.String;thin]
support-xa-emulation | false               |
min                  |                      | 2    |
max                  |                      | 30   |
step                 |                      | 1    |
period               |                      | 3600000|
enable-wait          | false               |
wait-time            |                      | 10000|
max-use-count        |                      | 0    |
dbaTimeout           |                      | -1   |
stmt-caching-size    |                      | -1   |
stmt-fetch-size      |                      | -1   |
connection-trace     | false               |
get-connection-trace | true                |
auto-commit-trace    | false               |
use-sql-trace        | false               |
keep-connection-handle-open| false               |
+-----+-----+
=====
[MASTER]domain1.adminServer>

```

The following describes each configuration item.

- **Pooling**

JDBC connection pool size and related settings.

Item	Description
Min	Minimum number of connections in the connection pool.
Max	Maximum number of connections in the connection pool.
Step	Number of DB connections to get, if connections are insufficient and the number of connections in the connection pool is smaller than the maximum value.
Period	Interval for adjusting the connection pool size based on the minimum value.  If the connection pool size is bigger than the minimum value, unused connections are closed. If the size is smaller than the minimum value, new DB connections are added. (Unit: milliseconds)

- **Wait free connection**

Method for handling a connection request when all connections in the connection pool are occupied.



Item	Description
Enable Wait	<p>Method for handling connection requests, when there are no available connections in the pool and no more connections can be added.</p> <ul style="list-style-type: none"> <li>• true: Wait for an available connection.</li> <li>• false: Create a new connection, but it will be discarded without entering the pool after use. This type of connection is called a disposable connection.</li> </ul>
Wait Time	<p>Timeout to wait for a connection if "enable-wait" is "true". If a connection cannot be obtained within the time period, JEUS generates a timeout exception. (Unit: milliseconds)</p>

#### • Connection validation

The function that executes a specific query to validate the connection status before passing the connection to the application that requested it. Useful for checking disconnection due to an internal JDBC connection error and socket disconnection due to firewall.

When there's a problem with the connection, a new connection from the DB is sent to the application. This configuration is required if the data source is in a cluster data source of RAC.

Item	Description
Check Query	<p>Since this is generally used to only validate DB connections, it is recommended to use a simple select query.</p> <p>Starting from JEUS 7 Fix # 2, you can use the isValid method added to java.sql.Connection in JDK1.6 instead of performing check-query to check database connection. You can enter "use isValid method" instead of the query statement.</p> <p>Configurations (Check Query Timeout, Check Query Period, etc.) related to connection validation are applied the same way when using the isValid method.</p>
Check Query Timeout	<p>When check-query is executed to check the connection status, the driver can be in the wait state indefinitely if the DB does not respond. This value is applied to the check-query to avoid this situation. (Unit: milliseconds)</p> <p>This value can be set by using the java.sql.Statement#setQueryTimeout method defined in the JDBC API.</p> <p>If the value is less than 1000ms, the value is set to 0.</p>

Item	Description
Non Validation Interval	<p>Option to reduce the overhead caused by frequent connection checks. (Unit: milliseconds) This configuration skips the connection check if the interval between the last check and the current check is within the interval.</p> <p>For example, if the configuration value is 5000 ms and 5 seconds has not passed since the last connection check, connections are sent to the applications without checking.</p>
Check Query Period	Option to check and delete faulty connections in a connection pool at a specified interval. Each data source in a cluster data source must configure this setting to check its own state. (Unit: milliseconds)
Check Query Class	<p>Class name including the package name that is implemented by users or developers to customize the connection check function.</p> <p>The class must implement the interface, <code>jeus.jdbc.connectionpool.JEUSConnectionChecker</code>. For more information, refer to <a href="#">JEUSConnectionChecker Interface</a>.</p>
Check Query Retrial Count	<p>When "Destroy Policy On Check Query" is set to the default value of "FAILED_CONNECTION_ONLY", the connection check is executed only once.</p> <p>When "Destroy Policy On Check Query" is set to "ALL_CONNECTIONS", if a connection problem is detected during the initial connection check, then another connection check is performed for another connection. A total of two connection checks can be performed, and they increment the basic connection check count that determines the final total check count.</p>
Destroy Policy On Check Query	<p>Policy for other connections in the connection pool when invalid connections are detected.</p> <ul style="list-style-type: none"> <li>• FAILED_CONNECTION_ONLY: Delete only invalid connections.</li> <li>• ALL_CONNECTIONS: Delete the invalid connections and validate other connections in the connection pool. If another invalid connection is detected in the pool, delete all connections from the connection pool.</li> </ul>

## • Connection pool

Add-on functions of the connection pool.

Item	Description
Delegation Datasource	<p>When a request does not involve a transaction, it is better to get a connection through a connection pool data source instead of an XA data source.</p> <p>This is because the XA connection, which involves transaction-related functions, gives more burden on the system, and in this case, the two provide the same functionality. When using the XA data source, specify the connection pool data source to delegate connection requests that are not related to transactions.</p> <p>This setting can be used to prevent unexplained exceptions that may occur when Oracle and DB2 use XA connections for both with or without transactions.</p>
Max Use Count	Maximum number of times a connection can be used. If a connection is used more than the specified number, the connection will be replaced by a new connection. The default value is 0, meaning connections are not replaced.
Delegation Dbա	<p>JNDI name of the data source (hereafter DBA delegation data source) that has the permission to forcibly terminate database sessions (DBA permission). If there is a delay in handling a query through the connection received from the data source, a query to forcibly terminate the DB session is sent to the DB through the DBA delegation data source.</p> <p>After the application handles exceptions that occurred due to the disabled connection, it closes the connection. Then JEUS deletes the connection and gets a new connection from the DB and puts it in the connection pool.</p> <p>Currently, this function is supported for Tibero, Oracle, and Sybase. This function is used to suspend queries that take too long to process for JDBC driver version 2.0 or earlier. For JDBC driver version 3.0 or later, it is recommended to use <code>java.sql.Statement#setQueryTimeout</code> instead of forcibly terminating DB sessions.</p> <p>Especially for XA data sources, if DB sessions are terminated while XA is normally processing, the XA operation can generate an error. In this case, the statement query timeout and transaction timeout properties should be used instead.</p>
Dbա Timeout	<p>Timeout for the delegation DBA data source to wait for a query to finish using a connection. If the time expires, it sends a query to the DB to forcibly terminate the DB session.</p> <p>This only applies when the Delegation Dbա is set. (Unit: milliseconds)</p>
Stmt Caching Size	JDBC driver parses the SQL statements sent as parameters whenever an application requests for a PreparedStatement. Since this can affect the system performance, JEUS provides a function that internally caches the prepared statements. This sets the number of prepared statements to be cached.

Item	Description
Stmt Fetch Size	Fetch size of a JDBC driver statement.
Use Sql Trace	<p>Option to display SQL queries running on each connection. If the jeus.jdbc.sql logger level is set to FINE, the SQL query history can be checked in the server logs.</p> <p>If set, the statement implementation class of the JDBC driver will be wrapped by JEUS class. Therefore, the applications that cast and use the statement objects of the JDBC driver cannot use this function.</p>
Keep Connection Handle Open	<p>Option to always keep the connection handle (or logical connection) open when the connection pooling is used.</p> <p><b>[Note]</b></p> <ol style="list-style-type: none"> <li>1. It is recommended to use this setting when the Universal Driver (JCC) type 4 provided by IBM DB2 is using the XA data source. This is because "hang up" occurs when threads open and close connection handles that are physically different from each other.</li> </ol> <p>Since the threads internally use and share the same vector (java.util.Vector), when a thread occupies a vector lock and falls into an infinite loop, then other threads are also put in an infinite wait for the lock. The root cause of the hang up is unknown, not the internal logic of DB2. However, according to the test result, no error occurred when the connection handle is always left open. When the connection handle is always left open, the thread only accesses the internal share vector when the connection handle is created for the first time and when the physical connection is closed.</p> <ol style="list-style-type: none"> <li>2. This bug has been fixed for DB2 driver 3.53.95 and later. IBM's official bug report number is 'APAR IZ41181' and can be found in the DB2 9.5 Fixpak 4 document.</li> </ol> <p><b>[CAUTION]</b></p> <p>If set, the connection handle cannot be closed which means that the driver does not clear the task when the connection is closed. For example, when using this setting with Oracle's JDBC driver and auto-commit set to false, if the connection is closed without executing a commit or rollback, then commit, which normally executes automatically, does not execute.</p>
Init Sql	SQL query that is processed first after a connection has been created.

#### • Connection Trace

Settings to enable connection trace.

Item	Description
Enabled	Settings to enable connection trace. <ul style="list-style-type: none"> <li>• false: overrides '<b>Get Connection Trace</b>' and '<b>Auto Commit Trace</b>' settings.</li> </ul>
Get Connection Trace	Allows the application to check the stack trace when <code>java.sql.DataSource#getConnection</code> is called.
Auto Commit Trace	Writes logs and stack trace in the server log when <code>java.sql.Connection#setAutoCommit</code> is called. You must set the log level of the <code>jeus.jdbc.connection-trace</code> logger to FINE to use this option.

## JEUSConnectionChecker Interface

The following is the specification of the interface mentioned in the Check Query Class configuration, `jeus.jdbc.connectionpool.JEUSConnectionChecker`. To use a custom check-query function, you must implement the interface and set the class as a Check Query Class.

`jeus.jdbc.connectionpool.JEUSConnectionChecker`

```
public interface JEUSConnectionChecker {
    void setConnectionPoolID(String connectionPoolID);
    void setQueryString(String query);
    void setQueryTimeout(int timeout);
    void checkConnection(Connection vcon) throws SQLException;
}
```

Method	Description
<code>setConnectionPoolID()</code>	The set value of the data source ID is passed as an argument. This can be used as informational data.
<code>setQueryString()</code>	If check query string is configured, it is given as an argument for use in the connection check.
<code>setQueryTimeout()</code>	If Check Query Timeout is configured, it is given as an argument for use in the connection check.
<code>checkConnection()</code>	Called during the actual connection check call. The developer can implement the tasks to perform during connection check in this method.

JEUS provides two default implementations for the `JEUSConnectionChecker` interface:

`jeus.jdbc.connectionpool.DefaultConnectionChecker` and  
`jeus.jdbc.connectionpool.TimeLimitedConnectionChecker`.

If users do not specify an option for the Check Query Class, the default implementation, `DefaultConnectionChecker`, is automatically set and operates. If users set `TimeLimitedConnectionChecker`, both the JDBC driver and JEUS itself handle the Check Query Timeout, enabling the connection validation. If `TimeLimitedConnectionChecker` is used, JEUS can continue with connection validation even if the JDBC driver hangs during query execution. However, this may create zombie threads. Therefore, to identify problems through logs, the logger level of

jeus.connectionpool.time-limited-connection-checker must be set more detailed than at least FINE.

## 6.5. Cluster Data Source Configuration

This section describes how to configure a cluster data source. For explanations on configuration using the console tool, refer to "add-cluster-data-source" in *JEUS Reference Guide*.

### 6.5.1. Configuring a Cluster Data Source

```
[MASTER]domain1.adminServer>add-cluster-data-source -id cds1 -cds ds1,ds2
Successfully performed the ADD operation for cluster data source [cds1] to domain.
Check the results using "add-cluster-data-source".
[MASTER]domain1.adminServer>addcds
Shows the current configuration.
Data sources in domain
=====
+-----+-----+-----+
| ds1 | common data source |
| ds2 | common data source |
| cds1 | cluster data source |
+-----+-----+-----+
=====
```

The following describes each configuration item.

- **Basic Settings**

Item	Description
Data Source Id	Cluster data source ID. It must be unique within the domain.
Export Name	Cluster data source JNDI name.  If two data sources are guaranteed to be bound to different JNDIs on different servers, they can have the same JNDI name. This means that data sources with the same JNDI names are not allowed on the same server. If not set, the cluster data source ID is used as the JNDI name.
Data Source Selector	When getting a connection from a cluster data source, users or developers can define the policy for selecting a component data source.  To define a selector, implement the jeus.jdbc.helper.DataSourceSelector interface, and include the implementation class package name in the class name. For more information, refer to <a href="#">DataSourceSelector Interface</a> .  If set, the Load Balance setting is ignored. You must define the policy taking synchronization into consideration.
Load Balance	Option to use load balancing. If set to true, the <b>Use Failback</b> setting is ignored.

Item	Description
Component Data Sources	ID of a component data source that is part of the cluster data source. Main data sources are determined based on the order in which they are entered.

#### • Advanced Options

Item	Description
Is Pre Conn	Option to create connection pool of component data sources in the cluster data source in advance. Creating the connection pool in advance can be beneficial for performance, but is not a good use of resources.
Use Failback	<p>Since only failover is supported in the earlier versions of JEUS, this option is provided for backward compatibility.</p> <p>This determines whether to failback using the main data source after failing over by using the assistant data source. By default, failback is used. To use failback, the '<b>Check Query</b>' and '<b>Check Query Period</b>' settings must be set for the main data source.</p>
DataSource Affinity	Option to set affinity for data source during a transaction. If this option is used, performance of handling global transactions will be improved because the transactions are processed in one member data source instance. In addition, local transactions with XA emulation enabled can be guaranteed when using a cluster data source.
Ons Support	<p>Specifies a cluster data source associated with ONS. Cluster data sources associated with ONS can efficiently detect whether component data sources are failed or recovered through ONS. In addition, more efficient load balancing can be supported by using runtime load balancing advisory.</p> <p>The following can be configured.</p> <ul style="list-style-type: none"> <li>Ons Config</li> </ul> <p>IP address and port number used for ONS communications of RAC nodes in ONS. A cluster data source establishes a socket connection by using the IP address and port number and then operates as an ONS client.</p> <p>Example:</p> <pre>nodes=host1:6200,host2:6200</pre>

### DataSourceSelector Interface

The following is the jeus.jdbc.helper.DataSourceSelector interface specification mentioned in the Data Source Selector configuration. The user can define a policy for selecting a component data source by implementing this interface and configuring the implementation class as a Data Source Selector.

```
public interface DataSourceSelector {
    public void setComponentDataSourceList(List<String> componentDataSourceList);
    public String selectDataSource();
}
```

Method	Description
setComponentDataSourceList( )	Gets the list of component data source IDs that belong to the cluster data source.
selectDataSource( e())	Defines the policy for selecting a component data source in a cluster data source. The return value is the ID of the data source selected through the defined policy. In most environments, multiple request threads can simultaneously access a connection obtained from a cluster data source. Hence, synchronization should usually be considered when implementing the policy.

The following DataSourceSelector interface can be implemented based on these specifications. This DataSourceSelector implementation has a 2:1 selection ratio for the two data sources in the cluster data source.

#### DataSourceSelector Implementation Example

```
package foo.bar;
import jeus.jdbc.helper.DataSourceSelector
public class MyDataSourceSelector implements DataSourceSelector {
    List<String> componentDataSourceList = new ArrayList<String>();

    // Ensures synchronization
    AtomicInteger dataSourceIndex = new AtomicInteger(0);

    public void setComponentDataSourceList(List<String> componentDataSourceList) {
        this.componentDataSourceList.addAll(componentDataSourceList);
    }

    public String selectDataSource() {
        int reminder = (dataSourceIndex.getAndIncrement() & 0x7fffffff) % 3;
        if(reminder < 2) {
            return componentDataSourceList.get(0);
        }
        else {
            return componentDataSourceList.get(1);
        }
    }
}
```

## 6.5.2. Configuring a Component Data Source in a Cluster Data Source

A cluster data source delegates the connection request processing to one of its component data



sources. If load balancing is not configured, then the cluster data source always delegates connection request processing to the main component data source. However, if an error is detected in the main component data source, then a new component data source is used as the main component data source in order to guarantee uninterrupted processing of connection requests. This is called the cluster data source failover.

To failback after a failover, it must be checked to see if the previous main component data source has been recovered. The check is done using the check queries that are periodically sent to the component data source. In order to use failback in the cluster data source, the '**Check Query**' and '**Check Query Period**' settings must be configured for each component data source in the cluster. To only use failover, set '**Use Failback**' to 'false'. Failover commands can be executed manually. For more information, refer to "control-cluster-data-source" in *JEUS Reference Guide*.



It is beneficial to set the 'Destroy Policy On Check Query' configuration of the component data source to 'ALL\_CONNECTIONS'. Otherwise, inaccessible connections can remain in the connection pool of the faulty component data source for some time after the failover. Although they are cleaned up during periodic connection checks, it is preferable that they are destroyed when failover occurs.

## 6.6. Dynamic Data Source Configuration

One of the major features in JEUS 9 is dynamic configuration. This means that some settings can be changed dynamically at runtime so that they are applied immediately without restarting JEUS. For more information about dynamic configuration, refer to "Changing Domain Configuration" in *JEUS Domain Guide*.

By using examples, this section describes how to dynamically change the configurations related to cluster data sources. For better understanding, knowledge about JEUS domain structure and data source management structure in the domain are required. Knowledge about using each configuration item of the cluster data source is also required. For more information about the JEUS domain structure, refer to "Domain Components" in *JEUS Domain Guide*. For explanation on the architecture of data source management in domains and usage of each data source configuration item, refer to [Management of Data Sources and Connection Pools](#) and [Data Source Configuration](#), respectively.

Dynamic changes can be applied using the console tool. Both methods are described in this section. Abbreviated commands and option names are used for describing the console tool method. For more information about how to use abbreviated commands and option names, execute help in the console tool or refer to "Data Source Commands" and "Connection Pool Control and Monitoring Commands" in *JEUS Reference Guide*.

First perform some preparation tasks. Create a domain named domain1 and a MASTER named adminServer. After starting the MASTER, add three MSs named server1, server2, and server3 to domain1 and start them. Then, combine server2 and server3 into a cluster named cluster1. For more information about how to configure the domain, clusters and servers, refer to "Creating a Domain"

and "JEU5 Clustering" in *JEU5 Domain Guide*, and [JEU5 Configuration](#) in this guide. From here on, it is assumed that the aforementioned preparation tasks have been completed.



Since all examples are related, it is recommended to read them in order.

## 6.6.1. Adding a Data Source

It is possible to add dynamic data sources. Adding a data source involves registering the data source configuration to the domain so that it can be referenced by the server and cluster in the domain.

### Using the Console Tool

You can use the **add-data-source** command to dynamically add a data source to the domain. To add a data source using **add-data-source**, you must configure each data source properties. For more information about **add-data-source**, refer to "add-data-source" in *JEU5 Reference Guide*.

The following example adds a data source with ID ds1 to domain1 by using the **add-data-source** command.

```
[MASTER]domain1.adminServer>add-data-source -id ds1 -dst ConnectionPoolDataSource -dscn
oracle.jdbc.pool.OracleConnectionPoolDataSource -sn 192.168.1.165 -pn 1521 -dn ora10g
-user jeustest1 -password jeustest1 -property driverType;java.lang.String;thin
Successfully performed the ADD operation for data source [ds1] to domain.
Check the results using "add-data-source"
```

Execute **add-data-source** to check the result.

```
[MASTER]domain1.adminServer>add-data-source
Shows the current configuration
Data sources in domain
=====
+-----+-----+-----+
| ds1 | common data source |
+-----+-----+-----+
=====
```

Repeat the previous steps to add two more data sources, ds2 and ds3 to domain1 to continue with the rest of the examples.

## 6.6.2. Registering a Data Source on a Server

To reference and use the data sources after adding them to the domain, you must register them on the server. Data sources must be registered on the server to bind their information to the JNDI

repository to allow applications deployed on the server to look them up. Data sources can be registered dynamically.

## Using the Console Tool

The **add-data-sources-to-server** command is used to dynamically add data sources to a server. For more information about using add-data-sources-to-server, refer to "add-data-sources-to-server" in *JEUS Reference Guide*.

The following is an example of registering ds1 and ds2 on server1 by executing add-data-sources-to-server.

```
[MASTER]domain1.adminServer>add-data-sources-to-server -server server1 -ids ds1,ds2
Successfully performed the ADD operation for data sources to the server [server1].
Check the results using "add-data-sources-to-server -server server1"
```

As shown in the previous result message, the result of registering ds1 and ds2 can be checked by executing '**add-data-sources-to-server -server server1**'.

```
[MASTER]domain1.adminServer>add-data-sources-to-server -server server1
Shows the current configuration.
Data sources registered in the server [server1].
=====
+-----+-----+
| data sources | ds1, ds2 |
+-----+-----+
=====
```

Execute **jndilist** to verify that ds1 and ds2 are bound to the JNDI repository of server1. Since a separate JNDI name has been entered for each data source, each data source ID is also used as the JNDI name.

```
[MASTER]domain1.adminServer>jndilist -server server1
The JNDI list on the server1
List of the context /
=====
+-----+-----+-----+
| Name | Value | Local Binding |
+-----+-----+-----+
| ds1 | jeus.jdbc.info.JDBCBindInfo | true |
| ds2 | jeus.jdbc.info.JDBCBindInfo | true |
| JEUSMQ_DLQ | jeus.jms.common.destination.JeusQueue | false |
| mgmt | jeus.jndi.JNSContext | false |
+-----+-----+-----+
=====
```

### 6.6.3. Removing a Data Source from a Server

A data source registered on a server can be removed dynamically. If a data source is removed from a server, the data source information is unbound from JNDI and connection pool, if exists, is destroyed.

#### Using the Console Tool

The **remove-data-sources-from-server** command is used to dynamically remove a data source from a server. For more information about using **remove-data-sources-from-server**, refer to "remove-data-sources-from-server" in *JEUS Reference Guide*.

The following is an example of removing ds2 from server1 by executing **remove-data-sources-from-server**.

```
[MASTER]domain1.adminServer>remove-data-sources-from-server -server server1 -ids ds2
Successfully performed the REMOVE operation for data sources from the server [server1].
Check the results using "remove-data-sources-from-server -server server1"
```

Execute **remove-data-sources-from-server-server server1** to verify that ds2 has been removed from server1 and only ds1 remains.

```
[MASTER]domain1.adminServer>remove-data-sources-from-server -server server1
Shows the current configuration.
Data sources registered in the server [server1].
=====
+-----+-----+
| data sources | ds1 |
+-----+-----+
=====
```

Execute **jndilist** to verify that ds2 is removed from the JNDI repository of server1 and only ds1 is still bound to JNDI.

```
[MASTER]domain1.adminServer>jndilist -server server1
The JNDI list on the server1
List of the context /
=====
+-----+-----+-----+
| Name | Value | Local Binding |
+-----+-----+-----+
| ds1 | jeus.jdbc.info.JDBCBindInfo | true |
| JEUSMQ_DLQ | jeus.jms.common.destination.JeusQueue | false |
| mgmt | jeus.jndi.JNSContext | false |
+-----+-----+-----+
=====
```

## 6.6.4. Registering a Data Source in a Cluster

Like registering a data source on a server, a data source can be registered in a cluster.

A data source registered in a cluster is valid on all servers in the cluster. Hence, a server in a cluster can use a data source registered in the cluster as if it is registered on the server. As a data source can be registered dynamically on the server, it can also be dynamically registered in a cluster.

### Using the Console Tool

The **add-data-sources-to-cluster** command is used to dynamically add data sources to a cluster. For detailed information about how to use add-data-sources-to-cluster, refer to "add-data-sources-to-cluster" in *JEUS Reference Guide*.

The following is an example of registering ds2 and ds3 to cluster1 by executing add-data-sources-to-cluster.

```
[MASTER]domain1.adminServer>add-data-sources-to-cluster -cluster cluster1 -ids ds2,ds3
Successfully performed the ADD operation for data sources to the cluster [cluster1].
Check the results using "add-data-sources-to-cluster -cluster cluster1"
```

As shown in the previous result message, the result of registering ds2 and ds3 to cluster1 can be checked by executing '**add-data-sources-to-cluster -cluster cluster1**'.

```
[MASTER]domain1.adminServer>add-datas-sources-to-cluster -cluster cluster1
Shows the current configuration.
The data sources registered in the cluster [cluster1].
=====
+-----+-----+
| data sources | ds2, ds3 |
+-----+-----+
=====
```

Now that data sources ds2 and ds3 are registered in cluster1, server2 and server3 in cluster1 can use ds2 and ds3. This means that ds2 and ds3 are JNDI bound to the JNDI repository of server2 and server3.

Execute **jndilist** to verify that ds2 and ds3 are bound to the JNDI repository of server2.

```
[MASTER]domain1.adminServer>jndilist -server server2
The JNDI list on the server2
List of the context /
=====
+-----+-----+-----+
| Name | Value | Local Binding |
+-----+-----+-----+
| ds2 | jeus.jdbc.info.JDBCBindInfo | true |
| ds3 | jeus.jdbc.info.JDBCBindInfo | true |
| JEUSMQ_DLQ | jeus.jms.common.destination.JeusQueue | false |
| mgmt | jeus.jndi.JNSContext | false |
```

```
+-----+-----+-----+
=====
```

Execute **jndilist** again to verify that ds2 and ds3 are bound to the JNDI repository of server3.

```
[MASTER]domain1.adminServer>jndilist -server server3
The JNDI list on the server3
List of the context /
=====
+-----+-----+-----+
|      Name      |      Value      | Local Binding |
+-----+-----+-----+
| ds2            | jeus.jdbc.info.JDBCBindInfo | true         |
| ds3            | jeus.jdbc.info.JDBCBindInfo | true         |
| JEUSMQ_DLQ     | jeus.jms.common.destination.JeusQueue | false        |
| mgmt           | jeus.jndi.JNSContext      | false        |
+-----+-----+-----+
=====
```

## 6.6.5. Removing Data Sources from a Cluster

As data sources can be dynamically removed from a server, they can also be dynamically removed from a cluster. If all data sources registered in a cluster are removed, then all servers in the cluster that were using those data sources cannot use them anymore. If a data source is removed from a cluster, the data source information is unbound from JNDI and connection pool, if exists, is also destroyed on each server.

### Using the Console Tool

The **remove-data-sources-from-cluster** command is used to dynamically remove data sources from a cluster. For more information about how to use **remove-data-sources-from-cluster**, refer to "remove-data-sources-from-cluster" in *JEUS Reference Guide*.

The following is an example of removing ds2 from cluster1 by executing **remove-data-sources-from-cluster**.

```
[MASTER]domain1.adminServer>remove-data-sources-from-cluster -cluster cluster1 -ids ds2
Successfully performed the REMOVE operation for data sources from the cluster [cluster1].
Check the results using "remove-data-sources-from-cluster -cluster cluster1"
```

Execute '**remove-data-sources-from-cluster -cluster cluster1**' to verify that ds2 has been removed from cluster1 and only ds3 remains.

```
[MASTER]domain1.adminServer>remove-data-sources-from-cluster -cluster cluster1
Shows the current configuration.
The data sources registered in the cluster [cluster1].
=====
+-----+-----+-----+
```

data sources	ds3	
+-----+	+-----+	+
=====	=====	=====

Now that ds2 is removed from cluster1, server2 and server3 in cluster1 can no longer use ds2. This means that ds2 is unbound from the repository of server2 and server3 and the connection pool for ds2, if exists, is also destroyed.

Execute **jndilist** to verify that ds2 is removed from the JNDI repository of server2.

```
[MASTER]domain1.adminServer>jndilist -server server2
The JNDI list on the server2
List of the context /
=====
+-----+-----+-----+
| Name | Value | Local Binding |
+-----+-----+-----+
| ds3 | jeus.jdbc.info.JDBCBindInfo | true |
| JEUSMQ_DLQ | jeus.jms.common.destination.JeusQueue | false |
| mgmt | jeus.jndi.JNSContext | false |
+-----+-----+-----+
=====
```

Execute **jndilist** to verify that ds2 is removed from the JNDI repository of server3.

```
[MASTER]domain1.adminServer>jndilist -server server3
The JNDI list on the server3
List of the context /
=====
+-----+-----+-----+
| Name | Value | Local Binding |
+-----+-----+-----+
| ds3 | jeus.jdbc.info.JDBCBindInfo | true |
| JEUSMQ_DLQ | jeus.jms.common.destination.JeusQueue | false |
| mgmt | jeus.jndi.JNSContext | false |
+-----+-----+-----+
=====
```

## 6.6.6. Adding a Server to a Cluster

A server can be dynamically added to a cluster. This involves modifying both the cluster and server configuration, and data source configuration can also be affected by this.

When a server is dynamically added to a cluster, cluster data sources registered in the cluster are valid on the newly added server. Therefore, a server added to the cluster can use the cluster data source registered to the cluster as if the cluster data source is registered on the server.

However, any data sources previously registered on the newly added server are unbound from JNDI and their connection pool, if exists, is also destroyed.

## Using the Console Tool

The **add-servers-to-cluster** command is used to dynamically add a server to a cluster. For more information about how to use add-servers-to-cluster, refer to "add-servers-to-cluster" in *JEUS Reference Guide*.

The following is an example of adding server1 to cluster1 by executing add-servers-to-cluster.

```
[MASTER]domain1.adminServer>add-servers-to-cluster cluster1 -servers server1
Successfully performed the ADD operation for The server list for cluster(cluster1)..
Check the results using "list-clusters cluster1 or add-servers-to-cluster cluster1"
```

As shown in the previous result message, the result of adding server1 to cluster1 can be checked by executing '**list-clusters cluster1**' or '**add-servers-to-cluster cluster1**'.

```
[MASTER]domain1.adminServer>add-servers-to-cluster cluster1
Shows the current configuration.
The server list for cluster(cluster1).
=====
+-----+-----+
| List of Servers | server2, server3, server1 |
+-----+-----+
=====
```

Now that server1 is added to cluster1, server1 can use ds3 registered in cluster1. This means that ds3 is JNDI bound to the JNDI repository of server1. Also, ds1, which was registered on server1, is still valid on server1.

When the JNDI monitoring information for server1 is checked, it can be seen that ds3 and ds1 are JNDI bound.

```
[MASTER]domain1.adminServer>jndilist -server server1
The JNDI list on the server1
List of the context /
=====
+-----+-----+-----+
| Name | Value | Local Binding |
+-----+-----+-----+
| ds1 | jeus.jdbc.info.JDBCBindInfo | true |
| ds3 | jeus.jdbc.info.JDBCBindInfo | true |
| JEUSMQ_DLQ | jeus.jms.common.destination.JeusQueue | false |
| mgmt | jeus.jndi.JNSContext | false |
+-----+-----+-----+
=====
```

### 6.6.7. Removing a Server from a Cluster

A server can be dynamically removed from a cluster. Same as in dynamically adding a server to a cluster, dynamically removing a server from a cluster also involves modifying both the cluster and



server configurations. This may also affect data source configurations.

When a server is dynamically removed from a cluster, the data source registered to the cluster is no longer valid on the server that is deleted from the cluster. The server removed from the cluster unbinds all data source information registered to the cluster and connection pool, if exists, is also destroyed.

However, any data sources that were previously registered on the server are JNDI bound and their connection pool, if exists, is recreated.

## Using the Console Tool

The **remove-servers-from-cluster** command is used to dynamically remove a server from a cluster. For more information about how to use remove-servers-from-cluster, refer to "remove-servers-from-cluster" in *JEUS Reference Guide*.

The following is an example of removing server3 from cluster1 by executing remove-servers-from-cluster.

```
[MASTER]domain1.adminServer>remove-servers-from-cluster cluster1 -servers server3
Successfully performed the REMOVE operation for The server list for cluster(cluster1)..
Check the results using "list-clusters cluster1 or remove-servers-from-cluster cluster1"
```

As shown in the previous result message, execute '**list-clusters cluster1**' or '**remove-servers-from-cluster cluster1**'. To verify that server3 has been removed from cluster1.

```
[MASTER]domain1.adminServer>remove-servers-from-cluster cluster1
Shows the current configuration.
The server list for cluster(cluster1).
=====
+-----+-----+
| List of Servers | server2, server1 |
+-----+-----+
=====
```

Now that server3 is removed from cluster1, server3 can no longer use ds3 registered to cluster1. This means that ds3 is unbound from the JNDI repository of server3 and the connection pool of ds3, if exists, is also destroyed.

Execute **jndilist** to verify that ds3 is removed from the JNDI repository of server3.

```
[MASTER]domain1.adminServer>jndilist -server server3
The JNDI list on the server3
List of the context /
=====
+-----+-----+-----+-----+
| Name | Value | Local Binding |
+-----+-----+-----+-----+
| JEUSMQ_DLQ | jeus.jms.common.destination.JeusQueue | false |
+-----+-----+-----+-----+
```

mgmt	jeus.jndi.JNSContext	false
------	----------------------	-------

## 6.6.8. Removing a Cluster

A cluster can be removed from a domain. This involves modifying both the domain and cluster configuration, and data source configuration can also be affected by this.

When a cluster is removed from the domain, all data sources registered to the cluster are no longer valid on the servers in the cluster. The servers in the removed cluster unbinds all the data source information registered to the cluster and their connection pool, if exists, is also destroyed. However, any data sources that were previously registered on the servers in the removed cluster are JNDI bound and their connection pool, if exists, is recreated.

### Using the Console Tool

The **remove-cluster** command is used to dynamically remove a cluster from the domain. For more information about how to use remove-cluster, refer to "remove-cluster" in *JEUS Reference Guide*.

The following is an example of removing cluster1 from domain1 by executing remove-cluster.

```
[MASTER]domain1.adminServer>remove-cluster cluster1
Successfully performed the REMOVE operation for cluster (cluster1).
Check the results using "list-clusters or remove-cluster"
```

As shown in the previous result message, execute '**list-clusters**' or '**remove-cluster**' to verify that cluster1 has been removed from domain1.

```
[MASTER]domain1.adminServer>remove-cluster
Shows the current configuration.
cluster list
=====
+-----+
| List of Clusters | empty |
+-----+
=====
```

Now that cluster1 is removed from domain1, server1 and server2 in cluster1 can no longer use ds3 registered in cluster1. This means that ds3 is unbound from the JNDI repository of server1 and server2 and the connection pool of ds3, if exists, is also destroyed.

Execute **jndilist** to verify that ds3 is unbound from the JNDI repository of server1 only ds1 remains bound.

```
[MASTER]domain1.adminServer>jndilist -server server1
The JNDI list on the server1
```

```
List of the context /
=====
+-----+-----+-----+
| Name | Value | Local Binding |
+-----+-----+-----+
| ds1 | jeus.jdbc.info.JDBCBindInfo | true |
| JEUSMQ_DLQ | jeus.jms.common.destination.JeusQueue | false |
| mgmt | jeus.jndi.JNSContext | false |
+-----+-----+-----+
=====
```

Execute **jndilist** to verify that no data source is bound to the JNDI repository of server2.

```
[MASTER]domain1.adminServer>jndilist -server server2
The JNDI list on the server2
List of the context /
=====
+-----+-----+-----+
| Name | Value | Local Binding |
+-----+-----+-----+
| JEUSMQ_DLQ | jeus.jms.common.destination.JeusQueue | false |
| mgmt | jeus.jndi.JNSContext | false |
+-----+-----+-----+
=====
```

## 6.6.9. Removing a Data Source

A data source can be dynamically removed from a domain. When a data source is removed from a domain, the deleted data source is no longer valid on the servers and cluster that were using it. The servers that were using the removed data source unbinds the data source and its connection pool, if exists, is also destroyed.

### Using the Console Tool

The **remove-data-source** command is used to dynamically remove a data source from the domain. For more information about remove-data-source refer to "remove-data-source" in *JEUS Reference Guide*.

The following is an example of removing ds1 from domain1 by executing remove-data-source.

```
[MASTER]domain1.adminServer>remove-data-source -id ds1
Successfully performed the REMOVE operation for data source [ds1] from the domain.
=====
+-----+-----+-----+
| resources.dataSource.database | MODIFY | ACTIVATED |
| servers.server.{? name == 'server1' }.dataSources | MODIFY | ACTIVATED |
+-----+-----+-----+
=====
Check the results using "remove-data-source"
```

The previous result message shows that executing `remove-data-source` to delete `ds1` has also modified `server1` to which `ds1` was registered. The result shows that `ds1` as well as its information on `server1` have been removed from `domain1`. When a dynamic configuration command is executed in this way, the result shows all related configuration changes that have occurred at runtime due to the command execution. For more information about how to use dynamic configuration commands, refer to "Changing Domain Configuration" in *JEUS Domain Guide*.

As shown in the previous result message, the result can be checked by executing `remove-data-source` without any options.

```
[MASTER]domain1.adminServer>remove-data-source
Shows the current configuration.
Data sources in domain
=====
+-----+-----+
| ds2 | common data source |
| ds3 | common data source |
+-----+-----+
```

Now that `ds1` is removed from `domain1`, `server1` can no longer use `ds1`. This means that `ds1` is unbound from the JNDI repository of `server1` and the connection pool of `ds1`, if exists, is also destroyed.

Execute `jndilist` and verify that `ds1` is unbound from the JNDI repository of `server1`.

```
[MASTER]domain1.adminServer>jndilist -server server1
The JNDI list on the server1
List of the context /
=====
+-----+-----+-----+-----+
| Name | Value | Local Binding |
+-----+-----+-----+-----+
| JEUSMQ_DLQ | jeus.jms.common.destination.JeusQueue | false |
| mgmt | jeus.jndi.JNSContext | false |
+-----+-----+-----+-----+
```

## 6.6.10. Modifying Data Source Configuration

So far, we saw how data sources are managed when a data source, server, or cluster is added or removed. This section describes how to modify the data source configuration.

As discussed in [Data Source Configuration](#), the data source configuration largely consists of the basic configurations needed for driver set up and the connection pool configuration. The driver set up configurations cannot be changed dynamically, but some basic configurations and most of the connection pool configurations can be changed dynamically. When static configurations are changed they are recorded in the configuration file but not applied to runtime. The server must be restarted to apply the changes to runtime. The dynamic configuration changes are also recorded in the

configuration file, and they are applied immediately to runtime.

There are various dynamic data source configurations, and we will look into examples of the minimum and maximum connection value of the connection pool which are expected to change dynamically frequently. The configurations are changed and verified using the data source ds2, and ds2 is registered on server1 and its connection pool is created. The method for registering data sources on the server have already been described. From here on, the method for creating the actual ds2 connection pool on server1 is described.

## Using the Console Tool

Connection Pool can be created by executing **create-connection-pool** in the console tool. For more detailed usage of create-connection-pool, refer to the *JEUS Reference Guide*. "create-connection-pool" in *JEUS Reference Guide*.

The following is an example of creating a connection pool for ds2 by executing create-connection-pool. The result message shows that the connection pool for ds2 has been created on server1.

```
[MASTER]domain1.adminServer>create-connection-pool -id ds2
Servers that successfully created a connection pool : server1
Servers that failed to create a connection pool : none.
```

The **connection-pool-info** command can be executed to check the minimum and maximum values of runtime connections in the connection pool. For more information about the command, refer to "connection-pool-info" in *JEUS Reference Guide*.

The following is an example of checking the minimum and maximum connection values of the ds2 runtime connection pool created on server1 by executing connection-pool-info.

```
[MASTER]domain1.adminServer>connection-pool-info -server server1
The connection pool information on the server [server1].
=====
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Connec | Min | Max | Active | Act | Active | Idle | Dispos | Tot | Wait | Enab |
| tion  |     |     | Max    | ive | Average|      | able   | al  |      | led  |
| Pool ID |     |     |        |     |        |      |        |    |     |     |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| ds2    | 2   | 30  | 0       | 0   | 0.0    | 2    | 0      | 2   | false | true |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
* : has not been created, total = active + idle + disposable
=====
```

As shown in the previous example, the minimum and maximum connection values of the ds2 runtime connection pool are configured to 2 and 30 respectively. These are the default values set when ds2 is initially added to the domain.

The following describes how to change the minimum and maximum connection values of ds2.

You can change the data source settings dynamically by executing **modify-data-source**. Execute **help** or refer to "modify-data-source" in JEUS Reference Guide to learn about all configurations that can be changed with modify-data-source. Dynamic configurations and related options are tagged to show which ones are dynamic configurations.

The following is an example of changing ds2's minimum and maximum connection values to 10 and 50 by executing modify-data-source.

```
[MASTER]domain1.adminServer>modify-data-source -id ds2 -min 10 -max 50
Successfully performed the MODIFY operation for configuration of the data source [ds2].
Check the results using "modify-data-source -id ds2"
```

As shown in the previous result message, the result can be checked by executing 'modify-data-source -id ds2'.

```
[MASTER]domain1.adminServer>modify-data-source -id ds2
Shows the current configuration.
configuration of the data source [ds2]
=====
+-----+-----+
| id          | ds2          |
| export-name | ds2          |
| data-source-class-name | oracle.jdbc.pool.OracleConnectionPoolDataSource |
| data-source-type | ConnectionPoolDataSource |
| server-name  | 192.168.1.165 |
| port-number  | 1521         |
| database-name | ora10g       |
| user         | jeustest1    |
| password     | jeustest1    |
| login-timeout | 0            |
| auto-commit  | DRIVER       |
| stmt-query-timeout | 0           |
| pool-destroy-timeout | 10000       |
| property     | driverType;java.lang.String;thin |
| support-xa-emulation | false       |
| min          | 10           |
| max          | 50           |
| step         | 1            |
| period       | 3600000      |
| enable-wait  | false        |
| wait-time    | 10000        |
| max-use-count | 0            |
| dbaTimeout   | -1           |
| stmt-caching-size | -1          |
| stmt-fetch-size | -1           |
| connection-trace | false       |
| get-connection-trace | true        |
| auto-commit-trace | false       |
| use-sql-trace | false        |
| keep-connection-handle-open | false       |
+-----+-----+
=====
```

The following is an example of checking the minimum and maximum connection values of the ds2 runtime connection pool created on server1 by executing **connection-pool-info**. The minimum and maximum connection values of ds2 runtime have been changed to 10 and 50 respectively.

```
[MASTER]domain1.adminServer>connection-pool-info -server server1
The connection pool information on the server [server1].
=====
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Connec | Min | Max | Active | Act | Active | Idle | Dispos | Tot | Wait | Enab |
| tion   |     |     | Max    | ive | Average|      | able   | al  |      | led  |
| Pool ID|     |     |        |     |        |      |        |    |     |     |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| ds2    | 10 | 50 |      0 | 0   | 0.0   | 10  |      0 | 10 | false | true |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
* : has not been created, total = active + idle + disposable
=====
```

## 6.6.11. Checking Data Source Configuration

In JEUS 7 and later versions, it is not recommended to manually edit the configuration settings in the configuration file. To check the settings, it is recommended to use the data source configuration commands instead of directly accessing the configuration file.

This section describes how to check the data source configurations using the console tool.

### Using the Console Tool

The **list-data-sources** command is used to check the list of all data sources that exist in the domain.

The following is an example of checking the data sources in the domain by executing list-data-sources.

```
[MASTER]domain1.adminServer>list-data-sources
The list of data sources
=====
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Data Source ID | JNDI Export Name | Data Source Type |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| ds2            | ds2              | ConnectionPoolDataSource |
| ds3            | ds3              | ConnectionPoolDataSource |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
=====
```

If a data source ID is given as an option, data source configuration details are displayed.

The following is an example of checking the detailed configuration of ds2 by executing list-data-sources.

```
[MASTER]domain1.adminServer>list-data-sources -id ds2
```

```
The configuration of the data source [ds2]
```

Configuration name	Configuration value
id	ds2
export-name	ds2
data-source-class-name	oracle.jdbc.pool.OracleConnectionPoolDataSource
data-source-type	ConnectionPoolDataSource
server-name	192.168.1.165
port-number	1521
database-name	ora10g
user	jeustest1
password	jeustest1
login-timeout	0
auto-commit	DRIVER
stmt-query-timeout	0
pool-destroy-timeout	10000
property	[driverType;java.lang.String;thin]
support-xa-emulation	false
min	10
max	50
step	1
period	3600000
enable-wait	false
wait-time	10000
max-use-count	0
dbaTimeout	-1
stmt-caching-size	-1
stmt-fetch-size	-1
connection-trace	false
get-connection-trace	true
auto-commit-trace	false
use-sql-trace	false
keep-connection-handle-open	false

## 6.7. Dynamically Changing Cluster Data Source Configuration

By using examples, this section describes how to dynamically change the configurations related to cluster data sources. For better understanding, knowledge about JEUS domain structure and data source management structure in the domain are required. Knowledge about using each configuration item of the cluster data source is also required.

For information on the JEUS domain structure, refer to "Domain Components" in *JEUS Domain Guide*. For information about the data source management structure in the domain and how to use each configuration item, refer to [Management of Data Sources and Connection Pools](#) and [Cluster Data Source Configuration](#), respectively.



Dynamic changes can be applied using the console tool. Both methods are described in this section. Abbreviated commands and option names are used for describing the console tool method. For more information about how to use abbreviated commands and option names, execute **help** in the console tool or refer to "Connection Pool Control and Monitoring Commands" in *JEUS Reference Guide*.

First perform some preparation tasks to help with the explanation.

First, create a domain named **domain1** and a Master named **adminServer**. After starting the Master, add and start three MSs named **server1**, **server2**, and **server3** to **domain1**. Then, group **server2** and **server3** into a cluster named **cluster1**. For information on how to set up domains, clusters, and servers, refer to "Creating a Domain" and "JEUS Clustering", in *JEUS Domain Guide*, and "JEUS Configuration" in *JEUS Server Guide*. From here on, it is assumed that the aforementioned preparation tasks have been completed.



Since all examples are related to each other, the contents of this section should be read in order.

### 6.7.1. Adding a Cluster Data Source

Like other data sources, a cluster data source can also be added dynamically. Always keep in mind that when a cluster data source is added, the component data sources in the cluster data source must also be added.

This section describes an example of adding a cluster data source made up of data sources **ds1** and **ds2** to **domain1**. The cluster data source ID is **cds1**.

#### Using the Console Tool

The data sources **ds1** and **ds2** that are the component data source of **cds1** must be added to **domain1**.

The following is an example of adding **ds1** and **ds2** to **domain1** by performing **add-data-source**. For more detailed usage of **add-data-source**, "add-data-source" in *JEUS Reference Guide*.

```
[MASTER]domain1.adminServer>add-data-source -id ds1 -dst ConnectionPoolDataSource -dscn
oracle.jdbc.pool.OracleConnectionPoolDataSource -sn 192.168.1.165 -pn 1521 -dn ora10g -user jeustest1
-password jeustest1 -property driverType;java.lang.String;thin
Successfully performed the ADD operation for data source [ds1] to domain.
Check the results using "add-data-source"

[MASTER]domain1.adminServer>add-data-source -id ds2 -dst ConnectionPoolDataSource -dscn
oracle.jdbc.pool.OracleConnectionPoolDataSource -sn 192.168.1.165 -pn 1521 -dn ora10g -user jeustest1
-password jeustest1 -property driverType;java.lang.String;thin
Successfully performed the ADD operation for data source [ds2] to domain.
Check the results using "add-data-source"
```

Now that we have added **ds1** and **ds2** to **domain1**, let's add **cds1** to **domain1**. You can dynamically

add a cluster data source to a domain by performing **add-cluster-data-source**. For more detailed usage of add-cluster-data-source, see "add-cluster-data-source" in *JEUS Reference Guide*. To add data sources using add-cluster-data-source, you need to specify the respective option for each detailed configuration of cluster data sources.

The following is an example of entering some required configurations by executing **add-cluster-data-source** and adding the cluster data source cds1 to domain1.

```
[MASTER]domain1.adminServer>add-cluster-data-source -id cds1 -cds ds1,ds2
Successfully performed the ADD operation for cluster data source [cds1] to domain.
Check the results using "add-cluster-data-source"
```

As shown in the previous result message, the result of adding cds1 can be checked by executing **add-cluster-data-source** without any options.

```
[MASTER]domain1.adminServer>add-cluster-data-source
Shows the current configuration.
Data sources in domain
=====
+-----+-----+
| ds1 | common data source |
| ds2 | common data source |
| cds1 | cluster data source |
+-----+-----+
=====
```

## 6.7.2. Registering a Cluster Data Source to a Server

To use a cluster data source, it must be registered to a server like other data sources. Only when a cluster data source is registered on a server, the cluster data source information is bound to the JNDI repository of the server. After this, deployed applications can look up the cluster data source, but the component data sources grouped into the cluster data source must be registered on the server.

Each component data source provides the connection pool service, and their connection pool is created on the server. To use the cluster data sources as desired, the component data sources in the cluster data source must be registered on the server where the cluster data source is registered on.

This section describes how to register cds1 to server1.

### Using the Console Tool

The component data sources ds1 and ds2 of cds1 must also be registered on server1.

The following is an example of registering ds1, ds2, and cds1 on server1 by executing **add-data-sources-to-server**. For more information about how to use add-data-sources-to-server, refer to "add-data-sources-to-server" in *JEUS Reference Guide*.

```
[MASTER]domain1.adminServer>add-data-sources-to-server -server server1 -ids ds1,ds2,cds1
Successfully performed the ADD operation for data sources to the server [server1].
Check the results using "add-data-sources-to-server -server server1"
```

As shown in the previous result message, the result of registering ds1,ds2, and cds2 can be checked by executing '**add-data-sources-to-server -server server1**'.

```
[MASTER]domain1.adminServer>add-data-sources-to-server -server server1
Shows the current configuration.
Data sources registered in the server [server1].
=====
+-----+-----+
| data sources | ds1, ds2, cds1 |
+-----+-----+
=====
```

Execute **jndilist** to verify that ds1, ds2, and cds1 are bound to the JNDI repository of server1.

```
[MASTER]domain1.adminServer>jndilist -server server1
The JNDI list on the server1
List of the context /
=====
+-----+-----+-----+-----+
| Name | Value | Local Binding |
+-----+-----+-----+-----+
| cds1 | jeus.jdbc.info.JDBCBindInfo | true |
| ds1 | jeus.jdbc.info.JDBCBindInfo | true |
| ds2 | jeus.jdbc.info.JDBCBindInfo | true |
| JEUSMQ_DLQ | jeus.jms.common.destination.JeusQueue | false |
| mgmt | jeus.jndi.JNSContext | false |
+-----+-----+-----+-----+
=====
```

### 6.7.3. Removing a Cluster Data Source from the Server

Cluster data sources registered on the server can also be dynamically removed from the server like other data sources.

Like other data sources, when a cluster data source is removed from the server, the cluster data source information is unbound from JNDI. Cluster data source does not have its own connection pool, and the actual connection pool is created in the component data sources in the cluster data source. Therefore, when a cluster data source is removed from a server, unlike other data sources, the 'destroy' command does not execute for the connection pool. Although it may be assumed that the connection pool of the component data sources in the cluster data source will be destroyed, since each component data source can function as an independent data source, it is neither unbound from JNDI nor destroyed.

## Using the Console Tool

The **remove-data-sources-from-server** command is used to dynamically remove a cluster data source from the server. For more information about how to use remove-data-sources-from-server, refer to "remove-data-sources-from-server" in *JEUS Reference Guide*.

The following is an example of removing cds1 from server1 by executing remove-data-sources-from-server.

```
[MASTER]domain1.adminServer>remove-data-sources-from-server -server server1 -ids cds1
Successfully performed the REMOVE operation for data sources from the server [server1].
Check the results using "remove-data-sources-from-server -server server1"
```

As shown in the previous result message, the result of removing cds1 from server1 can be checked by executing '**remove-data-sources-from-server -server server1**'.

```
[MASTER]domain1.adminServer>remove-data-sources-from-server -server server1
Shows the current configuration.
Data sources registered in the server [server1].
=====
+-----+-----+
| data sources | ds1, ds2 |
+-----+-----+
=====
```

It shows that cds1 has been removed, and ds1 and ds2 still remain registered.

Execute **jndilist** to verify that cds1 has been unbound from the JNDI repository of server1.

```
[MASTER]domain1.adminServer>jndilist -server server1
The JNDI list on the server1
List of the context /
=====
+-----+-----+-----+-----+
| Name | Value | Local Binding |
+-----+-----+-----+-----+
| ds1 | jeus.jdbc.info.JDBCBindInfo | true |
| ds2 | jeus.jdbc.info.JDBCBindInfo | true |
| JEUSMQ_DLQ | jeus.jms.common.destination.JeusQueue | false |
| mgmt | jeus.jndi.JNSContext | false |
+-----+-----+-----+-----+
=====
```

### 6.7.4. Registering a Data Source in a Cluster

Cluster data sources can be dynamically registered to a cluster like other data sources. Cluster data sources registered to a cluster are valid on all servers in the cluster. Therefore, servers in a cluster can use the cluster data source registered on the cluster as if it was registered on each server. Like registering a cluster data source to a server, the component data sources in the cluster data source

must be registered with the cluster data source to a cluster.

This section describes the process of registering cds1 to cluster1.

## Using the Console Tool

The component data sources ds1 and ds2 of cds1 must also be registered to cluster1.

The following is an example of registering ds1, ds2, and cds1 to cluster1 by executing **add-data-sources-to-cluster** in the console tool. For more information about how to use add-data-sources-to-cluster, refer to "add-data-sources-to-cluster" in *JEUS Reference Guide*.

```
[MASTER]domain1.adminServer>add-data-sources-to-cluster -cluster cluster1 -ids ds1,ds2,cds1
Successfully performed the ADD operation for data sources to the cluster [cluster1].
Check the results using "add-data-sources-to-cluster -cluster cluster1"
```

As shown in the previous result message, the result of registering ds1,ds2, and cds2 can be checked by executing '**add-data-sources-to-cluster -cluster cluster1**'.

```
[MASTER]domain1.adminServer>add-data-sources-to-cluster -cluster cluster1
Shows the current configuration.
The data sources registered in the cluster [cluster1].
=====
+-----+-----+
| data sources | ds1, ds2, cds1 |
+-----+-----+
=====
```

Now that ds1, ds2, and cds1 are registered to cluster1, they can be used by server1 and server2 in cluster1.

Execute **jndilist** to verify that ds1, ds2, and cds1 are bound to the JNDI repository of server1.

```
[MASTER]domain1.adminServer>jndilist -server server2
The JNDI list on the server2
List of the context /
=====
+-----+-----+-----+
| Name | Value | Local Binding |
+-----+-----+-----+
| cds1 | jeus.jdbc.info.JDBCBindInfo | true |
| ds1 | jeus.jdbc.info.JDBCBindInfo | true |
| ds2 | jeus.jdbc.info.JDBCBindInfo | true |
| JEUSMQ_DLQ | jeus.jms.common.destination.JeusQueue | false |
| mgmt | jeus.jndi.JNSContext | false |
+-----+-----+-----+
=====
```

## 6.7.5. Removing Data Sources from a Cluster

Cluster data sources registered to a cluster can also be dynamically removed from the cluster like other data sources. When a cluster data source registered to a cluster is removed, all the servers in the cluster that were using it cannot use the cluster data source anymore. Therefore, when a cluster data source is removed from the cluster, the servers in the cluster unbind the cluster data source information from JNDI. However, as when cluster data sources are removed from the server, "connection pool destroy" does not execute on the component data sources.

### Using the Console Tool

The **remove-data-sources-from-cluster** command is used to dynamically remove data sources from a cluster. For more information about how to use remove-data-sources-from-cluster, refer to "remove-data-sources-from-cluster" in *JEUS Reference Guide*.

The following is an example of removing cds1 from cluster1 by executing remove-data-sources-from-cluster.

```
[MASTER]domain1.adminServer>remove-data-sources-from-cluster -cluster cluster1 -ids cds1
Successfully performed the REMOVE operation for data sources from the cluster [cluster1].
Check the results using "remove-data-sources-from-cluster -cluster cluster1"
```

As shown in the previous result message, the result of removing cds1 from cluster1 can be checked by executing '**remove-data-sources-from-cluster -cluster cluster1**'.

```
[MASTER]domain1.adminServer>remove-data-sources-from-cluster -cluster cluster1
Shows the current configuration.
The data sources registered in the cluster [cluster1].
=====
+-----+-----+
| data sources | ds1, ds2 |
+-----+-----+
=====
```

Now that cluster data source cds1 is removed from cluster1, server2 and server3 in cluster1 can no longer use cds1. This means that cds1 is unbound from the JNDI repository of server2 and server3.

Execute **jndilist** to verify that cds1 has been unbound from the JNDI repository of server2.

```
[MASTER]domain1.adminServer>jndilist -server server2
The JNDI list on the server2
List of the context /
=====
+-----+-----+-----+-----+
| Name | Value | Local Binding |
+-----+-----+-----+-----+
| ds1 | jeus.jdbc.info.JDBCBindInfo | true |
| ds2 | jeus.jdbc.info.JDBCBindInfo | true |
| JEUSMQ_DLQ | jeus.jms.common.destination.JeusQueue | false |
```

mgmt	jeus.jndi.JNSContext	false	
+-----+	+-----+	+-----+	+
=====			

### 6.7.6. Adding a Server to a Cluster

When a server is dynamically added to a cluster, like with other data sources registered in the cluster, cluster data sources registered in the cluster are valid on the newly added server. Therefore, a server added to the cluster can use the cluster data source registered to the cluster as if the cluster data source is registered on the server.

Since the overall mechanism is similar to that described in [Adding a Server to a Cluster](#), description has been omitted in this section. For more information, refer to each applicable sections.

### 6.7.7. Removing a Server from a Cluster

When a server is dynamically removed from a cluster, like with other data sources registered to the cluster, the cluster data sources registered to the cluster are no longer valid on the server. Hence, the server removed from the cluster unbinds all cluster data source information registered to the cluster.

Since the overall mechanism is similar to that described in [Removing a Server from a Cluster](#), description has been omitted in this section. For more information, refer to [Removing a Server from a Cluster](#).

### 6.7.8. Removing a Cluster

When a cluster is dynamically removed from the domain, like with other cluster data sources registered to the cluster, the cluster data sources registered to the cluster are no longer valid on the servers in the cluster. Therefore, the servers that belonged to the removed cluster unbinds all cluster data source information registered to the cluster.

Since the overall mechanism is similar to that described in [Removing a Cluster](#), description has been omitted in this section. For more information, refer to [Removing a Cluster](#).

### 6.7.9. Removing a Cluster Data Source

Cluster data sources can be dynamically deleted from the domain like other data sources. When a cluster data source is deleted from the domain, it is no longer valid on the server and cluster that were using it. Hence, the servers that were using the deleted cluster data source unbind the cluster data source information from JNDI. To check how the cluster data source registered on the server and cluster is processed when removing a cluster data source, cds1 is registered on server1 and cluster1. From here on, it is assumed that cds1 is registered on server1 and cluster1.

This section describes the process of removing cds1 from domain1.

## Using the Console Tool

The **remove-cluster-data-source** command is used to dynamically delete a cluster data source from the domain. For more information about remove-cluster-data-source, refer to "remove-cluster-data-source" in *JEUS Reference Guide*.

The following is an example of removing cds1 from domain1 by executing remove-cluster-data-source.

```
[MASTER]domain1.adminServer>remove-cluster-data-source -id cds1
Successfully performed the REMOVE operation for cluster data source [cds1] from domain.
=====
+-----+-----+-----+
| resources.dataSource.clusterDs          | MODIFY | ACTIVATED |
| servers.server.{? name == 'server1' }.dataSources | MODIFY | ACTIVATED |
| clusters.cluster.{? name == 'cluster1' }.dataSources | MODIFY | ACTIVATED |
+-----+-----+-----+
=====
Check the results using "remove-cluster-data-source"
```

As shown in the previous result message, removing cds1 by executing remove-cluster-data-source also affects server1 and cluster1 where cds1 is registered to. The result displays that when cds1 is removed from domain1, the cds1 information registered on server1 and cluster1 are also deleted.

When executing a dynamic configuration command, you can check for all other runtime configuration changes that occurred from it. For more information about how to use the dynamic configuration commands, refer to "Changing Domain Configuration" in *JEUS Domain Guide*.

As shown in the previous result message, execute **remove-cluster-data-source** to verify that cds1 has been removed.

```
[MASTER]domain1.adminServer>remove-cluster-data-source
Shows the current configuration.
Data sources in domain
=====
+-----+-----+-----+
| ds1 | common data source |
| ds2 | common data source |
+-----+-----+-----+
=====
```

Now that cds1 is deleted from domain1, server1, where cds1 is registered and used, and server2 and server3, which were able to use cds1 that is registered to cluster1, cannot use cds1 anymore. This means that cds1 is unbound from the repository of server1, server2, and server3.

Execute **jndilist** to verify that cds1 has been unbound from the JNDI repository of server1.



```
[MASTER]domain1.adminServer>jndilist -server server1
The JNDI list on the server1
List of the context /
=====
+-----+-----+-----+
| Name | Value | Local Binding |
+-----+-----+-----+
| ds1 | jeus.jdbc.info.JDBCBindInfo | true |
| ds2 | jeus.jdbc.info.JDBCBindInfo | true |
| JEUSMQ_DLQ | jeus.jms.common.destination.JeusQueue | false |
| mgmt | jeus.jndi.JNSContext | false |
+-----+-----+-----+
=====
```

Execute jndilist to verify that cds is unbound from JNDI repository of server2.

```
[MASTER]domain1.adminServer>jndilist -server server2
The JNDI list on the server2
List of the context /
=====
+-----+-----+-----+
| Name | Value | Local Binding |
+-----+-----+-----+
| ds1 | jeus.jdbc.info.JDBCBindInfo | true |
| ds2 | jeus.jdbc.info.JDBCBindInfo | true |
| JEUSMQ_DLQ | jeus.jms.common.destination.JeusQueue | false |
| mgmt | jeus.jndi.JNSContext | false |
+-----+-----+-----+
=====
```

## 6.7.10. Changing Cluster Data Source Configuration

Up to this point, we examined how data sources are managed when cluster data sources, servers, and clusters are added/deleted. This section will show how to change the configuration of a cluster data source. It is possible to dynamically change the configurations of a cluster data source. We will look into an example of changing the component data source of a cluster data source.

Since currently there is no cluster data source in the domain, first add cds1 with ds1 and ds2 as the component data source1 to domain1 by executing add-cluster-data-source. From here on, it is assumed that cds1, which has ds1 and ds2 as its component data sources, has been added to domain1.

This section describes the process of removing ds2 from cds1.

### Using the Console Tool

The **modify-cluster-data-source** command is used to change the cluster data source configuration. Use the **help** command or refer to "modify-cluster-data-source" in JEUS Reference Guide to learn about all configurations that can be changed using modify-cluster-data-source. Dynamic

configurations and related options are tagged to show which ones are dynamic configurations.

The following is an example of removing ds2 from the component data source list of cds1 by executing `modify-cluster-data-source`.

```
[MASTER]domain1.adminServer>modify-cluster-data-source -id cds1 -cds ds1
Successfully performed the MODIFY operation for configuration of the cluster data source [cds1].
Check the results using "modify-cluster-data-source -id cds1"
```

The result of removing ds2 from the cds1 component data source list can be verified by executing **'modify-cluster-data-source-id cds1'** with other option values.

```
[MASTER]domain1.adminServer>modify-cluster-data-source -id cds1
Shows the current configuration.
configuration of the cluster data source [cds1]
=====
+-----+-----+
| id                | cds1 |
| export-name       | cds1 |
| load-balance      | false|
| is-pre-conn       | false|
| use-failback      | true |
| component-data-sources | ds1  |
+-----+-----+
=====
```

## 6.7.11. Checking the Cluster Data Source Configuration

In JEUS 7 and later versions, it is not recommended to manually edit the configuration settings in the configuration file. To check the settings, it is recommended to use the data source configuration commands instead of directly accessing the configuration file.

This section describes how to check the cluster data source configurations using the console tool.

### Using the Console Tool

The **list-cluster-data-sources** command is used to view the list of all cluster data sources in the domain.

The following is an example of checking the list of all cluster data sources in the domain by executing `list-cluster-data-sources`.

```
[MASTER]domain1.adminServer>list-cluster-data-sources
The list of cluster data sources
=====
+-----+-----+-----+
| Data Source ID | JNDI Export Name | Component Data Sources |
+-----+-----+-----+
| cds1          | cds1             | [ds1]                  |
+-----+-----+-----+
```

```
+-----+-----+-----+
=====
```

If a cluster data source ID is given as an option, cluster data source configuration details are displayed. The following is an example of checking the configuration details of cds1 by executing list-cluster-data-sources.

```
[MASTER]domain1.adminServer>list-cluster-data-sources -id cds1
The configuration of cluster data source [cds1]
=====
+-----+-----+
| Configuration Name | Configuration Value |
+-----+-----+
| id                 | cds1                 |
| export-name        | cds1                 |
| load-balance        | false                |
| is-pre-conn        | false                |
| use-failback        | true                 |
| component-data-sources | [ds1]                |
+-----+-----+
=====
```

## 6.8. Monitoring JDBC Connection Pool

This section provides an example of monitoring a connection pool on a server. To fully understand this, familiarity with the JEUS domain structure and the data source and connection pool management within the domain is recommended. For the JEUS domain structure, refer to "Domain Components" in *JEUS Domain Guide*. For further details on data sources and connection pool management in the domain, see [Management of Data Sources and Connection Pools](#).

The JDBC connection pool can be monitored by using the console tool. Both methods are described in this section. Abbreviated commands and option names are used for describing the console tool method. For information about abbreviated commands, option names, and command usage descriptions, execute **help** with the command name as an argument, or refer to "Connection Pool Control and Monitoring Commands" in *JEUS Reference Guide*.

First perform some preparation tasks to help with the explanation. Create a domain named domain1 and a MASTER named adminServer. After starting MASTER, add two MSs named server1 and server2 to domain1 and start them. Then, add the data sources ds1 and ds2 to domain1. Register both ds1 and ds2 on server1, and only ds1 on server2. Lastly, create a connection pool for ds1 on both server1 and server2.

For more information about domain and server configuration, refer to "Creating a Domain" in *JEUS Domain Guide* and "JEUS Configuration" in *JEUS Server Guide*. For explanation on data source configuration, see [Dynamic Data Source Configuration](#). For information about how to create a connection pool, see [Controlling JDBC Connection Pool](#). From here on, it is assumed that the aforementioned preparation tasks have been completed.



The server is the final target where data sources are registered on. The data sources registered to the cluster as well as directly on the server affect the server. A cluster is just a logical group of servers, and the server is the one that actually uses a data source to create and use a connection pool. For this reason, the server is the monitoring unit for the connection pool.

## 6.8.1. Checking a JDBC Connection Pool List

This section describes how to check the JDBC connection pool list by using the console tool.

### Using Console Tool

The **connection-pool-info** command is used to check for the list of all connection pools on a server. Since the connection-pool-info command provides the same function for JDBC connection pools as well as JCA connection pools, the -jdbc option must be provided when using a JDBC connection pool. For more information about using connection-pool-info, refer to "connection-pool-info" in *JEUS Reference Guide*.

The following is an example of checking the list of all JDBC connection pools on server1 by executing connection-pool-info.

```
[MASTER]domain1.adminServer>connection-pool-info -server server1 -jdbc
The connection pool information on the server [server1].
=====
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| Connec | Min | Max | Active | Act | Active | Idle | Dispos | Tot | Wait | Enab |
| tion   |     |     | Max    | ive | Average|      | able   | al  |      | led  |
| Pool ID|     |     |        |     |        |      |        |    |     |     |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| ds1    | 2   | 30  | 0       | 0   | 0.0    | 2   | 0      | 2   | fal  | true |
|         |     |     |         |     |        |     |        |     | se   |      |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| ds2 *  | 2   | 30  | 0       | 0   | 0.0    | 0   | 0      | 0   | fal  | false|
|         |     |     |         |     |        |     |        |     | se   |      |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+

* : has not been created, total = active + idle + disposable
=====
```

Since both ds1 and ds2 are registered on server1, the connection pool information for both ds1 and ds2 exist. The connection pool for ds1 has already been created as a preparation task, but ds2 connection pool has not been created yet.

The columns consist of the connection pool ID, minimum number of connections, maximum number of connections, active connection count, idle connection count, disposable connection count, total number of connections, whether to wait when there are no available connections, and whether the connection pool has been enabled.

To only check for the currently created connection pools, use the 'active' option as in the following.

```
[MASTER]domain1.adminServer>connection-pool-info -server server1 -active -jdbc
The connection pool information on the server [server1].
```

Connec tion Pool ID	Min	Max	Active Max	Act ive	Active Average	Idle	Dispos able	Tot al	Wait	Enab led
ds1	2	30	0	0	0.0	2	0	2	fal se	true

\* : has not been created, total = active + idle + disposable

Use the JNDI option to also display the JNDI name of the data source.

```
[MASTER]domain1.adminServer>connection-pool-info -server server1 -jndi -jdbc
The connection pool information on the server [server1].
```

Connection Pool ID	JNDI Expo rt Name	Min	Max	Act ive Max	Act ive	Act ive Aver age	Idle	Dis pos able	Tot al	Wait	Ena bled
ds1	ds1	2	30	0	0	0.0	2	0	2	fal se	true
ds2 *	ds2	2	30	0	0	0.0	0	0	0	fal se	fal se

\* : has not been created, total = active + idle + disposable

The following shows the list of all connection pools that exist on server2.

```
[MASTER]domain1.adminServer>connection-pool-info -server server2 -jdbc
The connection pool information on the server [server2].
```

Connec tion Pool ID	Min	Max	Active Max	Act ive	Active Average	Idle	Dispos able	Tot al	Wait	Enab led
ds1	2	30	0	0	0.0	0	0	0	fal se	fal se

\* : has not been created, total = active + idle + disposable

Since only ds1 is registered on server2, only the connection pool for ds1 exists.

## 6.8.2. Checking JDBC Connection Pool Information

This section describes how to check detailed information of a specific JDBC connection pool by using the console tool.

### Using Console Tool

The **connection-pool-info** command is used to check for the details of a connection pool on the server. Since the connection-pool-info command provides the same function for JDBC connection pools as well as JCA connection pools, the **-jdbc** option must be provided when using a JDBC connection pool. For more information about using connection-pool-info, refer to "connection-pool-info" in *JEUS Reference Guide*.

The following is an example of checking details of connection pool for ds1 on server1 by executing connection-pool-info.

```
[MASTER]domain1.adminServer>connection-pool-info -server server1 -id ds1 -jdbc
Information about connections in the server [server1]'s connection pool [ds1].
=====
+-----+-----+-----+-----+-----+-----+
| Connection ID | State | State Time(sec) | Use Count | Type |
+-----+-----+-----+-----+-----+
| ds1-1         | idle  | 965.837         | 0         | pooled |
| ds1-2         | idle  | 965.806         | 0         | pooled |
+-----+-----+-----+-----+-----+
=====
```

It shows two unused connections that have been idle for about 965 seconds, and both are not disposable connections.

Use the **-t** option as in the following example to check for the name of the thread that owns the connection.

```
[MASTER]domain1.adminServer>connection-pool-info -server server1 -id ds1 -t -jdbc
Information about connections in the server [server1]'s connection pool [ds1].
=====
+-----+-----+-----+-----+-----+-----+
| Connection ID | State | State Time(sec) | Use Count | Type | Thread name |
+-----+-----+-----+-----+-----+-----+
| ds1-1         | active | 1105.954        | 1         | pooled | http-w1     |
| ds1-2         | idle  | 1105.923        | 0         | pooled |             |
+-----+-----+-----+-----+-----+-----+
=====
```

## 6.9. Controlling JDBC Connection Pool

This section provides an example of controlling a connection pool on a server. To fully understand this, familiarity with the JEUS domain structure and the data source and connection pool management within the domain is recommended. For more information about the JEUS domain structure, refer to "Domain Components" in *JEUS Domain Guide*. For further details on data sources and connection pool management in the domain, see [Management of Data Sources and Connection Pools](#).

The JDBC connection pool can be controlled by using the console tool. This section describes both methods.

First perform some preparation tasks to help with the explanation. Create a domain named domain1 and a MASTER named adminServer. After starting MASTER, add two MSs named server1 and server2 to domain1 and start them. Then, add the data sources ds1 and ds2 to domain1. Lastly, register both ds1 and ds2 on server1 and server2. For more information about how to configure domains and servers, refer to "Creating a Domain" in *JEUS Domain Guide* and "JEUS Configuration" in *JEUS Server Guide*. For details on data source configurations, see [Dynamic Data Source Configuration](#).



For examples that show the result of controlling a connection pool, refer to [Monitoring JDBC Connection Pool](#).

### 6.9.1. Creating a Connection Pool

This section describes how to create a connection pool by using the console tool.

#### Using the Console Tool

The **create-connection-pool** command is used to create a connection pool for data sources on the server. For more information about using create-connection-pool, refer to "create-connection-pool" in *JEUS Reference Guide*.

The following is an example of creating connection pool for ds1 on server1 by executing create-connection-pool.

```
[MASTER]domain1.adminServer>create-connection-pool -server server1 -id ds1
Servers that successfully created a connection pool : server1
Servers that failed to create a connection pool : none.
```

If the server option value is not provided, then a connection pool is created for all the servers where the data source is registered on.

The following is an example of executing create-connection-pool to create connection pool for ds1 on server1 and server2 where ds2 is registered on.

```
[MASTER]domain1.adminServer>create-connection-pool -id ds2
Servers that successfully created a connection pool : server1, server2
Servers that failed to create a connection pool : none.
```

A JDBC connection pool can be created even if control-connection-pool is used. For more information about using control-connection-pool, refer to "control-connection-pool" in *JEUS Reference Guide*.

The following is an example of creating a connection pool for ds1 on server1 by executing control-connection-pool.

```
[MASTER]domain1.adminServer>control-connection-pool -server server1 -id ds1 -create
Servers that successfully created a connection pool : server1
Servers that failed to create a connection pool : none.
```

## 6.9.2. Disabling a Connection Pool

This section describes how to disable a connection pool by using the console tool.

### Using the Console Tool

The **disable-connection-pool** command is used to disable the connection pool on the server. For more information about how to use disable-connection-pool, refer to "disable-connection-pool" in *JEUS Reference Guide*.

The following is an example of disabling the connection pool for ds1 on server1 by executing disable-connection-pool.

```
[MASTER]domain1.adminServer>disable-connection-pool -server server1 -id ds1
Servers that successfully disabled a connection pool : server1
Servers that failed to disable a connection pool : none.
```

If the server option is not provided, then the connection pool will be disabled for all servers where the data source is registered on.

The following is an example of disabling the connection pool for ds2 on server1 and server2 where ds2 is registered on by executing disable-connection-pool.

```
[MASTER]domain1.adminServer>disable-connection-pool -id ds2
Servers that successfully disabled a connection pool : server1, server2
Servers that failed to disable a connection pool : none.
```

JDBC connection pool can also be disabled by executing control-connection-pool. For more information about using control-connection-pool, refer to "control-connection-pool" in *JEUS Reference Guide*.



The following is an example of disabling the ds1 connection pool created on server1 by executing control-connection-pool.

```
[MASTER]domain1.adminServer>control-connection-pool -server server1 -id ds1 -disable
Servers that successfully disabled a connection pool : server1
Servers that failed to disable a connection pool : none.
```

### 6.9.3. Enabling a Connection Pool

This section describes how to enable a connection pool by using the console tool.

#### Using the Console Tool

The **enable-connection-pool** command is used to enable the connection pool on the server. For more information about using enable-connection-pool, refer to "enable-connection-pool" in *JEUS Reference Guide*.

The following is an example of enabling the connection pool for ds1 created on server1 by executing enable-connection-pool.

```
[MASTER]domain1.adminServer>enable-connection-pool -server server1 -id ds1
Servers that successfully enabled a connection pool : server1
Servers that failed to enable a connection pool : none.
```

If the server option is not provided, then the connection pool will be enabled for all servers where the data source is registered on.

The following is an example of enabling the connection pool for ds2 on server1 and server2 where ds2 is registered on by executing enable-connection-pool.

```
[MASTER]domain1.adminServer>enable-connection-pool -id ds2
Servers that successfully enabled a connection pool : server1, server2
Servers that failed to enable a connection pool : none.
```

The **control-connection-pool** command can also be used to enable a JDBC connection pool. For more information about using control-connection-pool, refer to "control-connection-pool" in *JEUS Reference Guide*.

The following is an example of creating a connection pool for ds1 on server1 by executing control-connection-pool.

```
[MASTER]domain1.adminServer>control-connection-pool -server server1 -id ds1 -enable
Servers that successfully enabled a connection pool : server1
Servers that failed to enable a connection pool : none.
```

## 6.9.4. Replacing a Connection in a Connection Pool

This section describes how to replace a connection of a connection pool by using the console tool.

### Using the Console Tool

The **refresh-connection-pool** command is used to replace the connections in the connection pool of the server with new connections. For more information about how to use refresh-connection-pool, refer to "refresh-connection-pool" in *JEUS Reference Guide*.

The following is an example of replacing the connections in the connection pool for ds1 on server1 with new connections by executing refresh-connection-pool.

```
[MASTER]domain1.adminServer>refresh-connection-pool -server server1 -id ds1  
Servers that successfully refreshed a connection pool : server1  
Servers that failed to refresh a connection pool : none.
```

If the server option value is not provided, then a connection pool is replaced for all the servers where the data source is registered on.

The following is an example of refreshing the newly added connections in the connection pool on server1 and server2, where ds2 is registered, by executing refresh-connection-pool.

```
[MASTER]domain1.adminServer>refresh-connection-pool -id ds2  
Servers that successfully refreshed a connection pool : server1, server2  
Servers that failed to refresh a connection pool : none.
```

JDBC connection pool's connections can also be refreshed by executing **control-connection-pool**. For more information about using control-connection-pool, refer to "control-connection-pool" in *JEUS Reference Guide*.

The following is an example of replacing the connections in the connection pool for ds1 on server1 with new connections by executing refresh-connection-pool.

```
[MASTER]domain1.adminServer>control-connection-pool -server server1 -id ds1 -refresh  
Servers that successfully refreshed a connection pool : server1  
Servers that failed to refresh a connection pool : none.
```

## 6.9.5. Minimizing the Number of Connections in a Connection Pool

This section describes how to minimize the number of connections in the connection pool by using the console tool.

## Using the Console Tool

The **shrink-connection-pool** command is used to adjust the number of connections in the connection pool on the server to the specified minimum value. For more information about using shrink-connection-pool, refer to "shrink-connection-pool" in *JEUS Reference Guide*.

The following is an example of adjusting the number of connections in the connection pool for ds1 on server1 to the set minimum connection value by executing shrink-connection-pool.

```
[MASTER]domain1.adminServer>shrink-connection-pool -server server1 -id ds1
Servers that successfully shrank a connection pool : server1
Servers that failed to shrink a connection pool : none.
```

If the server option value is not provided, then the number of connections for the connection pool will be adjusted for all the servers where the data source is registered on.

The following is an example of adjusting the number of connections in the connection pool for ds2 on server1 and server2, where ds2 is registered on, to the set minimum connection value by executing shrink-connection-pool.

```
[MASTER]domain1.adminServer>shrink-connection-pool -id ds2
Servers that successfully shrank a connection pool : server1, server2
Servers that failed to shrink a connection pool : none.
```

The number of connections in the JDBC connection pool can also be adjusted to the specified minimum value by executing **control-connection-pool**. For more information about using control-connection-pool, refer to "control-connection-pool" in *JEUS Reference Guide*.

The following is an example of adjusting the number of connections in the connection pool for ds1 on server1 to the set minimum value by executing control-connection-pool.

```
[MASTER]domain1.adminServer>control-connection-pool -server server1 -id ds1 -shrink
Servers that successfully shrank a connection pool : server1
Servers that failed to shrink a connection pool : none.
```

### 6.9.6. Returning Connections to Connection Pool

This section describes how to use the console tool to return connections to the connection pool.

If the application does not close the connection after making a getConnection request to a data source, the connection will remain active unless the data source is set to AutoClose. When such a connection leak occurs, you can return the connection to the connection pool by using the return connection function.

## Using the Console Tool

The **return-connection** command is used to return an active connection to the server's connection pool. For more information about return-connection, refer to "return-connection" in *JEUS Reference Guide*.

The following is an example of using return-connection to return ds1-1 connection to the ds1 connection pool created in server1. You can use the connection-pool-info command to check whether the ds1-1 connection is currently in active state due to a leak, and use the return-connection command to return the connection.

```
[MASTER]domain1.adminServer>connection-pool-info -server server1 -id ds1
Information about connections in the server [server1]'s connection pool [ds1].
=====
+-----+-----+-----+-----+-----+
| Connection ID | State | State Time(sec) | Use Count | Type |
+-----+-----+-----+-----+-----+
| ds1-2         | idle  | 22.311          | 0         | pooled |
| ds1-1         | active| 22.289          | 0         | pooled |
+-----+-----+-----+-----+-----+
=====
[MASTER]domain1.adminServer>return-connection -server server1 -cid ds1-1
Successfully returned the connections to the connection pool.
```

You can use the cid option to return multiple connections (delimited by a comma) in the same connection pool. You can also use the cpid option to return all connections in the connection pool.

The following example uses return-connection to return all active connections in connection pool ds1 to the connection pool.

```
[MASTER]domain1.adminServer>return-connection -server server1 -cpid ds1
Successfully returned the connections to the connection pool.
```

## 6.9.7. Forcibly Destroying Connections in Connection Pool

This section describes how to forcibly destroy a connection in the connection pool by using the console tool.

If an application makes a getConnection request to the data source but there is a problem with the active connection, the connection can be forcibly destroyed by using the destroy-connection command. However, since this function forcibly destroys the physical connection, use with caution.

## Using the Console Tool

The **destroy-connection** command is used to destroy an active connection in the server's connection pool. For more information about destroy-connection, refer to "destroy-connection" in *JEUS Reference Guide*.

The following is an example of using **destroy-connection** to destroy ds1-1 connection in the ds1 connection pool created in server1. You can use the **connection-pool-info** command to check whether the ds1-1 connection is currently in active state, and use the **destroy-connection** command to destroy the connection.

```
[MASTER]domain1.adminServer>connection-pool-info -server server1 -id ds1
Information about connections in the server [server1]'s connection pool [ds1].
=====
+-----+-----+-----+-----+-----+
| Connection ID | State | State Time(sec) | Use Count | Type |
+-----+-----+-----+-----+-----+
| ds1-2        | idle  | 22.311          | 0         | pooled |
| ds1-1        | active| 22.289          | 0         | pooled |
+-----+-----+-----+-----+-----+
=====
[MASTER]domain1.adminServer>destroy-connection -server server1 -cid ds1-1
Successfully destroyed the connections from the connection pool.
```

You can use the cid option to destroy multiple connections (delimited by a comma) in the same connection pool. You can also use the cpid option to destroy all connections in the connection pool.

The following example uses destroy-connection to destroy all active connections in connection pool ds1 to the connection pool.

```
[MASTER]domain1.adminServer>destroy-connection -server server1 -cpid ds1
Successfully destroyed the connections from the connection pool.
```

## 6.10. JEUS JDBC Programming

This section briefly describes the JDBC application programming rules.

### 6.10.1. Getting a Connection from a Data Source

The following code shows how to obtain a connection by using a data source.

```
Context ctx = new InitialContext();
DataSource ds = (DataSource) ctx.lookup("ds1");
Connection con = ds.getConnection();
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("select * from test");
...
con.close();
```



Be sure to always close the connection after use.

## 6.10.2. Transaction Programming Rules

JEUS defines the standard procedure for transaction programming. All applications that use the UserTransaction interface must follow the standard procedure.

1. Start a transaction.
2. Retrieve a DB connection.
3. Code rest of the transaction.



The user must obtain a new connection to process other transactions.

## 6.10.3. Getting a Connection Implementation Instance of the JDBC Driver

In JEUS, when a connection is passed to an application, a wrapper class is used to manage the connection state. However, since Oracle's CLOB, XMLType, etc. require direct access to the connection implementation instance of the JDBC driver, a ClassCastException occurs when class casting is attempted internally in the driver. This is a problem that can occur in most WAS products, not just in JEUS. Therefore, application developers cannot assume that connections from WAS is a connection implementation instance of the driver. However, there is a way to avoid this problem.

The application can obtain a connection implementation instance of the driver as in the following way.

```
import java.sql.Connection;
import java.sql.DatabaseMetaData;
...
Connection conn = datasource.getConnection();
DatabaseMetaData metaData = conn.getMetaData();
OracleConnection oraConn = (OracleConnection)metaData.getConnection();
```

## 6.10.4. Connection Pool in a Standalone Client

A standalone client can look up a data source registered on JEUS, get the data source information from JEUS, and configure the connection pool on its own JVM. Here, the standalone client must add the clientcontainer.jar or jclient.jar to the classpath.

This method has the benefit of conveniently accessing a database in the standalone client by using the JNDI lookup method. However, in the viewpoint of the database, this allows more client accesses to the JDBC driver besides the Web application server. If the client doesn't particularly need to efficiently manage DB connections, then configuring the connection pool in the client can be a resource waste. Therefore, it is not recommended to use this method when implementing the actual service.



Connections are not obtained from the connection pool configured on JEUS server. To do so, implement the logic for using the connection as an EJB, and deploy it on JEUS and use it by looking up the EJB.

## 6.11. Global Transaction (XA) and Connection Sharing

In the JCA standard, Connection Sharing is defined to guarantee that only one connection is always used for each resource in a transaction. Most of the resources from the JDBC viewpoint are databases or data sources. Without additional configuration, by default JEUS provides connection sharing to connection pool data sources that use the XA data source and emulation functions.

In application frameworks like ProFrame, since multiple connection requests are made within the same transaction, it is preferable to use connection sharing. Otherwise, multiple number of connections can participate in the transaction by using a single data source. This puts a lot of burden on the DB in managing the transaction lock which can reduce the overall performance of the transaction.

When it is not desirable to use connection sharing, compose the Jakarta EE component configuration file (web.xml, ejb-jar.xml, etc) that uses XA data source as in the following. For `<res-ref-name>`, enter the JNDI name of the data source.

```
<resource-ref>
  <res-ref-name>jdbc/xads</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-sharing-scope>Unshareable</res-sharing-scope>
</resource-ref>
```

If no configuration is made, then by default the `<res-sharing-scope>` element is set to shareable, and a servlet or EJB always uses connection sharing. However, since the connection pool data source that uses the XA emulation function must always share the connection, a `java.sql.SQLException` error occurs when it is configured to Unshareable.

## 6.12. Connection Pooling Service Support for Various Users

JEUS has been only supporting connection pooling service for default users specified in the data source configuration. When connection was requested with a different user information, the connection was always handled as a disposable connection. Disposable connection is created whenever there is a connection request and it is removed from the connection pool after use. Hence, when there are a lot of requests for disposable connections, it can be a considerable burden on connection pooling.

For this reason, connection pooling service is supported for various users. This means that a single connection pool is transparently divided and managed per user, and the client can use the

connection pooling service same as before without noticing any changes to the previous configuration. In the connection pool, the connections with different user information still all follow the overall configuration and policy of the connection pool. This way, even if multiple user-specific connections are used frequently, the connection pooling service can be used efficiently without having the burden of creating disposable connections.



# 7. Transaction Manager

This chapter describes the transaction manager and its surrounding components in JEUS.

## 7.1. Overview

The JEUS transaction manager provides various forms of transaction services to applications in the Jakarta EE environment. The JEUS transaction manager provides services according to an application's operation requirements and connections to various resource managers. Furthermore, an application can either assign the control of a transaction to the manager or take the control of the transaction itself as needed. The transaction manager plays a central role in the transaction processing tasks working with the resource manager, application, etc.

In enterprise systems, a transaction service is the basic and important service that safely and efficiently processes large scale client requests for resources. Recently, as the types and scale of resource requests have increased, there is a need for the transaction manager to provide a unified interface and operate more safely.

The JEUS transaction manager provides compatibility with Java Transaction Service (JTS) and supports Jakarta Transaction (JTA; formerly Java Transaction API). This is to provide a unified interface and functions for a client to use a transaction. For more information, refer to related API or specification documents. Transaction service is provided in many situations (clients and server applications) where an application can be used to support various forms of applications.

To provide stable services, the transaction manager must ensure the integrity of transactions even when a problem occurs. JEUS transaction manager provides transaction recovery functions that can recover transactions under various error conditions.

The following are the three components that participate in JEUS transaction processing. Each component uses JTA, the standard API for transaction processing, to communicate with each other.

- An application that receives a service

An application can access various resources through a web container, EJB container, or client container. The application itself can control the transaction by specifying the start and the end of a transaction, or the container's transaction manager can control the transaction.

- The transaction management function of the application and the transaction manager that provides the transaction management interface.

The transaction manager is divided into the server transaction manager and the client transaction manager depending on where the application is running. It directly sends a request to prepare and apply a transaction to the actual resource manager.

- The resource manager (e.g., database) which is affected by the actual transaction.

The resource manager accepts a request and processes it.

This section describes each component and its operation method. For basic information about transactions, refer to the related books or materials.

### 7.1.1. Application

An application uses JEUS transaction manager to perform transaction related tasks on the resource manager.

- **Local transaction**

Transaction tasks that use a single resource manager can be grouped into a single local transaction. An application starts or ends a local transaction by using the resource manager driver.

By default, JEUS transaction manager is not involved in a local transaction processing. An application can process a task as a global transaction, but when processing a simple transaction task that uses a single resource manager, it is recommended to use a local transaction which is significantly more efficient.

- **Global transaction**

When an application attempts a sequence of transaction tasks using two or more resource managers, the transaction manager can group the tasks into a single global transaction. A global transaction often uses the 2 phase commit protocol to perform transaction batch jobs of the resources in use.

When the transaction manager commits a transaction, it traces the execution flow of the application and decides to perform either 1 phase commit or 2 phase commit. This decision is made according to the number and type of the resource managers used in the transaction. A JEUS transaction does not support duplicate transactions, and does not allow the mixing of two or more transactions.

- **User Managed Transaction**

An application can specify the start and end of a transaction by using the `jakarta.transaction.UserTransaction` object. The decision to commit or rollback a transaction is made entirely by an application. However, even when a commit is executed, JEUS transaction manager can still perform a rollback if the resource manager is unable to process the transaction.

In an EJB, this user managed transaction is called a bean-managed transaction (BMT).

- **Container Managed Transaction**

In a container managed transaction, the container decides the start and end of a transaction.

The commit or rollback of the transaction are decided entirely by the container. This transaction can only be used in an EJB by specifying the transaction attributes (NotSupported, Required, Supports, RequiresNew, Mandatory, Never) according to each bean's method.

Either a **local transaction** or **global transaction** can be used depending on how many resource

managers an application uses. In addition, the configuration and application programming methods depend on which agent has the authority over the transaction.

## 7.1.2. JEUS Transaction Manager

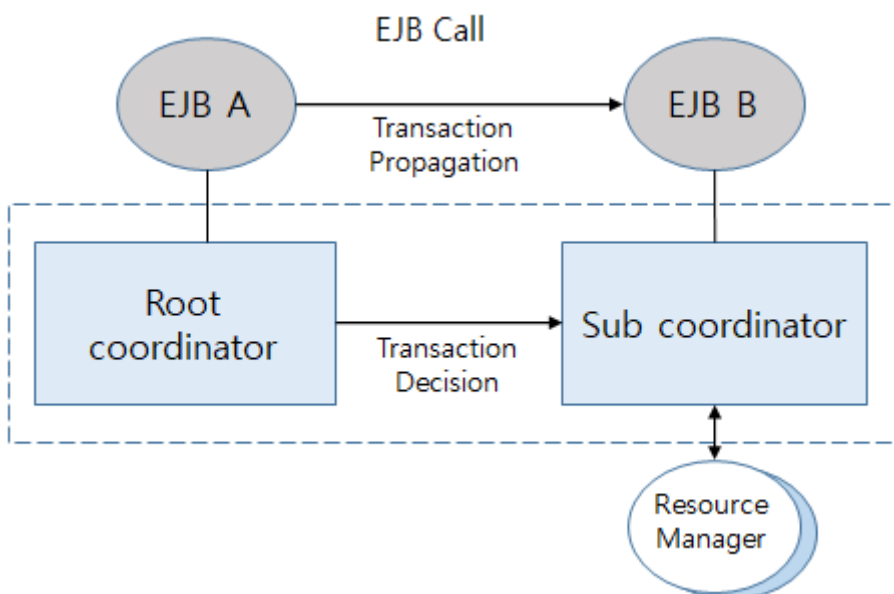
JEUS provides two transaction managers, the server transaction manager and client transaction manager.

The server transaction manager provides all functions for managing a global transaction, while the client transaction manager only acts as the proxy of a server transaction.

### Server Transaction Manager

The server transaction manager implements the Jakarta Transaction completely and is used when a server application uses a transaction.

When a global transaction is started, the server transaction manager collects all the information related to the global transaction and proceeds with the tasks acting as the transaction coordinator. Depending on the situation, the role of the server transaction manager consists of the **root coordinator** and **sub coordinator**.



Relationship between Root Coordinator and Sub Coordinator

- Root coordinator

The transaction manager that starts the transaction. The manager where the transaction is propagated to becomes the sub coordinator.

- Sub coordinator

Receives commands from the root coordinator to perform tasks. The root coordinator takes an active role in managing transactions, and the sub coordinator takes a passive role.

### Client Transaction Manager

The client transaction manager operates as the proxy of the server transaction manager.

The server transaction manager that receives the signal from a global transaction becomes the root coordinator and collects and processes all information related to the transaction. To commit the transaction, the transaction manager sends the commit signal to the root coordinator and receives the result of the commit.

### 7.1.3. Resource Manager

Applications can perform tasks with various resource managers. Connection managers manage connections to the resource manager.

JEUS provides four types of connection managers (JDBC Connection Manager, JMS Connection Manager, WebT Connection Manager, and Connector Manager). Connection managers report transaction related information to the transaction manager for processing a global transaction.

- JDBC Resource Manager

Java Database Connectivity (JDBC) defines the connection between databases and applications for Java-based application development. Applications look up and use the JDBC resource manager from the naming server. For more information, refer to [JNDI Naming Server](#) and [DB Connection Pool and JDBC](#).

- JMS Resource Manager

Jakarta Messaging (JMS; formerly Java Message Service) is the specification that defines the queue and topic and the API used for accessing the message saved in them. For more information, refer to "JEUS MQ Guide" or the JMS specification.

- Tmax (WebT) Resource Manager

Tmax is a TP-monitor developed by TmaxSoft. In JEUS, a bridge called WebT is provided for a transparent two-way service with the Tmax transaction manager. Since WebT supports a two-way call, general EJB clients can run on Tmax, and the same method can be used to call an EJB Bean on JEUS.

- Jakarta Connector (JCA) Resource Manager

As one of the Jakarta EE specifications, Jakarta Connector (JCA; formerly Java Connector Architecture) provides an integrated mechanism of the Jakarta EE compatible platform and various EISs (Enterprise Information System). By using a connector that supports XA Transactions, an application can handle various transaction tasks as a single global transaction. For more information, refer to "JEUS Jakarta Connectors Guide".

## 7.2. Server Transaction Manager Configuration

This section describes how to configure JEUS transaction manager.

JEUS users can make various selections through the configurations. The configuration items greatly influence JEUS transaction manager's performance and operation. Therefore, the user should be familiarized with each item to set an appropriate value for each.

The transaction manager can be configured using the console tool, and any additional settings can be configured as system properties.

## 7.2.1. Worker Thread Pool

In JEUS transaction manager, a multiple number of worker threads are used to support communication with other transaction managers. The transaction manager uses the system thread pool on the server by default. The thread pool can be configured on a server where transaction and fast processing is important.

- **Common Thread Pool**

If there are not too many but a constant level of transactions that occur and faster processing than other services is required, then use the system thread pool, which is shared by all services on the server.

The public thread pool can separately assign the thread count that can only be used by the transaction manager, in the '**Reserved Thread Num**' item. The number of reserved threads for the transaction manager must be set by considering the size of the entire thread pool to make sure that it doesn't cause problems for other services.



Other services can also configure their own threads, but the total number of threads cannot be greater than the thread pool size. When dynamically changing the **Reserved Thread Num**, ( $n \rightarrow m$ ) when  $n$  or  $m$  is 0, it is processed as pending and thus requires a restart.

- **Dedicated Thread Pool**

When there are a lot of transactions and if the load fluctuates a lot, it is better to create and use a dedicated thread for the transaction manager instead of using the system thread. Since the configurations for the thread pool are the same as those of the system thread pool, refer to [Thread Pool Configuration](#).

As previously described, the system thread pool or dedicated thread pool can be selected for use. However, the pool type cannot be changed during server operation. The server must be restarted to apply the modified pool type. However, with the exception of the dedicated thread pool's queue size, the pool configurations can all be dynamically changed without restarting the server.

## Using the Console Tool

The following describes how to change a thread pool by using the console tool.

- **Configuring a Common Thread Pool**

Execute **modify-system-thread-pool** to separately assign a number of threads from the thread pool to the transaction manager.

```
[MASTER]domain1.adminServer>modify-system-thread-pool server1 -service transaction -reservednum 10
Successfully performed the MODIFY operation for the transaction thread pool of the server (server1).
Check the results using "show-system-thread-pool server1 -service transaction or modify-system-thread-pool server1 -service transaction"
```

### • Configuring a Dedicated Thread Pool

Execute **modify-service-thread-pool** to use a dedicated thread for the transaction manager.

If the dedicated thread pool is configured while using the system thread pool, the server must be restarted to apply the changed configuration. If only the attributes are changed while using the dedicated thread pool, then the configuration can be applied dynamically without restarting the server.

```
[MASTER]domain1.adminServer>modify-service-thread-pool server1 -service transaction -min 10 -max 20
Successfully performed the MODIFY operation for The transaction thread pool of the server (server1), but all changes were non-dynamic. They will be applied after restarting.
Check the results using "show-service-thread-pool server1 -service transaction or modify-service-thread-pool server1 -service transaction"
```



For detailed explanations of the `modify-system-thread-pool` and `modify-service-thread-pool` commands, see "modify-service-thread-pool" in *JEUS Reference Guide*.

## 7.2.2. Timeout Settings

The JEUS transaction manager uses various timeout mechanisms for processing exceptions. The transaction manager can be tuned to be suitable for the application system by adjusting the timeout mechanism. The timeout configuration is static and requires a server restart. The configuration must be set before starting the server or the server must be restarted if it is changed while the server is running.

The following describes the configuration items.

### • Basic Settings

Item	Description
Active Timeout	Timeout to commit from the start of a global transaction. If the time expires, the transaction is forcibly rolled back.  (Default Value: 600000 (10 minutes), Unit: milliseconds)

Item	Description
Automatic Recovery	Option to automatically recover indoubt transactions to another server when the current transaction manager fails in a cluster environment. Clustering configuration is required to use this option. The log directory of the failed transaction manager must be accessible from another server.

#### • Advanced Options

Item	Description
Prepare Timeout	Timeout for the root coordinator to receive a 'prepare' signal from the sub-coordinator and resource manager. If not received within this time, then the root coordinator rolls back the global transaction. (Default Value: 120000 (2 minutes), Unit: milliseconds)
Prepared Timeout	Timeout for the sub-coordinator to wait for a global decision on whether to commit or rollback from the root coordinator. If not received within this time, then the sub-coordinator sends another response message for 'prepare' to the root coordinator. If it receives no global decision during the next timeout period, then it rolls back the global transaction. (Default Value: 60000 (1 minute), Unit: milliseconds)
Commit Timeout	Timeout for the root coordinator to receive a response (commit or rollback) from the sub-coordinator and resource manager. If not received within this time, then the root coordinator records the global transaction in the 'Incomplete List' to indicate that the transaction is incomplete. (Default Value: 240000 (4 minutes), Unit: milliseconds)
Recovery Timeout	<p>Timeout to receive the recovery information.</p> <p>To recover a transaction, the transaction manager attempts to obtain the transaction information. The transaction is rolled back if another transaction manager does not send the recovery information within this time.</p> <p>(Default Value: 120000 (2 minutes), Unit: milliseconds)</p>
Incomplete Timeout	<p>Timeout for an incomplete transaction to recover. To complete the entire transaction processing, the transaction manager maintains a list of failed global transactions. The incomplete global transaction information is used during recovery processing and is kept until the timeout expires.</p> <p>Therefore, if the timeout is too short, the recovery information will be deleted more frequently, and the transaction manager won't be able to recover the integrity of the global transaction. As a result, the system administrator must process many tasks in order to recover a global transaction. (Default Value: 86400000 (1 day), Unit: milliseconds)</p>

## Using the Console Tool

You can change the transaction manager timeout using the **modify-transaction-manager**

command. For detailed information about the command, see "modify-transaction-manager" in *JEUS Reference Guide*.

The following describes how to change the timeout setting in the console tool.

```
[MASTER]domain1.adminServer>modify-transaction-manager server1 -activeTimeout 20000
Successfully performed the MODIFY operation for transaction of server (server1), but all changes were
non-dynamic. They will be applied after restarting.
Check the results using "show-transaction-manager server1 or modify-transaction-manager server1"
```

### 7.2.3. Root Coordinator and Sub Coordinator

The transaction manager is divided into the root coordinator and sub coordinator depending on its role. Since the sub coordinator gets transaction-related information from the root coordinator, it must register itself with the root coordinator and make sure that the root coordinator knows that the sub coordinator exists.

The configuration consists of the following two system properties. The user should set the properties carefully by keeping performance and safety in mind.

- `-Djeus.tm.forcedReg=<true or false>`

Determines when the sub coordinator registers itself with the root coordinator.

Configuration Value	Description
true	Sub coordinator immediately registers itself when a transaction is transmitted. (default value)
false	Sub coordinator registers itself when the resource manager connected to the sub coordinator is actually used. If set to false, the communication count between transaction managers can be reduced when there are a lot of EJB calls that do not use the resource manager registered with the sub coordinator.

- `-Djeus.tm.checkReg=<true or false>`

Configuration Value	Description
true	When the sub coordinator registers itself, it waits for an ACK from the root coordinator. If the ACK doesn't arrive within the time frame, the registration is considered as failed and any pending transactions are rolled back. (default value)



Configuration Value	Description
false	When the sub coordinator registers itself, it doesn't wait for an ACK from the root coordinator. Therefore, when the registration fails due to a problem such as a network error, the sub coordinator is not notified of the problem and thus an error occurs during the commit operation.

## 7.2.4. Transaction Join

JEUS transaction manager joins the transaction resources of the same resource manager. This allows tasks to be performed without creating additional transaction branches and decreases the overhead. However, this method can be a problem in situations such as when Oracle OCI is used like RAC.

To prepare for such problems, set the following option to "true".

```
-Djeus.tm.disableJoin=true
```

## 7.3. Client Transaction Manager Configuration

The client transaction manager is created to allow a client application to start and end global transactions. This section describes the configurations that are related to the client.

Since the client application doesn't use the configurations in domain.xml, it adds the configuration in the application start script by using the system property format.

### 7.3.1. Using a Transaction Manager

The client transaction manager is initialized when the client application performs a JNDI lookup. However, depending on the client application, there are situations when the transaction manager is not used. To remove any additional overhead that occurs in such situation, the following information is saved in the start script.

```
-Djeus.tm.not_use=true
```

### 7.3.2. Transaction Manager Type

The client transaction manager is only used in the client container. By default, the transaction manager is used in the client, but to use the server transaction manager in the client, add the following setting to the client application script. For information about the difference between the server transaction manager and client transaction manager, refer to [JEUS Transaction Manager](#)

```
-Djeus.tm.version=server
```

### 7.3.3. TCP/IP Port of the Transaction Manager

The client transaction manager uses the TCP/IP port to communicate with other transaction managers.

The client transaction manager automatically finds and uses the appropriate port. To manually set the configuration, add the following setting to the client application script.

```
-Djeus.tm.port=<port number>
```

### 7.3.4. Worker Thread Pool

The worker thread pool setting configures the worker thread pool information.

- **Min**

- To configure this value in the client transaction manager to the number of worker threads created in the pool, add the following setting to the client start script.

```
-Djeus.tm.tmMin=<number of threads>
```

- **Max**

- To configure this value in the client transaction manager to the maximum number of threads created in the pool, add the following setting to the client start script.

```
-Djeus.tm.tmMax=<number of threads>
```

### 7.3.5. Timeout Settings

The following describes how to configure the transaction manager from a client.

- **Active Timeout**

- Timeout to commit from the start of a global transaction. If the time expires, the transaction is forcibly rolled back. (Default Value: 600000 (10min), Unit: ms)
- Add the following setting to the client application script to configure this value in the client container.

```
-Djeus.tm.activeto=<time in milliseconds>
```

#### • Prepare Timeout

- The root coordinator must receive a 'prepare' signal from the sub coordinator and resource manager within this time. (Default value: 120000 (2min), Unit: ms)
- If the signal is not received, then the root coordinator rolls back the global transaction.
- Add the following setting to the client application script to configure this value in the client container.

```
-Djeus.tm.prepareto=<time in milliseconds>
```

#### • Prepared Timeout

- The sub coordinator must receive a global decision on whether to commit or rollback from its root coordinator within this time. (Default value: 60000 (1min), Unit: ms)
- If not received within this timeout, then the sub-coordinator sends another response message for 'prepare' to the root coordinator. If it receives no global decision during the next timeout period, then it rolls back the global transaction.
- Add the following setting to the client application script to configure this value in the client container.

```
-Djeus.tm.preparedto=<time in milliseconds>
```

#### • Commit Timeout

- The root coordinator receives a commit or rollback response from the sub coordinator and resource manager within this time. (Default value: 240000 (4min), Unit: ms)
- If not received within this timeout, then the root coordinator records the global transaction in the "Incompleted List" to indicate that the transaction has not been completed.
- Add the following setting to the client application script to configure this value in the client container.

```
-Djeus.tm.committo=<time in milliseconds>
```

#### • Recovery Timeout

- Timeout to receive the recovery information. To recover a transaction, the transaction manager attempts to obtain the transaction information. (Default Value: 120000 (2min), Unit: ms)
- The transaction is rolled back if another transaction manager does not send the recovery information within this time.

- Add the following setting to the client application script to configure this value in the client container.

```
-Djeus.tm.recoveryto=<time in milliseconds>
```

- **Incomplete Timeout**

- To complete the entire transaction processing, the transaction manager maintains a list of failed global transactions. The incomplete global transaction information is used during recovery processing and is kept until the timeout expires. Therefore, if the timeout is too short, the recovery information will be deleted more frequently, and the transaction manager won't be able to recover the integrity of the global transaction. As a result, the system administrator must process many tasks in order to recover a global transaction. (Default value: 86400000 (1day), Unit: ms)
- Add the following setting to the client application script to configure this value in the client container.

```
-Djeus.tm.incompleteto=<time in milliseconds>
```

## 7.4. Transaction Application Programming

This section describes some JEUS transaction programming examples.

The following are the 4 programming patterns.

- **Local transaction**
- Client-managed transaction
- Bean-managed transaction
- Container-managed transaction

Users and application programmers must first specify in what form the application is to run before starting to program. This will allow them to achieve the expected results and easily resolve any problems that may occur later.

An application can manage transaction tasks as a single transaction. A local transaction is used if a single resource manager can process the tasks. Otherwise, a global transaction must be used to manage the transaction.

### 7.4.1. Local Transaction

Using a local transaction is an effective way to manage the transaction tasks of a single resource manager. Since a local transaction is light and fast, it is used whenever possible. The local transaction has nothing to do with JEUS transaction manager and can use all types of containers.

The following is an example of using a local transaction. Although it is a Java application, some of the code can be used in other Jakarta EE programs such as a servlet or EJB.

#### Example of Using a Local Transaction: <Client.java>

```
import javax.naming.*;
import javax.rmi.PortableRemoteObject;
import java.util.*;
import jakarta.transaction.UserTransaction;
import java.sql.*;
import javax.sql.*;

public class Client {
    private static Connection con;

    private static Connection getConnection() throws
        NamingException, SQLException {
        // get a JDBC connection
    }

    private static String insertCustomer(String id, String name,
        String phone) throws NamingException, SQLException {
        // insert a product entity given by the arguments to DB
    }

    private static void deleteCustomer(String id) throws
        NamingException, SQLException {
        // delete a product entity given by the arguments from DB
    }

    public static void main(String[] args) {
        try {
            // get a JDBC connection
            con = getConnection();

            // set the autocommit attribute as false
            con.setAutoCommit(false);

            // insert customers
            for (int i=0 ; i<number/2 ; i++) {
                System.out.println("inserting customer id="+i+"c... from Client");
                customers[i] = insertCustomer(i+"c", "Hong Kil Dong "+i, "000-123-1234-"+i);
            }
            System.out.println("completed inserting customers!!");
            con.commit();

            // delete customers
            for (int i=0 ; i<number/2 ; i++) {
                System.out.println("deleting customerid="+customers[i]+" ... from Client");
                deleteCustomer(customers[i]);
            }
            System.out.println("completed deleting customers!!");
            con.commit();

        } catch (NamingException ne) {
            System.out.println("Naming Exception caught : " + ne.getMessage());
        } catch (SQLException sqle) {
            System.out.println("SQL Exception caught : " + sqle.getMessage());
        }
    }
}
```

```

    } catch(Exception e) {
        try {
            con.rollback();
        } catch (Exception ee) {
            System.out.println("Transaction Rollback error : " + ee.getMessage());
        }
        System.out.println("Error caught : " + e.getMessage());
        e.printStackTrace();
    } finally {
        try {
            if (con!=null) con.close();
        } catch (SQLException se) {}
    }
}
}
}

```

## 7.4.2. Client-Managed Transaction

A global transaction can be used in an application of a client container (client managed transaction). The client itself manages transactions by using `UserTransaction` and calls EJB to process the actual task.

The following is an example of this.

### Client

Before starting a global transaction, import a `jakarta.transaction.UserTransaction` instance by looking up 'java:comp/UserTransaction'. After starting a global transaction by calling `begin()` on the instance, call the EJB bean multiple times. To commit the transaction task on the EJB Bean, call the `commit()` method of the `UserTransaction` instance to notify that the transaction is complete.

The method terminates if the transaction is successfully committed. If the global transaction is not committed due to a failure, then an exception is thrown by the method. The `jakarta.transaction.RollbackException` is thrown by JEUS transaction manager to guarantee that the global transaction is rolled back. There are also other exceptions that are thrown to notify that JEUS transaction manager is attempting to rollback the global transaction due to an unexpected exception. However, not all transaction tasks of the global transaction are completely rolled back. The `rollback()` method of the `UserTransaction` instance is called directly from the catch query to roll back the global transaction.



Since a standalone client does not include the concept of components, it supports 'java:/UserTransaction' in order to use another name to lookup 'java:comp/UserTransaction' in the standalone client. In practice, either `java:comp/UserTransaction` or `java:/UserTransaction` can be used to look up transactions.

## Example of Using a Client-Managed Transaction: <Client.java>

```
package umt;

import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.rmi.PortableRemoteObject;
import java.util.*;
import jakarta.transaction.UserTransaction;

public class Client {
    public static void main(String[] args) {
        UserTransaction tx = null;
        try {
            InitialContext initial = new InitialContext();
            ProductManager productManager =
                (ProductManager)initial.lookup("productmanager");
            System.out.println("");
            System.out.println("< Testing ProductManager EJBBean " + "Using User Managed Transaction >");
            System.out.println("");
            int number = 10;
            String products[] = new String[number];
            tx = (UserTransaction)initial.lookup("java:comp/UserTransaction");
            tx.begin();
            // insert products
            for (int i=0 ; i<number/2 ; i++) {
                System.out.println("inserting product id="+i+"b...");
                // bean call
                products[i] = productManager.insertProduct(i+"b","ball pen", i*10);
            }
            for (int i=number/2 ; i<number ; i++) {
                System.out.println("inserting product id="+i+"f...");
                products[i] = productManager.insertProduct(i+"f", "fountain pen", (i-number/3)*50);
            }
            System.out.println("completed inserting products!!");
            // delete products
            for (int i=0 ; i<number ; i++) {
                System.out.println("deleting productid="+products[i]+" ...");
                productManager.deleteProduct(products[i]); // bean call
            }
            System.out.println("completed deleting products!!");
            tx.commit();
        } catch (jakarta.transaction.SystemException se) {
            System.out.println("Transaction System Error caught : " + se.getMessage());
        } catch (jakarta.transaction.RollbackException re) {
            System.out.println("Transaction Rollback Errorcaught : " + re.getMessage());
        } catch (Exception e) {
            try {
                tx.rollback();
            } catch (Exception ee) {
                System.out.println("Transaction Rollback error : " + ee.getMessage());
            }

            System.out.println("Error caught : " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

## EJB

The following EJB bean has a method to process transactions. To manage global transactions on the client, the transaction type of the EJB must be container-managed (CMT), which is the default type. If you want to specify it explicitly, specify the `TransactionManagerType` value as follows:

Example of Using a Client-Managed Transaction: <ProductManagerEJB.java>

```
package umt;

import jakarta.ejb.*;
import jakarta.annotation.*;

@Stateless
@TransactionManagement(TransactionManagementType.CONTAINER)
public class ProductManagerEJB implements ProductManager {
    ...

    public String insertProduct(String id, String name, double price) {
        // insert a product entity given by the arguments to DB
    }

    public void deleteProduct(String id) {
        // delete a product entity indicated by the argument from
        // DB
    }
    ...
}
```

### 7.4.3. Bean-Managed Transaction

A bean managed transaction uses JTA in an application of a web container and EJB container to explicitly mark the boundaries of a global transaction. It is useful when the application needs to fine-tune the global transaction. The application can decide to rollback and commit on its own since client requests can be processed according to the execution order.

The following is an example of an application executing a global transaction in the EJB container.

#### Client

In a BMT (Bean-Managed Transaction), an EJB bean processes the global transaction area. Hence, a client simply needs to call the method that performs the transaction tasks.

Example of Using a Bean-Managed Transaction: <Client.java>

```
package bmt;

import javax.naming.*;
import javax.rmi.PortableRemoteObject;
import java.util.*;

public class Client {
    public static void main(String[] args) {
```



```

    try {
        ProductManager productManager;
        ...
        // Getting a reference to an instance of
        // ProductManager EJB bean.
        ...
        productManager.transactionTest();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
}

```

## EJB

The EJB bean uses `jakarta.transaction.UserTransaction` to start a global transaction.

It gets an instance of `UserTransaction` using `jakarta.ejb.EJBContext` (in this example, the EJB is a session bean, so `jakarta.ejb.SessionContext`). The global transaction is initiated and committed by the executing the method.

For the EJB to act as a bean-managed transaction, set the `TransactionManagement` to `TransactionManagementType.BEAN` using annotation.

Example of Using a Bean-Managed Transaction: `<ProductManagerEJB.java>`

```

package bmt;

import jakarta.ejb.*;
import javax.naming.*;
import java.sql.*;
import java.util.*;
import jakarta.annotation.*;
import jakarta.transaction.UserTransaction;

@Stateless
@TransactionManagement(TransactionManagementType.BEAN)
public class ProductManagerEJB implements ProductManager {
    @Resource SessionContext ejbContext;

    public void transactionTest() {
        UserTransaction tx = null;
        try {
            int number = 20;
            String products[] = new String[number];
            tx = ejbContext.getUserTransaction();
            tx.begin();
            // insert products
            for (int i=0 ; i<number/2 ; i++) {
                System.out.println("inserting product id="+i+"b ...");
                products[i] = insertProduct(i+"b", "ball pen", i*10);
            }
            for (int i=number/2 ; i<number ; i++) {
                System.out.println("inserting product id="+i+"f...");
                products[i] = insertProduct(i+"f", "fountain pen",
                    (i-number/3)*50);
            }
        }
    }
}

```

```

    }
    System.out.println("completed inserting products!!");
    // delete products
    for (int i=0 ; i<number ; i++) {
        System.out.println("deleting product id="+products[i]+" ...");
        deleteProduct(products[i]);
    }
    System.out.println("completed deleting products!!");
    tx.commit();
} catch (jakarta.transaction.SystemException se) {
    throw new EJBException(
        "Transaction System Error caught : " + se.getMessage());
} catch (jakarta.transaction.RollbackException re) {
    throw new EJBException(
        "Transaction Rollback Error caught : " + re.getMessage());
} catch (Exception e) {
    try {
        tx.rollback();
    } catch (Exception ee) {
        throw new EJBException("Transaction Rollback error : " + ee.getMessage());
    }
    throw new EJBException("Error caught : " + e.getMessage());
}
}

private String insertProduct(String id, String name, double price)
    throws NamingException, SQLException {
    // insert a product entity given by the arguments to DB
}

private void deleteProduct(String id) throws NamingException,SQLException {
    // delete a product entity indicated by the argument from
    // DB
}

// some EJB callback methods
}

```

## 7.4.4. Container-Managed Transaction

According to the Jakarta EE specification, an application can delegate the demarcation of a global transaction to a container. A Web or EJB container manages the global transactions related to the method. An application can configure the transaction attributes for each method in the configuration file. This is the easiest way to process a global transaction.

The following is an example of a global transaction managed by an EJB container.

### Client

In the following example, a server-side EJB container is managing the global transaction tasks of the EJB in a container managed transaction.

Example of Using a Container-Managed Transaction: <Client.java>

```
package bmt;
```

```

import javax.naming.*;
import javax.rmi.PortableRemoteObject;
import java.util.*;

public class Client {
    public static void main(String[] args) {
        try {
            ProductManager productManager;
            // getting a reference to an instance of
            // ProductManager EJB bean.
            productManager.transactionTest();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

## EJB

The advantage of using a container-managed transaction is that it prevents the developer from implementing transaction-related codes in the business logic.

As long as appropriate attributes are set in the method of an EJB bean, global transaction related tasks are all delegated to the EJB container. In the following example, the `transactionTest` method does not contain any transaction related codes. To notify the EJB container to run the EJB Bean as a container managed transaction, either set the `TransactionManagement` Annotation to `TransactionManagerType.CONTAINER` or don't set it to anything.

Example of Using a Container-Managed Transaction: <ProductManagerEJB.java>

```

package cmt;

import jakarta.ejb.*;
import javax.naming.*;
import java.sql.*;
import java.util.*;
import jakarta.annotation.*;

@Stateless
@TransactionManagement(TransactionManagementType.CONTAINER)**
public class ProductManagerEJB implements ProductManager {
    public void transactionTest() {
        try {
            int number = 20;
            String products[] = new String[number];
            // insert products
            for (int i=0 ; i<number/2 ; i++) {
                System.out.println("inserting product id="+i+"b...");
                products[i] = insertProduct(i+"b", "ball pen", i*10);
            }
            for (int i=number/2 ; i<number ; i++) {
                System.out.println("inserting product id="+i+"f...");
                products[i] = insertProduct(i+"f", "fountain pen", (i-number/3)*50);
            }
        }
    }
}

```

```

        System.out.println("completed inserting products!!");
        // delete products
        for (int i=0 ; i<number ; i++) {
            System.out.println("deleting product id="+products[i]+" ...");
            deleteProduct(products[i]);
        }
        System.out.println("completed deleting products!!");
    } catch(Exception e) {
        throw new EJBException("Error caught : " + e.getMessage());
    }
}

private String insertProduct(String id, String name, double price)
    throws NamingException, SQLException {
    // insert a product entity given by the arguments to DB
}

private void deleteProduct(String id) throws NamingException, SQLException {
    // delete a product entity indicated by the argument from
    // DB
}

// some EJB callback methods
}

```

## 7.4.5. Using a Transaction Manager

A transaction manager can be obtained by looking up the name of `java:appserver/TransactionManager` from the JNDI Context. A transaction manager can start and end a transaction and provide the function that registers an XAResource or gets a transaction object. For more information, refer to Jakarta Transaction (JTA) specification.



The method for getting a transaction manager or `UserTransaction` in JEUS is compatible with Glassfish or SunOne Application Server. Therefore, the same server configuration settings can be used when a 3rd-party library like Hibernate is used.

## 7.5. Transaction Recovery

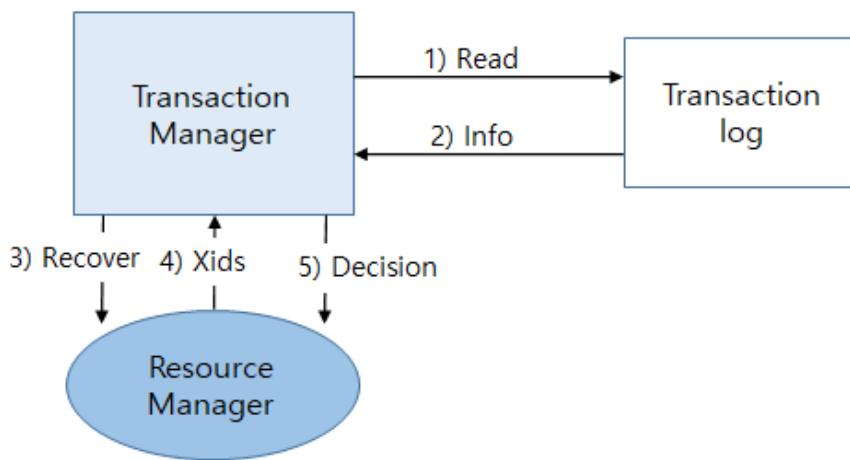
This section describes how to recover a transaction. Transaction recovery is an important function used for supporting transaction integrity when an unpredictable error occurs. Before using JEUS, make sure to fully understand the previously explained configurations and recovery process of transactions.

### 7.5.1. Transaction Recovery Process

A transaction manager must have a recovery plan for failures that can happen during a transaction.

The recovery plan is focused on guaranteeing the integrity of the transactions that were in progress. The recovery plan varies depending on whether the manager is the root coordinator or the sub coordinator, and whether tasks are performed with an external task manager instead of JEUS.

The following shows the recovery process of a transaction manager.



Simplified Transaction Recovery Process

- 1, 2) When a transaction manager recovers from a failure, the manager gets transaction information from the transaction log.
- 3) Next, it sends the recover command to a resource manager according to the log information.
- 4) In response, the resource manager sends the XIDs of incomplete transactions to the manager.
- 5) The transaction manager sends a commit or rollback command to the resource manager by using the information gathered from the XIDs and logs.

The recovery method differs slightly according to the transaction manager's role, but overall it follows the aforementioned process. A brief description of each component that participates in the recovery and related configurations are described next.

## The Recovery Process by Transaction Manager Type

The transaction manager plays a central role in the transaction recovery. The following describes how the recovery method differs according to the role of the transaction manager.

- Root Coordinator

When the transaction manager is the root coordinator, it performs operations such as [Simplified Transaction Recovery Process](#).

- Sub Coordinator

The sub coordinator is registered like a single resource manager to the root coordinator.

Since all decisions come from the root coordinator, the sub coordinator only performs up to the task of getting the XIDs for the resource managers and log files that are registered to the sub coordinator. After this, the XIDs are sent to the root coordinator and waits for a decision. When the decision comes from the root coordinator, the information is sent to the resource manager

and completes the recovery.

- Working with an External Transaction Manager

JEUS transaction manager runs as the root coordinator and an external transaction manager recognizes JEUS as a single resource manager. In this case, JEUS transaction manager performs all tasks other than making the decision. Later, when the external transaction manager sends the recover command and decision, the recovery task is completed accordingly.

## Automatic Transaction Recovery

In an automatic transaction recovery, if the current transaction manager shuts down abnormally in a cluster environment, another server's transaction manager automatically recovers any "indoubt" transactions. To properly execute this function, servers must be in a cluster configuration and the server that is executing the recovery must be able to access the transaction log of the failed server.

The following describes the process of configuring the automatic transaction recovery by using the console tool. This configuration is applied dynamically without restarting the server.

- Using the console tool

Execute **modify-transaction-manager** in the console tool to turn on the automatic transaction recovery function. For more information about the command, refer to "modify-transaction-manager" in *JEUS Reference Guide*.

```
[MASTER]domain1.adminServer>modify-transaction-manager server1 -automaticRecovery true
Successfully performed the MODIFY operation for transaction of server (server1), but all changes
were non-dynamic.
They will be applied after restarting.
Check the results using "show-transaction-manager server1 or modify-transaction-manager server1"
```

## 7.5.2. Recovery Log File

The following are the two types of log files that are used to recover transactions.

- The log that contains the transaction history.
- The log of the XA resource that was used.

The log file is created in a sub directory of 'SERVER\_HOME/.workspace/mlog' by default. For the path of SERVER\_HOME, refer to [Server Directory Structure](#). The log file path can be changed as described in the following.

For example, a public directory can be designated if it is preferable to save a log to a location where many servers can access for automatic recovery. Back up the log and then restart JEUS if a recovery related problem occurs during operation due to a transaction that doesn't need recovery.

The following describes the process of configuring recovery log files using the console tool.

- **Using the console tool**

The transaction log directory can be changed by using the **modify-transaction-manager** command of the console tool. For more information about the command, refer to "modify-transaction-manager" in *JEUS Reference Guide*.

```
[MASTER]domain1.adminServer>modify-transaction-manager server1 -txLogDir
/Users/user1/jeus/domain1/tmlogs
Successfully performed the MODIFY operation for transaction of server (server1) ,but ALL changes
were non-dynamic.
They will be applied after restarting.
Check the results using "show-transaction-manager server1 or modify-transaction-manager server1"
```

### 7.5.3. Recovery Related Configuration

Recovery related configurations are listed in this section. They must be added to the server's JVM configuration as system properties. For information about how to add each configuration, refer to "Changing a Server's JVM Settings" in *JEUS Domain Guide*.

- Recovery logging can be blocked for when the system administrator is executing recovery or when performance improvement is needed.

```
-Djeus.tm.noLogging=true
```

- If recovery fails due to a problem, then the transaction manager will attempt to recover the resource again. Recovery executes once every two minutes, and the user can designate the retry count.

Add the following property to the script to set the retry count. (Default value: 30)

```
-Djeus.tm.recoveryTrial=<number of trial>
```

- If there is a transaction that needs to be recovered when a server starts, recovery is performed before the server goes into STANDBY. If there's a failed resource during recovery, then recovery is reattempted in the background.

The retry interval can be configured. (Default value: (120000) 2 minutes, Unit: milliseconds)

```
-Djeus.tm.recoveryInterval=<time in milliseconds>
```

- JEUS cannot boot if a TM log file is deleted and cannot be recovered.

If recovery is not needed, JEUS detects the broken file and resets and deletes all TM log files for a successful startup. (Default value: false)

```
-Djeus.tm.ignore.broken.log.file=<true or false>
```

## 7.5.4. Resource Manager Failure

When the resource manager fails, the system manager uses the console tool to manually perform recovery. This is because JEUS transaction manager cannot find out whether the resource manager is again in a state where it can perform transaction processing.

After resolving the problem with the resource manager, recovery is proceeded by executing the **recover-transactions** command in the console tool. For more information about using the console tool, refer to "recover-transactions" in *JEUS Reference Guide*.

## 7.6. Transaction Profile Function

JEUS transaction manager provides a function that can execute the user's callback at critical points during each transaction. It cannot be used on the client, and can only be used on the server.

This requires the implementation of a profile listener for users inheriting the following interface.

Profile Listener: <CoordinatorProfileListener.java>

```
package jeus.transaction.profile;

import jeus.transaction.info.TransactionInfo;

public interface CoordinatorProfileListener extends ProfileListener {

    public void beforePrepare( TransactionInfo info );

    public void afterPrepare( jeus.transaction.info.TransactionInfo info );

    public void beforeSetDecision( TransactionInfo info );

    public void afterSetDecision( TransactionInfo info );

    public void beforeCommit( TransactionInfo info );

    public void afterCommit( TransactionInfo info );

    public void beforeOnePhaseCommit( TransactionInfo info );

    public void afterOnePhaseCommit( TransactionInfo info );

    public void beforeRollback( TransactionInfo info );

    public void afterRollback( jeus.transaction.info.TransactionInfo info );

    public void beforeDestroy( TransactionInfo info );

    public void afterDestroy( TransactionInfo info );

}
```



```

    public void beforeActiveTimeout( TransactionInfo info );

    public void afterActiveTimeout( TransactionInfo info );
}

```

Profile Listener: <TransactionInfo.java>

```

package jeus.transaction.info;

import jakarta.transaction.xa.Xid;

public interface TransactionInfo {

    public static final int JEUS_SPECIFIC_CURRENT_FORMAT_XID = 303077;
    public static final int UNSPECIFIED_TIME = -1;

    /**
     * Returns this transaction's XID. The returned XID is an XID of
     * the internal implementation of JEUS and is serializable.
     * Hereafter, when expressing this XID as a string,
     * use XidToString.getXidHexString().
     *
     * @return the TX's XID implementation
     */
    public Xid getXid();

    /**
     * Returns the transaction manager information as a string. Through this information,
     * the TX's TM server information (ip, port, etc.) can be known.
     *
     * @return the TX's transaction manager information.
     */
    public String getCoordinator();

    /**
     * Returns an external XID for an externally initiated TX. The returned XID is replaced
     * by an XID of the internal implementation of JEUS and is serializable.
     *
     * @return the TX's XID implementation
     */
    public Xid getExternalXid();

    /**
     * Returns the active timeout setting in ms.
     *
     * @return the active timeout setting of the TX.
     */
    public long getTimeout();

    /**
     * Returns the time elapsed since the start of the TX in ms.
     *
     * @return the elapsed time of the TX.
     */
    public long getElapseSinceBegin();

    /**
     * Returns the TX's status as a string.
     */
}

```

```

*
* @return the status value of the TX as a string.
*/
public String getStatus();

/**
* Returns the XAResources enlisted in the TX as a string.
*
* @return the XAResources enlisted in the TX.
*/
public String[] getXaResources();

/**
* Returns the thread that started the TX.
* This information cannot be obtained from where TX is propagated to.
*
* @return the thread that started the TX.
*/
public Thread getBeginThread();
}

```

The implemented listener is configured as a system property. A space, semi colon, or comma is used as a separator for multiple listener classes.

```
-Djeus.tm.profile.classes="test.profile.MyCoordinatorListener1,test.profile.MyCoordinatorListener2"
```

## 7.7. Transaction Communication Problem between Servers with Different IP Bands

An IP address is used to connect between JEUS transaction managers for communication. However, communication may not be possible between JEUS transaction managers in the NAT and external environments due to different IP bands.

- This can be resolved by setting the property that maps the real IP to a virtual IP.

Refer to 'JEUS\_HOME/templates/nat.properties.template' when creating an IP mapping property and configure it as a system property as follows:

```
-Djeus.tm.net.address-mapping-properties=<full path of properties file>
```

## 8. Logging

This chapter describes the JEUS logging system, JEUS logger structure, log message contents, and how to set each logger and handler.

### 8.1. Overview

JEUS logging stores and records information about the series of tasks executed on the system while JEUS was running. This function allows a system manager to obtain information about the tasks that are performed on JEUS, and to identify and handle runtime errors.

The JEUS logging system notifies the users about the events that occur in the system and applications. JEUS uses the standard logging API(`java.util.logging`) that is provided by Java SE. The logging system structure and configuration methods follow the logging API and directly reflects the logger, handler, and formatter structures. Developers can use jeus logger through the logging API.

Since JEUS logger system is implemented based on the Java SE Logging API, developers can customize the logger by using the API.

The JEUS launcher, JEUS server, and access log that is used by a web engine are jeus logger types. Each logger is created with "jeus" (the "jeus" here refers to the logger name, hereafter jeus logger) logger as the standard. JEUS server creates jeus logger by default. Various modules, which are created below jeus logger, can create and use the loggers below JEUS such as `jeus.ejb`.



The name of the top-level logger used in JEUS is jeus logger. The logger is created by default without additional configuration.

The following are the logger types that can be configured in the JEUS configuration file.

- JEUS logger
- User logger
- Access logger used in the web engine
- Sub logger of JEUS logger
- Java logger

A handler records log messages that a logger outputs, and it always accompanies a logger. In JEUS, the following 4 handler types can be configured depending on the output destination of the log message. The log messages are recorded using the configured handler, and a file handler is used most commonly.

Division	Description
File Handler	Records log messages recorded by a logger to a file.

Division	Description
SMTP Handler	Records log messages through e-mail.
Socket Handler	Records logs through a specified IP.
User Handler	Records log messages by using a user-configured handler.

Refer to [Checking Logger Information](#) for more information about other handlers and how to set a handler in the JEUS configuration file. For more information about the logging system, refer to logging API in Java SE.

## Major Changes by Version

The following describes the major changes for each version.

1. Console handlers can be configured up to JEUS 6, but not in JEUS 7.
2. JEUS 7 does not provide a console as a server process. Since a server starts through a launcher, the console is a process of the launcher not the server.

To output log messages from a server to a console tool, a launcher process must be running until the server shuts down. If the '-verbose' option is used to start the server, the launcher process does not terminate until the server shuts down and the log messages started by the server are outputted to the console.

3. Asynchronous logging is supported from JEUS 7 Fix #4. Logging is handled by a logger thread, not by a worker thread that calls log, to improve performance. If asynchronous logging is used, only file handler is supported because other handlers may delay logging. To use the other handlers, set the `jeus.logging.useAsync` option to false. When this logging is used, the `jeus.access.logging.skip.when.busy` option is set to true. This configuration automatically detects that a web worker which needs high performance waits for logging. Therefore, an access logger does not need to wait for logging.
4. From JEUS 8.5 version, we introduce a status logger that can record an exception condition of the Asynchronous logger.

The asynchronous logger is used to find the cause when a problem occurs in the logging operation. The status logger is not set by the user. By default, the file handler (FileHandler) logs the problem in the `JeusLoggerStatus.log` file when a problem occurs. If there is no problem, the `JeusLoggerStatus.log` file is not generated.

5. Starting from version 8.5, JEUS supports pattern formatting, allowing users to specify the format of logs to be displayed during logging.

This is applied to JEUS server logs, supporting 8 types of format strings (%d, %l, %j, %T, %c, %M, %N, %m). Users can combine format strings and general characters to set their desired logging pattern. However, '%' character cannot be included in the logging pattern to identify format

strings.

SimpleFormatter and SimpleMillisFormatter, which were supported by the previous versions of JEUS, can also output logs in logging patterns.

- SimpleFormatter

```
[%d{yyyy.MM.dd HH:mm:ss}][%l] [%J-%T] [%M-%N] %m
```

- SimpleMillisFormatter

```
[%d{yyyy.MM.dd HH:mm:ss:SSS}][%l] [%J-%T] [%M-%N] %m
```

The default value of the logging pattern is the same as the logging pattern used by the existing SimpleFormatter. For more information such as format strings, refer to [Checking Logger Information](#).

6. Starting with JEUS version 21, the size of the buffer for storing events of the Asynchronous logger and the waiting strategy to be used by the consumer when there are no log events in the buffer can be specified as system properties.

If you have a large amount of log events, you can increase `jeus.logging.async.bufferSize` to allow the event buffer to accommodate more events. The default is 2048, and the maximum is 262144.

The waiting strategy of the log event consumer can be set via `jeus.logging.async.waitStrategy`. The default value is `block`. In addition to this, `sleep`, `yield`, `busyspin`, and `timeout` are supported.

7. Starting with JEUS version 9, environment variables and system properties can be included in the path when setting the log-home and rotation-dir paths of each server and the file handler. Note that the variable name must be specified in the format `${VAR}`.

## 8.2. Basic JEUS Logger Structure

This section describes the basic concepts and configuration methods of jeus logger.

### 8.2.1. Overview

By default, jeus logger exists even if a logger is not configured on the server. It uses a file handler and allows the rotation of the e-mail log files. Refer to [Checking Logger Information](#) for more information about the log rotation and file handlers.



Jeus logger cannot be deleted by using the command that deletes loggers. If jeus logger is deleted when no other logger has been configured, no log messages are logged, causing problem with operation. In order to prevent leaving a logger

on a server, set jeus logger level to OFF.

Deleting a server registered in the domain does not delete the log directory of the server. Since the logger provides important information for the administrator to view at any time, it is left for the administrator to delete the logger.

## Log Directory Structure

If a file name is not separately configured in the logger, then the log files are created in a location that is set on JEUS. The following is the structure of the log directory.

```
SERVER_HOME
|--logs
|   |--servlet
|       |--access.log
|   |--JeusLauncher.log
|   |--JeusServer.log
|   |--JeusLoggerStatus.log
|   |--jvm.log
```

The following describes the default log files created in the log directory.

### servlet/access.log

Access log file for web application requests. By default, the requests that are sent to any web application on the server are recorded in this file. When a virtual host is configured on the web engine, a directory with the name of the virtual host is created below the servlet directory. The access.log files are created below this directory where the requests sent to the host are logged.

### JeusLauncher.log

Log file for the information created by the launcher to start the server and the log messages created during server startup.

### JeusServer.log

Basic log file of the server. If a server does not have logger-related configurations, this file becomes the basic log file of all the loggers including jeus logger.

### JeusLoggerStatus.log

JEUS 7 Fix4 is a log file that records exceptions during the operation of the Asynchronous logger (server logger) which is set as default. The log file is only generated if an exception occurs in the Asynchronous logger.

### jvm.log

GC logs or thread dumps that occur on the server JVM. This file is created when a specific JVM option is used to start a server through a launcher.

Add the following to the JVM option when starting a server through a launcher.

```
-XX:+UnlockDiagnosticVMOptions -XX:+LogVMOutput -XX:LogFile=SERVER_HOME/logs/jvm.log
```

Since the server's thread dump is not output to the console because the server is running as a background process, a separate file must be configured for JVM logs. To change the log file path, add the '-XX:LogFile' option to the JVM options of the server.

### User-defined gc log file

A gc log file can be specified by using the "-Xloggc:" and "-Xverbosegclog:" option for Oracle and IBM systems respectively. JEUS provides an option that automatically appends a time stamp to a specified log file so that the file can be identified from a gc log file created whenever JEUS starts (default). If you do not want to append a time stamp, add a JVM option called -Djeus.logging.gclog.timestamp.on=false to a startup script.

## Log Directory Configuration

Edit domain.xml to set a log directory to save all logs created on the server (including rotation backup logs) as in the following example.

```
<servers>
  <server>
    <name>server1</name>
    <node-name>node1</node-name>
    <log-home>${LOG_HOME}/logs</log-home>
    ...
  </server>
</servers>
```

You can put the value of a system environment variable or system property in the log-home path. Here, the variable name must be a pattern such as \${ENV\_NAME}, and if an unregistered environment variable is used, the log is created in the default path.

### 8.2.2. Launcher Logger

A launcher is used to start a server. After a server is started using a script or a start command, the launcher executes, reads the configuration file, and starts the server. The log messages that are created by the launcher at server startup are recorded in the JeusLauncher.log file and the console. A launcher process usually terminates after the server starts and just leaves the server boot log through the launcher log. However, if the '-verbose' option is used to start a server, the launcher process does not terminate until the server shuts down and all log messages created while the server is running are saved through the launcher's console and file.

The launcher process separately logs messages that the server logger logs in order to handle situations where the server cannot perform logging while the server is booting. For example, if the server fails to start because of a configuration file error, or if the server starts but it stops before the server logger is initialized due to a failure, the log messages that can show the cause of the failure are not logged by the server. Therefore, the launcher records the server boot log using its own logger so that the user can handle startup errors by checking the launcher log. For more information

about the launcher, refer to "Launcher" in JEUS Server Guide.

The following is an example of a log message left by the launcher when the boot fails due to a configuration file error.

```
[2016.08.24 12:38:52][0] [launcher-1] [XmlValidationEventHandler] [FATAL_ERROR]
Invalid content was found starting with element <system-clustering-framework>
(of parent element <domain>). One of '{password-validator, admin-server-name}'
is expected.

[2016.08.24 12:38:52][0] [launcher-1] [SERVER-0522] An exception occurred while processing
[domain.xml].
<<__Exception__>>
jeus.xml.binding.JeusJAXBException: Unmarshalling the XML descriptor failed:
domain.xml
class org.xml.sax.SAXParseException :
cvc-complex-type.2.4.a: Invalid content was found starting with element
'system-clustering-framework'. One of '{"http://www.tmaxsoft.com/xml/ns/jeus":password-validator,
"http://www.tmaxsoft.com/xml/ns/jeus":admin-server-name}' is expected..
    at jeus.xml.binding.BindingHelper.getDescriptor(BindingHelper.java:96)
    at jeus.service.descriptor.DescriptorFile.getDeploymentDescriptor(DescriptorFile.java:210)
    at
jeus.service.descriptor.JEUSDomainDescriptorFile.getConfigDescriptor(JEUSDomainDescriptorFile.java:54
)
    at jeus.launcher.Launcher.initDomainType(Launcher.java:222)
    at jeus.launcher.Launcher.readDescriptor(Launcher.java:210)
    at jeus.launcher.Launcher.start(Launcher.java:105)
    at jeus.launcher.Launcher.main(Launcher.java:58)
```

### 8.2.3. Server Logger

The JEUS server logger logs messages that are created while a server is running. Each message contains activities on the server while it is running.

The default JEUS logger is called jeus logger, which is the top-level logger. The sub loggers of jeus logger are logged using jeus logger's handler. The loggers and handlers on the server can be dynamically added, deleted, or changed.

It is possible to change the format of log messages by using a format pattern. When there is no specific configuration applied, you can use the default format pattern provided by JEUS, which is '[%d{yyyy.MM.dd HH:mm:ss}][%l] [%J-%T] [%M-%N] %m'.

The following are the logging patterns of SimpleFormatter and SimpleMillisFormatter, which are formatters supported in previous versions.

- SimpleFormatter

```
[%d{yyyy.MM.dd HH:mm:ss}][%l] [%J-%T] [%M-%N] %m
```

- SimpleMillisFormatter



```
[%d{yyyy.MM.dd HH:mm:ss:SSS}][%l] [%J-%T] [%M-%N] %m
```

The following describes the format of the server logger.

- Log format

```
[%d{yyyy.MM.dd HH:mm:ss}] [%l] [%J-%T] [%M-%N] %m
```

Item	Description
%d{yyyy.MM.dd HH:mm:ss}	Displays the date of a log event. Enter the DATE_FORMAT inside '{}' to specify the format of the date you want to display. The DATE_FORMAT is formatted according to the JDK's SimpleDateFormat.
%l	Each log level of the log event is displayed as a number mapped to it. <ul style="list-style-type: none"><li>• 0 : SEVERE</li><li>• 1 : WARNING</li><li>• 2 : INFO</li><li>• 3 : CONFIG</li><li>• 4 : FINE</li><li>• 5 : FINER</li><li>• 6 : FINEST</li><li>• 7 : ALL</li></ul>
%J	Displays the process of logging the message. It represents the thread information.
%T	Displays the thread number that logs the message. Log messages with the same thread information are logged by the same thread.
%M	Displays the name of the module that outputs the log. Refer to <a href="#">Log Messages with Module Names</a> for the name of each module.
%N	Displays the message number of the output log event.
%m	Contains activities on the server while it is running.
%c	Displays the name of the logger that logs the message.



The log format of the Launcher process is the same as that of the server process.

The following is an example of the log messages output by JEUS server.

```
[2024.09.25 16:01:33][2] [ms1-1] [SERVER-0248] The JEUS server is STARTING.  
[2024.09.25 16:01:34][0] [ms1-1] [SERVER-0000] Version information - JEUS 9.1 (9.1.0.0).  
[2024.09.25 16:01:34][0] [ms1-1] [SERVER-0001] java.specification.version=[17],  
java.runtime.version=[17.0.2+8-86], vendor=[Oracle Corporation]
```

```
[2024.09.25 16:01:34][2] [ms1-1] [SERVER-0003] os.name=[linux], os.arch=[amd64], os.version=[4.18.0-408.el8.x86_64]
[2024.09.25 16:01:34][2] [ms1-1] [SERVER-0002] Domain=[domain1], Server=[ms1], baseport=[19736], pid=[17004]
[2024.09.25 16:01:34][2] [ms1-1] [SERVER-0004] The current system time zone : KST
[2024.09.25 16:01:34][2] [ms1-1] [SERVER-0571] All JEUS system properties have been confirmed.
[2024.09.25 16:01:34][2] [ms1-1] [SERVER-0568] Service address='0.0.0.0:19736', hostname='JEUS-PC', representation ip='192.168.1.105'
[2024.09.25 16:01:34][2] [ms1-1] [SERVER-0561] The default RMI export port = 19743 and bind address = JEUS-PC/192.168.1.105.
[2024.09.25 16:01:34][2] [ms1-1] [NET-0002] Beginning to listen to NonBlockingChannelAcceptor: 0.0.0.0:19736.
```

The previous example shows that the first message was output at 16:01:33 on September 25th, 2024. This message, server message number 0248, was logged by thread number 1 on the server named "ms1". It is the first message output for the server showing that JEUS server has initiated to boot.

Other messages are also logged by the same process, and the server module messages are also output. The previous log messages occurred while a server was booting up, and they include environment information such as the JEUS version, Java version, process ID, timezone, and network information.

JEUS provides [%d{yyyy.MM.dd HH:mm:ss}] [%l] [%j-%T] [%M-%N] %m as the default format pattern of the server logger. This is the format pattern that SimpleFormatter outputs in the previous version, and the format pattern of SimpleMillisFormatter, which was provided along with SimpleFormatter in the previous version, is [%d{yyyy.MM.dd HH:mm:ss:SSS}] [%l] [%j-%T] [%M-%N] %m.

The following is an example of the log messages that are created when you set the format pattern that outputs the same log format as the SimpleMillisFormatter. For more information about the formatter configuration, refer to [Logging Configuration](#).

```
[2024.09.25 16:01:33][2] [ms1-1] [SERVER-0248] The JEUS server is STARTING.
[2024.09.25 16:01:34:385][0] [ms1-1] [SERVER-0000] Version information - 9.1 (9.1.0.0).
[2024.09.25 16:01:34:385][0] [ms1-1] [SERVER-0001] java.specification.version=[17],
java.runtime.version=[17.0.2+8-86], vendor=[Oracle Corporation]
[2024.09.25 16:01:34:386][2] [ms1-1] [SERVER-0003] os.name=[linux], os.arch=[amd64],
os.version=[4.18.0-408.el8.x86_64]
[2024.09.25 16:01:34:387][2] [ms1-1] [SERVER-0002] Domain=[domain1], Server=[ms1], baseport=[19736], pid=[17004]
[2024.09.25 16:01:34:387][2] [ms1-1] [SERVER-0004] The current system time zone : KST
[2024.09.25 16:01:34:560][2] [ms1-1] [SERVER-0571] All JEUS system properties have been confirmed.
[2024.09.25 16:01:34:568][2] [ms1-1] [SERVER-0568] Service address='0.0.0.0:19736', hostname='JEUS-PC', representation ip='192.168.1.105'
[2024.09.25 16:01:34:580][2] [ms1-1] [SERVER-0561] The default RMI export port = 19743 and bind address = JEUS-PC/192.168.1.105.
[2024.09.25 16:01:35:030][2] [ms1-1] [NET-0002] Beginning to listen to NonBlockingChannelAcceptor: 0.0.0.0:19736.
```

## 8.2.4. Access Logger

This logger records all application requests processed by the server. It records information about

web applications requests. This section describes access loggers in web engines.

## Access Loggers in Web Engines

Access loggers in a web engine record all requests processed by the web engine. Access loggers specify the information accessed from the application and contents to record so that administrators can obtain the necessary information. After the web engine finishes processing all requests on the web engine, the access logger records the configured information.

Logs that are recorded by the access logger on the web engine usually support common log format. This format is commonly used in many areas such as Apache and provides a lot of information when analyzing access logs. Also, there are many tools available for analyzing common log formats that are helpful in analyzing access logs.



JEUS 6 does not support the common log format.

A virtual host can be configured for a web engine. When a virtual host is configured, separate access logs can be gained by configuring access loggers in each virtual host. The following is the name of the access log of a virtual host.

```
SERVER_HOME/logs/servlet/<virtual host name>/access.log
```

If an access log is recorded in the basic access log format of the web engine, the following logs will be in the access.log.

```
192.168.15.57 [29/Aug/2016:17:37:02 +0900] "GET /example/test1.jsp HTTP/1.1" 200 5 38
```

The above log means that the request to `/example/test1.jsp` from 192.168.15.57 successfully sent a 38-byte response at 29/Aug/2016:17:37:02 +0900. For more detailed log format, please refer to the documentation on common log formats.

For more information on changing the format of the recorded log or configuring other web engine access loggers, see "Access Log Settings" in JEUS Web Engine Guide.

## 8.2.5. User Logger

A user logger is provided for each JEUS server so that the developer does not need to use a separate logger. It is possible to use a user logger through `java.util.logging.logger` API of Java SE Logging API.

## 8.2.6. Logger List

The following describes loggers.

- EJB loggers

Division	Description
jeus.ejb.bean	EJB Home/Object stub logger
jeus.ejb.cluster	EJB cluster logger
jeus.ejb.compiler	EJB stub compiler logger
jeus.ejb.connector	MDB and resource adapter logger
jeus.ejb.container	EJB container logger
jeus.ejb.ejbserver	EJB engine logger
jeus.ejb.interop	EJB CORBA linkage logger
jeus.ejb.persistence	CMP logger
jeus.ejb.schema	EJB QL logger
jeus.ejb.timer	EJB Timer logger
jeus.ejb.transaction	EJB Transaction & Synchronization logger
jeus.ejb.webserver	EJB class FTP logger
jeus.ejb.util	Other logger

- JPA loggers

Division	Description
jeus.persistence	Top-level logger of the JPA module
jeus.persistence.provider	Top-level logger of TopLink Essentials (default provider)
jeus.persistence.container	Container logger for JPA

- Servlet loggers

Division	Description
jeus.servlet.common	Common servlet module logger
jeus.servlet.connection	Connector logger
jeus.servlet.connector	NIO connector logger
jeus.servlet.deployment	Deploy logger
jeus.servlet.engine	Main servlet processing logger
jeus.servlet.filter	Filter logger

Division	Description
jeus.servlet.jsp	JSP logger
jeus.servlet.listener	Servlet listener logger
jeus.servlet.loader	Class loader logger
jeus.servlet.property	Property logger
jeus.servlet.servlets	Logger for servlets that are provided by JEUS
jeus.servlet.util	Logger that is used in the utility
jeus.websocket	Logger related to websocket servers.
jeus.webserver	Logger that is used in the FTP class service

- Session manager loggers

Division	Description
jeus.session	Session Manager's top-level logger that is used in common
jeus.session.distributed	Distributed session manager logger
jeus.session.central	Centralized session manager logger
jeus.session.redis	Redis session manager logger
jeus.session.hazelcast	Hazelcast session manager logger

- Web service loggers

- The JAX-RPC/SAAJ loggers

Division	Description
jeus.webservices.client	jeus.webservices.client package (client invocation framework) logger
jeus.webservices.encoding	SOAP serialize/deserialize logger
jeus.webservices.message	SAAJ and SOAP message logger
jeus.webservices.wsdl	Logger for WSDL handling
jeus.webservices	Other web service loggers

- The UDDI logger

Division	Description
jeus.uddi.datastore	Database processing logger
jeus.uddi.function	UDDI API message processing logger
jeus.uddi.judy	Registry server engine logger
jeus.uddi.registry	Transport layer, request/response processing, and registry engine logger

- The WS-\* logger (based on JAX-RPC)

Division	Description
jeus.webservices.wss	ws-security logger

- Others

Division	Description
jeus.xml.binding.webservicesHelper	Logger for DD binding used in EWS (JSR109)

- Transaction logger

Division	Description
jeus.transaction	Logger that is generally used by the transaction manager
jeus.transaction.logging	Resource factory logger used for recovery
jeus.transaction.ots	OTS logger
jeus.transaction.recovery	Logger that records transaction recovery

- Security logger

Division	Description
jeus.security	JEUS security logger
jeus.security.impl.login	JEUS security login service logger
jeus.security.util	JEUS security utility logger

- Other key loggers

Division	Description
jeus.classloader	JEUS class loading logger
jeus.clustering	JEUS server cluster logger
jeus.config	Logger for changing dynamic configurations of JEUS
jeus.connector	Jakarta EE connector logger
jeus.converter	XML converter logger
jeus.ddinit	Deployment descriptor initializer logger
jeus.deploy	Application deploy logger
jeus.domain	Domain logger
jeus.filetransfer	Logger for configuration and application file transmission
jeus.io	JEUS network I/O library logger
jeus.jdbc	JDBC connection pool logger
jeus.jmx	JMX logger
jeus.jndi	JNDI logger
jeus.jnlp	JNLP logger
jeus.jtmax	JTmax logger
jeus.management	JMX MBean framework logger
jeus.net	JEUS network API logger
jeus.node	SSH node manager logger
jeus.nodemanager	Java node manager logger
jeus.rmi	JEUS RMI logger
jeus.scheduler	JEUS scheduler logger
jeus.service	JEUS service MBean logger
jeus.weld	JEUS CDI logger
jeus.logger.status	Logger for JEUS Asynchronous logger (server logger) exceptions

## 8.2.7. Log Messages with Module Names

Log messages in JEUS provide a lot of information. Log messages with information about modules have module names. This section describes these modules.

Module Name	Module Information
C_SESSION	Centralized session server
Connector	Connector
Console	Console command
Config	Synchronization of configuration files and related modules

Module Name	Module Information
CORBA	CORBA
CPOOL	Connection
D_SESSION	Distributed session server
Deploy	Application deployment
Domain	Domain
EJB	EJB engine
JDBC	JDBC
JMS	JMS engine
JMSC	JMS clustering
JMSF	JMS failover
JMX	JMX
JMXR	JMX remote
JNDI.Common	JNDI common
JNDI.Context	JNDI context
JNDI.Local	JNDI local client
JNDI.Remote	JNDI remote client
JNSS	JNDI server
JPA	JPA
JTMAX	JTmax
Launcher	JEUS launcher (launcher process)
Network	JEUS network
NodeManager	Node manager
OTS	OTS
SCF	SCF(JEUS System Clustering Framwork)
Scheduler	Scheduler
Secutiry	JEUS security
SERVER	Module related to starting, stopping, and monitoring a server
Session	Session server
TM	Transaction manager
TMRecovery	Recovery of the transaction manger
UDDI	UDDI
WEB	Servlet engine
WebT	WebT



Module Name	Module Information
WebtobLight	WebtoB
WSS	Web service security
WSVC	Web service

## 8.3. Logging Configuration

This section describes how to configure and customize logging in JEUS.

### 8.3.1. Checking Logger Information

The top-level logger, jeus logger, is created by default, but other loggers output log messages by using the handler of jeus logger when they are not configured. The file handler is used when jeus logger is not configured. If a server does not have any logger configuration, log messages created at server runtime is recorded in the default file. In this case, the log level is set to INFO, and jeus logger does not use the top-level logger's handler.



To check the log messages of a server in the console, use the '-verbose' option when starting the server. When the '-verbose' option is used, the launcher process does not terminate until the server shuts down so that it can output server log messages to the console.

The following is an example of using the console tool to check for information about jeus logger, the default logger on the server.

#### Using the Console Tool

You can check information about the jeus logger and its registered handlers by executing the **log-level** and **list-log-handlers** commands in the console tool as follows. For detailed information on how to use each command, see "Server Management Commands" in JEUS Reference Guide.

```
[MASTER]domain1.adminServer>log-level -server server1 jeus
The logger[jeus] information for the server [server1]
Information about the logger[jeus].
=====
Logger Name : jeus
Level : INFO
Use Parent Handlers : false

+-----+-----+-----+
|           Handler Name           | Handler Type | Handler Level |
+-----+-----+-----+
| jeus.util.logging.ConsoleHandler@1698156408 | ConsoleHandler | ALL           |
| FileHandler                       | FileHandler   | FINEST        |
+-----+-----+-----+
```

```

=====
[MASTER]domain1.adminServer>modify-logger -server server1 jeus
Show the current configuration.
The logger[jeus] information for the server [server1]
=====
+-----+-----+
|      Name      |      Value      |
+-----+-----+
| Level          | INFO            |
| Use Parent Handlers | false          |
| Formatter      | [%d{yyyy.MM.dd HH:mm:ss}] [%L] [%J-%T] [%M-%N] %m      |
+-----+-----+
=====

[MASTER]domain1.adminServer>list-log-handlers -server server1 jeus
List of Loggers
=====
+-----+-----+
| Handler Name    | FileHandler     |
| Handler Type    | FileHandlerType |
| Handler Level   | FINEST          |
| Filename        | JeusServer.log  |
| Enable Rotation | true            |
| Rotation Directory | ${SERVER_HOME}/logs |
| Valid Day       | 1               |
| Buffer Size     | 1024            |
| Append Logs     | true            |
+-----+-----+
=====

```

In the following cases, the console handler can be displayed when checking jeus logger information in jeusadmin.

- When '-verbose' option is used to start a server.
- When starting a server.

When the '-verbose' option is used to start a server, the console log is output through the launcher process. When the server is starting, log messages can also be displayed on the console screen through the launcher process.

### 8.3.2. Dynamically Configuring a Logger

Loggers or handlers can be dynamically added, deleted, or modified at runtime by using the console tool.

#### Using the Console Tool

Logging configurations can be dynamically changed by using the **add-logger**, **modify-logger**, and **remove-logger** commands in the console tool. For more information about these commands, see "Server Management Commands" in JEUS Reference Guide.

The currently configured logger information can be checked by using the **list-loggers** command.

```
[MASTER]domain1.adminServer>list-loggers server1
```

List of Loggers

Logger Name	Level	Use Parent Handlers	Filter	Formatter
jeus	INFO	false		[%d{yyyy.MM.dd HH:mm:ss}][%l] [%J-%T] [%M-%N] %m

```
[MASTER]domain1.adminServer>add-logger -server server1 jeus.ejb -level FINEST
```

Successfully performed the ADD operation for The logger for the server(server1)..  
Check the results using "list-loggers or add-logger".

```
[MASTER]domain1.adminServer>add-logger -server server1 jeus.ejb.clustering -level FINEST
```

Successfully performed the ADD operation for The logger for the server(server1)..  
Check the results using "list-loggers or add-logger".

```
[MASTER]domain1.adminServer>list-loggers server1
```

List of Loggers

Logger Name	Level	Use Parent Handlers	Filter	Formatter
jeus	INFO	false		[%d{yyyy.MM.dd HH:mm:ss}][%l] [%J-%T] [%M-%N] %m
jeus.ejb	FINEST	true		[%d{yyyy.MM.dd HH:mm:ss}][%l] [%J-%T] [%M-%N] %m
jeus.ejb.clu stering	FINEST	true		[%d{yyyy.MM.dd HH:mm:ss}][%l] [%J-%T] [%M-%N] %m

```
[MASTER]domain1.adminServer>modify-logger -server server1 jeus.ejb.clustering -level FINE
```

Successfully performed the MODIFY operation for The logger[jeus.ejb.clustering] information for the  
server [server1].

Check the results using "modify-logger".

```
[MASTER]domain1.adminServer>list-loggers server1
```

List of Loggers

Logger Name	Level	Use Parent Handlers	Filter	Formatter
jeus	INFO	false		[%d{yyyy.MM.dd HH:mm:ss}][%l] [%J-%T] [%M-%N] %m

jeus.ejb	FINEST	true		[%d{yyyy.MM.dd HH:mm:ss}}[%l] [%J-%T] [%M-%N] %m
jeus.ejb.clu stering	FINE	true		[%d{yyyy.MM.dd HH:mm:ss}}[%l] [%J-%T] [%M-%N] %m

```
[MASTER]domain1.adminServer>remove-logger -server server1 jeus.ejb
Successfully performed the REMOVE operation for The logger for the server(server1)..
Check the results using "list-loggers or remove-logger".
```

```
[MASTER]domain1.adminServer>list-loggers server1
```

```
List of Loggers
```

Logger Name	Level	Use Parent Handlers	Filter	Formatter
jeus	INFO	false		[%d{yyyy.MM.dd HH:mm:ss}}[%l] [%J-%T] [%M-%N] %m
jeus.ejb.clu stering	FINE	true		[%d{yyyy.MM.dd HH:mm:ss}}[%l] [%J-%T] [%M-%N] %m

### 8.3.3. Standard Output and Standard Error Log Format Configuration

JEUS enables the standard output and error to be output in the log format. By using the default format used in JEUS, standard output and standard error can be output in a format similar to jeus logger.

The following describes how to configure the standard output and error to the JEUS log format by using the console tool.

#### Using the Console Tool

The following is an example of using the console tool to configure the standard output and error to use the JEUS log format.

```
[MASTER]domain1.adminServer>modify-server server1
```

```
Shows the current configuration.
```

```
server (server1)
```

JVM Configs	-Xmx1024m -XX:MaxMetaspaceSize=512m
Action On Resource Leak	WARNING
Stdout to Raw Format	true

```
| MEJB | false |
| Class FTP | false |
| Server Log Home Directory | none |
+-----+-----+
=====
```

```
[MASTER]domain1.adminServer>modify-server server1 -logStdoutToRawFormat false
Successfully performed the MODIFY operation for server (server1).
Check the results using "list-servers server1 or modify-server server1".
```

```
[MASTER]domain1.adminServer>list-servers server1
List of Editable Servers
```

```
=====
+---+---+---+---+---+---+---+---+---+---+---+
| Ser | Base | Node | JVM | Action | Stdout | MEJB | Cla | Server | Type |
| ver | Listen | | Configs | On | to Raw | | ss | Log Home | |
| | Address | | | Resource | Format | | FTP | Directo | |
| | /Port | | | Leak | | | | ry | |
+---+---+---+---+---+---+---+---+---+---+---+
| ser | 0.0.0.0 | | -Xmx102 | Warning | false | fal | fal | none | ser | |
| ver1 | / 9836 | | 4m | | | se | se | | ver |
| | | | -XX:MaxM | | | | | | | |
| | | | etaspace | | | | | | | |
| | | | Size=512m | | | | | | | |
+---+---+---+---+---+---+---+---+---+---+---+
=====
```

The standard output uses the following format.

```
[%d{yyyy.MM.dd HH:mm:ss}] [%l] [%J-%T] [%M-%N] [STDOUT/STDERR] %m
```

Item	Description
%d{yyyy.MM.dd HH:mm:ss}	Displays the date of a log event. Enter the DATE_FORMAT inside '{ }' to specify the format of the date you want to display. The DATE_FORMAT is formatted according to the JDK's SimpleDateFormat.
%l	Each log level of the log event is displayed as a number mapped to it. <ul style="list-style-type: none"> <li>◦ 0 : SEVERE</li> <li>◦ 1 : WARNING</li> <li>◦ 2 : INFO</li> <li>◦ 3 : CONFIG</li> <li>◦ 4 : FINE</li> <li>◦ 5 : FINER</li> <li>◦ 6 : FINEST</li> <li>◦ 7 : ALL</li> </ul>
%j	Displays the process of logging the message. It represents the thread information.

Item	Description
%T	Displays the thread number that logs the message. Log messages with the same thread information are logged by the same thread.
%M	Displays the name of the module the outputs the log. Refer to <a href="#">Log Messages with Module Names</a> for the name of each module.
%N	Displays the message number of the output log event.
%m	<p>Message to be output through System.out or System.err. If it is a log for standard output or standard error, the message is output with prefixes such as [STDOUT] or [STDERR] in front of it.</p> <ul style="list-style-type: none"> <li>◦ STDOUT: When the message is standard output.</li> <li>◦ STDERR: When the message is standard error such as an exception.</li> <li>◦ UNKNOWN: Unknown.</li> </ul>
%c	Displays the name of the logger that logs the message.

### 8.3.4. Logger Settings

Loggers and handlers can be added, deleted, or modified by using the console tool and the changes can be applied dynamically during runtime without restarting the server.



The level and option to use the parent handler (use-parent-handlers) can be configured and applied dynamically to the logger. Only the level can be applied dynamically to the handler, and only the file handler can be added or modified by using console tool command.

### Adding Logger

You can add a logger or query the loggers in use using the console tool as follows:

```
[MASTER]domain1.adminServer>add-logger logger2 -level SEVERE -useParentHandlers false -server server1
Successfully performed the ADD operation for The logger for the server(server1)., but all changes
were non-dynamic. They will be applied after restarting.
Check the results using "list-loggers or add-logger".
[MASTER]domain1.adminServer>list-loggers server1
List of Loggers
=====
+-----+-----+-----+-----+-----+
| Logger Name | Level | Use Parent | Filter | Formatter |
|             |       | Handlers  |        |            |
+-----+-----+-----+-----+-----+
| jeus        | INFO  | false     |        | [%d{yyyy.MM.dd |
|             |       |           |        | HH:mm:ss}][%l] [%J-%T] |
|             |       |           |        | [%M-%N] %m |
+-----+-----+-----+-----+-----+
| logger2     | SEVERE | false    |        | [%d{yyyy.MM.dd |
```

				HH:mm:ss}}][%l] [%J-%T]	
				[%M-%N] %m	
+-----+	+-----+	+-----+	+-----+	+-----+	+-----+
=====	=====	=====	=====	=====	=====

The following describes the configuration items.

Item	Description
server-name	Name of the server to which to add the logger.
logger-name	Logger name.
log-level	<p>Logger level. Only messages at or below this level is output through the logger.</p> <ul style="list-style-type: none"> <li>◦ SEVERE</li> <li>◦ WARNING</li> <li>◦ INFO (default value)</li> <li>◦ CONFIG</li> <li>◦ FINE</li> <li>◦ FINER</li> <li>◦ FINEST</li> <li>◦ ALL</li> </ul>
use-parent-handlers	<p>Option to set a logger to use a handler of the parent logger as well as its own handler.</p> <p>Although the default value is true, it is set to false for jeus logger because the logger is the top-level logger of JEUS. If a logger's handler is not configured, this value must be set to true in order to be able to use the parent handlers.</p>
filter-class	Class used for process filtering before the logger sends a log message to the handler. The class must be in the JAR file of the 'lib/application' directory.
formatter-pattern	<p>Format Pattern used by the logger to format log messages before sending them to a handler.</p> <p>(Default: [%d{yyyy.MM.dd HH:mm:ss}}][%l] [%J-%T] [%M-%N] %m)</p>



The log levels FATAL, NOTICE, INFORMATION, and DEBUG used in JEUS 4x are not available in JEUS 7.

### Adding Handler

**Handlers** include the file handler, SMTP handler, socket handler, and user handler. The following are the configuration items of each handler.

## • File Handler

A file handler is a handler that outputs log messages to a file. It can be added using the jeusadmin command as follows:

```
[MASTER]domain1.adminServer>add-handler handler2 -server server1 -logger logger2 -level FINEST  
-filename logFile2.log -enable false -day 1  
Successfully performed the ADD operation for The handler for the logger(logger2) on the  
server(server1)., but all changes were non-dynamic. They will be applied after restarting.  
Check the results using "list-log-handlers or add-log-handler".
```

Item	Description
Name	Handler name. The name must be unique within each logger. If not set, a combination of the class name and hash code is used.
File Name	Name of the file to which the handler outputs log messages. If an absolute path is used, then files are created in that path.  Otherwise, the files are created in the path specified by each logger and messages are saved in them.
Level	Level of the messages that the handler outputs. When log messages that are output through the logger are forwarded to each handler used by the logger, log messages that correspond to the level of the handler are output by the handler.  The default value is FINEST and all log messages through the logger are output by the handler.
Log chown	Sets the owner and owner group of the log files. Enter the owner and owner group in order, separated by a comma (,).
Log Permission	Access permissions for the log file. Use 9 characters made up of r, w, x, and -. (This option is valid only for Unix OS)
Enable Rotation	Option for a handler to use the log file rotation function. The rotation function is enabled by default. (default value: true)
Rotation Count	Rotation count. Only applies when a handler uses the log file rotation. (Default Value: 10)
Valid Day	Used when a handler outputs files by date.  '_YYYYMMDD' is appended to the end of the file name.
Valid Hour	Used when a handler creates files based on time.  A valid-hour is a multiple of 24 (3, 6...) or a number whose remainder is a multiple of 24 (27, 30...). '_YYYYMMDD_HH' is appended to the end of the file name.
Valid Size	Used when a handler outputs files based on size. Only applies when a handler uses the log file rotation function.



Item	Description
Encoding	Encoding of handler's string output. The default value is system encoding.
Filter Class	Used when a handler outputs log messages that are filtered. As for the logger filter-class, a jar file with this class must be in the 'lib/application' directory.
Append	<p>Option to overwrite an existing file or append to it when outputting to a log file after server starts.</p> <ul style="list-style-type: none"> <li>◦ true: Append to the existing file. (Default value)</li> <li>◦ false: Depends on whether rotation is used. If rotation is used, then the existing log file is backed up and a new log file is created at server startup. Otherwise, the previous log file is deleted at server startup.</li> </ul>
Rotation Dir	<p>Used only when the handler uses log file rotation.</p> <p>You can specify the path using the system environment variables or system properties. For example, you can set the path as &lt;rotation-dir&gt;\${JEUS_HOME}/rotatedLog/\${jeus.server.name}&lt;/rotation-dir&gt;.</p>
Buffer Size	<p>Size of the buffer that is used when writing to a file. Logging performance improves as the buffer size increases, but more logs are lost if JEUS is abnormally terminated. (Default value: 1024, unit: kilobytes)</p> <p>When the accumulated log messages in the buffer are larger than the configured buffer size, the log messages are written to the file.</p>

## • SMTP Handler

The SMTP handler sends log messages by e-mail. A single log message is sent to a single e-mail.

To add a handler other than the file handler, you must set the **jeus.logging.useAsync** option to **false**, and you can add it by editing domain.xml as follows:

```
<smtp-handler>
  <name>smtphandler</name>
  <level>WARNING</level>
  <smtp-host-address>gw.tmaxsoft.com</smtp-host-address>
  <sender-id>sender@tmaxsoft.com</sender-id>
  <sender-password>1234567</sender-password>
  <from-address>sender@tmaxsoft.com</from-address>
  <to-address>receiver@tmaxsoft.com</to-address>
  <property>
    <key>mail.smtp.socketFactory.port</key>
    <value>465</value>
  </property>
  <property>
    <key>mail.smtp.socketFactory.class</key>
    <value>javax.net.ssl.SSLSocketFactory</value>
  </property>
  <property>
    <key>mail.smtp.auth</key>
    <value>true</value>
  </property>
</smtp-handler>
```

```

</property>
<send-for-all-messages>true</send-for-all-messages>
</smtp-handler>

```

Item	Description
Name	Name used when a handler is visible in the tool. The name must be unique within the logger. If not set, a combination of the class name and hash code is used.
Level	Level of the messages that the handler outputs. When log messages that are output through the logger are forwarded to each handler used by the logger, log messages that correspond to the level of the handler are output by the handler. The default value is FINEST which means that all log messages through the logger are output by each handler.
Smtp Host Address	Host address of the e-mail recipient.
From Address	E-mail sender address.
Sender ID	Sender ID.
Sender Password	Sender password. To encrypt and save it, use the format "{algorithm to be encrypted) encrypted algorithm".
To Address	Address of the e-mail recipient.
Property	SMTP property for a specific mail server.
Send For All Messages	Option to use the SMTP handler to send all messages. If set to false, only the e-mail sent messages are sent using this handler. This configuration is currently only available for user loggers. (Default value: false)
Encoding	Encoding of handler's string output. The default value is system encoding.
Filter Class	Class used for process filtering before the logger sends a log message to the handler. The class must be in the JAR file of the 'lib/application' directory.
Cc Address	Addresses of the recipients to receive a carbon copy of the e-mail.
Bcc Address	Addresses of the recipients who will receive a copy of the e-mail and whose addresses will be concealed.

## • Socket Handler

The socket handler sends log messages to the socket.

Item	Description
Name	Specifies the name that will be used when a handler is shown on the tool. The name must be unique within one logger. If the name is not specified, use a name which combines the class name and the hash code.

Item	Description
Level	Level of the messages that the handler outputs. When log messages that are output through the logger are forwarded to each handler used by the logger, log messages that correspond to the level of the handler are output by the handler. The default value is FINEST which means that all log messages through the logger are output by each handler.
Address	IP address of the server that the handler accesses.
Port	Port number of the machine that the handler accesses.
Encoding	Encoding of handler's string output. The default value is system encoding.
Filter Class	Class used for process filtering before the logger sends a log message to the handler. The class must be in the JAR file of the 'lib/application' directory.

#### • User Handler

The user handler specifies a user-defined handler class.

Item	Description
Name	Specifies the name that will be used when a handler is shown on the tool. The name must be unique within one logger. If the name is not specified, use a name which combines the class name and the hash code.
Level	Level of the messages that the handler outputs. When log messages that are output through the logger are forwarded to each handler used by the logger, log messages that correspond to the level of the handler are output by the handler. The default value is FINEST and all log messages through the logger are output by the handler.
Handler Class	User-defined handler class. This class must be in the JAR file of 'lib/application' directory. The class must inherit the java.util.logging.Handler class of the logging API and implement the jeus.util.logging.JeusHandler interface.
Handler Property	Properties to include in the map objects that are used in the setProperty() method of jeus.util.logging.JeusHandler.
Formatter Pattern	Format of the log to be output. (Default: [%d{yyyy.MM.dd HH:mm:ss}] [%I] [%]-%T] [%M-%N] %m)
Formatter Property	Properties to include in the map objects that are used in the setProperty() method of jeus.util.logging.JeusFormatter.
Encoding	Encoding of handler's string output. The default value is system encoding.
Filter Class	Class used for process filtering before the logger sends a log message to the handler. The class must be in the JAR file of the 'lib/application' directory.

### 8.3.5. Log File Rotation Configuration

When JEUS starts at a configured time or when the file size exceeds its limit, the current log file is

automatically backed up under a different name and the file rotation allows new logs to be written to the existing file.

## Using the Console Tool

Log file rotation settings for the file handler can be configured using the **modify-log-handler** command. There are three options available for rotation settings in the console tool: **-hour**, **-size**, and **-day**. The default is day 1.

If none of the three conditions for rotation are set, the name of the file being logged will be changed and backed up at 00:00 every day, and files logged thereafter will be newly logged to the previously logged file.

The following is an example of modifying the log rotation by using jeusadmin.

```
[MASTER]domain1.adminServer>modify-log-handler fileHandler -server adminServer -logger jeus -day 2
Successfully performed the MODIFY operation for The handler(fileHandler) for the logger(jeus) in
server (adminServer), but all changes were non-dynamic. They will be applied after restarting.
Check the results using "modify-log-handler".
```

### 8.3.6. Property Configuration

The following describes the properties required for configuration.

- System properties

When the logger is configured in a standalone client, the log level can be configured using system properties.

- Logging properties file

Logging can be configured in the Java logging property (logging.properties) file.

(Default path: JEUS\_HOME/bin/logging.properties)

```
-Djava.util.logging.config.file = <property file path>
```



The jeuslogging.properties file used in JEUS 6 cannot be used in JEUS 7.

## Level Configuration Priority

The following is the priority of level configurations apart from the dynamic configurations set by using JEUS tool.

1. System properties

2. Java logging.properties file
3. Logging configuration in domain.xml



The log level configured in the handler does not override this priority. Since logs are output through the handler, the handler has the highest priority. Note that the default log level of the handler is FINEST.

# Appendix A: JEUS Server Ports

This appendix defines the ports used by JEUS. When you are configuring a firewall, refer to this appendix when configuring a firewall.

## A.1. Server Ports

- BASEPORT

<b>Description</b>	Basic service port that JEUS servers need for JNDI services and operation. This can be configured in the base listener.
<b>Base Port</b>	9736

- COS Naming Server Port

<b>Description</b>	Used for COS Naming service
<b>Base Port</b>	BASEPORT + 4 (e.g. 9740)

- ORB Port

<b>Description</b>	IIOP Port
<b>Base Port</b>	BASEPORT + 1

- ORB SSL Port

<b>Description</b>	IIOP SSL Port
<b>Base Port</b>	BASEPORT + 2

- ORB SSL Mutual Authorization Port

<b>Description</b>	IIOP mutual authentication port
<b>Base Port</b>	BASEPORT + 3

- RMI Port

<b>Description</b>	RMI port that is used by functions using RMI such as EJB, schedulers, etc.
<b>Base Port</b>	BASEPORT + 7

# Appendix B: JDBC Data Source Configuration Examples

This appendix provides examples of configuring data sources of major database vendors.

## B.1. Overview

Configurations of the following JDBC data sources are shown in the examples of this appendix.

- Oracle Thin
- Oracle OCI
- DB2 Type4(JCC)
- DB2 Type2(JCC)
- Sybase jConnect 5.x, 6.x
- Microsoft SQL Server 2005 Type4
- Informix Type4
- Tibero Type4
- MySQL 5.x Type4

You can configure a data source by referring to this appendix. For more information on how to configure data sources, refer to [Data Source Configuration](#).

## B.2. Oracle Thin (Type4) Configuration Example

### B.2.1. Oracle Thin Connection Pool Data Source

The following example shows how to configure an Oracle Thin connection pool data source.

Oracle Thin Connection Pool Data Source: <domain.xml>

```
<domain>
  . . .
  <resources>
    <data-source>
      <database>
        <data-source-id>ora_thin_cpds</data-source-id>
        <export-name>ora_thin_cpds</export-name>
        <data-source-class-name>
          oracle.jdbc.pool.OracleConnectionPoolDataSource
        </data-source-class-name>
        <data-source-type>
          ConnectionPoolDataSource
        </data-source-type>
      </database>
    </data-source>
  </resources>
</domain>
```

```

        <vendor>oracle</vendor>
        <server-name>192.168.1.2</server-name>
        <port-number>1521</port-number>
        <database-name>orcl</database-name>
        <user>scott</user>
        <password>tiger</password>
        <property>
            <name>driverType</name>
            <type>java.lang.String</type>
            <value>thin</value>
        </property>
        <connection-pool>
            . . .
        </connection-pool>
    </database>
    . . .
</data-source>
. . .
</resources>
</domain>

```

## B.2.2. Oracle Thin XA Data Source

The following example shows how to configure an Oracle Thin XA data source.

Oracle Thin XA Data Source: <domain.xml>

```

<domain>
    . . .
    <resources>
        <data-source>
            <database>
                <data-source-id>ora_thin_xads</data-source-id>
                <export-name>ora_thin_xads</export-name>
                <data-source-class-name>
                    oracle.jdbc.xa.client.OracleXADataSource
                </data-source-class-name>
                <data-source-type>
                    XADataSource
                </data-source-type>
                <vendor>oracle</vendor>
                <server-name>192.168.1.2</server-name>
                <port-number>1521</port-number>
                <database-name>orcl</database-name>
                <user>scott</user>
                <password>tiger</password>
                <property>
                    <name>driverType</name>
                    <type>java.lang.String</type>
                    <value>thin</value>
                </property>
                <connection-pool>
                    . . .
                </connection-pool>
            </database>
            . . .
        </data-source>
    </resources>
</domain>

```



```

        </data-source>
        . . .
    </resources>
</domain>

```

### B.2.3. Configuring java.util.Properties File with Oracle ASO

This section describes how to configure java.util.Properties with Oracle ASO.

Using the <property> element, set <type> to java.util.Properties and <value> as follows:

```
[property name 1]=[value 1], [property name 2]=[value 2]
```

Example of java.util.Properties Configuration with Oracle ASO: <domain.xml>

```

<domain>
    . . .
    <resources>
        <data-source>
            <database>
                <data-source-id>oracle_CPDS1</data-source-id>
                <export-name>oracle_CPDS1</export-name>
                <data-source-class-name>
                    oracle.jdbc.pool.OracleConnectionPoolDataSource
                </data-source-class-name>
                <data-source-type>
                    ConnectionPoolDataSource
                </data-source-type>
                <vendor>oracle</vendor>
                <server-name>192.168.1.2</server-name>
                <port-number>1521</port-number>
                <database-name>ora9</database-name>
                <user>scott</user>
                <password>tiger</password>
                <property>
                    <name>driverType</name>
                    <type>java.lang.String</type>
                    <value>thin</value>
                </property>
                <property>
                    <name>ConnectionProperties</name>
                    <type>java.util.Properties</type>
                    <value>
                        oracle.net.encryption_client=xxxx,
                        oracle.net.encryption_types_client=(3DES168,3DES112),
                        oracle.net.crypto_checksum_client=xxxx,
                        oracle.net.crypto_checksum_types_client=(MD5)
                    </value>
                </property>
                <connection-pool>
                    . . .
                </connection-pool>
            </database>
        . . .
    </resources>
</domain>

```

```

        </data-source>
        . . .
    </resources>
</domain>

```

## B.3. Oracle OCI (Type2) Configuration Example

### B.3.1. Oracle OCI Connection Pool Data Source

To use the Oracle OCI driver, configure the driver's native library path by using the '-Djava.library.path' option when starting JEUS.

Oracle OCI Connection Pool Data Source: <domain.xml>

```

<domain>
    . . .
    <resources>
        <data-source>
            <database>
                <data-source-id>ora_oci_cpds</data-source-id>
                <export-name>ora_oci_cpds</export-name>
                <data-source-class-name>
                    oracle.jdbc.pool.OracleConnectionPoolDataSource
                </data-source-class-name>
                <data-source-type>
                    ConnectionPoolDataSource
                </data-source-type>
                <vendor>oracle</vendor>
                <server-name>192.168.1.2</server-name>
                <port-number>1521</port-number>
                <database-name>ora9</database-name>
                <user>scott</user>
                <password>tiger</password>
                <property>
                    <name>driverType</name>
                    <type>java.lang.String</type>
                    <value>oci</value>
                </property>
                <property>
                    <name>TNSEntryName</name>
                    <type>java.lang.String</type>
                    <value>ORCL</value>
                </property>
                <connection-pool>
                    . . .
                </connection-pool>
            </database>
            . . .
        </data-source>
        . . .
    </resources>
</domain>

```

## B.4. DB2 Configuration Example

### B.4.1. DB2 Type4 (JCC) Connection Pool Data Source

The following example shows how to configure a DB2 Type4 connection pool data source.

DB2 Type4 (JCC) Connection Pool Data Source: <domain.xml>

```
<domain>
  . . .
  <resources>
    <data-source>
      <database>
        <data-source-id>db2_type4_cpsd</data-source-id>
        <export-name>db2_type4_cpds</export-name>
        <data-source-class-name>
          com.ibm.db2.jcc.DB2ConnectionPoolDataSource
        </data-source-class-name>
        <data-source-type>
          ConnectionPoolDataSource
        </data-source-type>
        <vendor>db2</vendor>
        <server-name>192.168.1.1</server-name>
        <port-number>50000</port-number>
        <database-name>TEST1</database-name>
        <user>db2inst1</user>
        <password>password</password>
        <property>
          <name>driverType</name>
          <type>java.lang.Integer</type>
          <value>4</value>
        </property>
        <connection-pool>
          . . .
        </connection-pool>
      </database>
    </data-source>
  </resources>
</domain>
```

### B.4.2. DB2 Type4 (JCC) XA Data Source

The following example shows how to configure a DB2 Type4 XA data source.

DB2 Type4 (JCC) XA Data Source: <domain.xml>

```
<domain>
  . . .
  <resources>
    <data-source>
      <database>
        <data-source-id>db2_type2_cpsd</data-source-id>
```

```

        <export-name>db2_type2_cpds</export-name>
        <data-source-class-name>
            com.ibm.db2.jdbc.DB2XADataSource
        </data-source-class-name>
        <data-source-type>
            XADataSource
        </data-source-type>
        <vendor>db2</vendor>
        <server-name>192.168.1.1</server-name>
        <port-number>50000</port-number>
        <database-name>TEST1</database-name>
        <user>db2inst1</user>
        <password>password</password>
        <connection-pool>
            . . .
        </connection-pool>
    </database>
    . . .
</data-source>
. . .
</resources>
</domain>

```

### B.4.3. DB2 Type2 (JCC) XA Data Source

To use a DB2 Type 2 driver, configure the native library path of the DB2 client in the system library path, or use the '-Djava.library.path' option when starting JEUS after installing the DB2 client and configuring the DB Alias. The following example shows that the database name is the DB Alias, and the server address and port number of DB2 are not needed.

B2 Type2 (JCC) XA Data Source: <domain.xml>

```

<domain>
    . . .
    <resources>
        <data-source>
            <database>
                <data-source-id>db2_type2_xads</data-source-id>
                <export-name>db2_type2_xads</export-name>
                <data-source-class-name>
                    com.ibm.db2.jdbc.DB2XADataSource
                </data-source-class-name>
                <data-source-type>
                    XADataSource
                </data-source-type>
                <vendor>db2</vendor>
                <server-name>192.168.1.1</server-name>
                <port-number>50000</port-number>
                <database-name>TEST1</database-name>
                <user>db2inst1</user>
                <password>password</password>
                <connection-pool>
                    . . .
                </connection-pool>
            </database>
            . . .
        </data-source>
    </resources>
</domain>

```

```

        </data-source>
        . . .
    </resources>
</domain>

```

## B.5. Sybase Configuration Example

### B.5.1. Sybase jConnect 5.x Connection Pool Data Source

The following example shows how to configure a Sybase jConnect 5.x connection pool data source.

Sybase jConnect 5.x Connection Pool Data Source: <domain.xml>

```

<domain>
    . . .
    <resources>
        <data-source>
            <database>
                <data-source-id>syb_jconn5_cpds</data-source-id>
                <export-name>syb_jconn5_cpds</export-name>
                <data-source-class-name>
                    com.sybase.jdbc2.jdbc.SybConnectionPoolDataSource
                </data-source-class-name>
                <data-source-type>
                    ConnectionPoolDataSource
                </data-source-type>
                <vendor>sybase</vendor>
                <server-name>192.168.1.1</server-name>
                <port-number>5000</port-number>
                <database-name>testdb</database-name>
                <user>sa</user>
                <password>password</password>
                <property>
                    <name>networkProtocol</name>
                    <type>java.lang.String</type>
                    <value>Tds</value>
                </property>
                <connection-pool>
                    . . .
                </connection-pool>
            </database>
            . . .
        </data-source>
        . . .
    </resources>
</domain>

```

### B.5.2. Sybase jConnect 6.x XA Data Source

The following example shows how to configure a Sybase jConnect 6.x XA data source.

```
<domain>
  . . .
  <resources>
    <data-source>
      <database>
        <data-source-id>syb_jconn6_xads</data-source-id>
        <export-name>syb_jconn6_xads</export-name>
        <data-source-class-name>
          com.sybase.jdbc2.jdbc.SybConnectionPoolDataSource
        </data-source-class-name>
        <data-source-type>
          ConnectionPoolDataSource
        </data-source-type>
        <vendor>sybase</vendor>
        <server-name>192.168.1.1</server-name>
        <port-number>5000</port-number>
        <database-name>testdb</database-name>
        <user>sa</user>
        <password>password</password>
        <property>
          <name>networkProtocol</name>
          <type>java.lang.String</type>
          <value>Tds</value>
        </property>
        <connection-pool>
          . . .
        </connection-pool>
      </database>
    </data-source>
  </resources>
</domain>
```

## B.6. Microsoft SQL Server Configuration Example

### B.6.1. Microsoft SQL Server 2005 Connection Pool Data Source

The following example shows how to configure a Microsoft SQL Server 2005 connection pool.

Microsoft SQL Server 2005 Connection Pool Data Source: <domain.xml>

```
<domain>
  . . .
  <resources>
    <data-source>
      <database>
        <data-source-id>mssql_2005_cpds</data-source-id>
        <export-name>mssql_2005_cpds</export-name>
        <data-source-class-name>
          com.microsoft.sqlserver.jdbc.SQLServerConnectionPoolDataSource
        </data-source-class-name>
        <data-source-type>
```

```

        ConnectionPoolDataSource
    </data-source-type>
    <vendor>mssql</vendor>
    <server-name>192.168.1.1</server-name>
    <port-number>1411</port-number>
    <database-name>testdb</database-name>
    <user>jeusdb1</user>
    <password>password</password>
    <connection-pool>
        . . .
    </connection-pool>
</database>
. . .
</data-source>
. . .
</resources>
</domain>

```



ODBC must be configured before configuring JDBC.

## B.7. Informix Configuration Example

### B.7.1. Informix Connection Pool Data Source

The following example shows how to configure an Informix connection pool data source.

Informix Connection Pool Data Source: <domain.xml>

```

<domain>
    . . .
    <resources>
        <data-source>
            <database>
                <data-source-id>infomix_cpds</data-source-id>
                <export-name>infomix_cpds</export-name>
                <data-source-class-name>
                    com.informix.jdbcx.IfxCConnectionPoolDataSource
                </data-source-class-name>
                <data-source-type>
                    ConnectionPoolDataSource
                </data-source-type>
                <vendor>informix</vendor>
                <server-name>192.168.1.1</server-name>
                <port-number>2002</port-number>
                <database-name>skynps</database-name>
                <user>informix</user>
                <password>password</password>
                <property>
                    <name>IfxIFXHOST</name>
                    <type>java.lang.String</type>
                    <value>192.168.1.43</value>
                </property>
            </database>
        </data-source>
    </resources>
</domain>

```

```

        <connection-pool>
            . . .
        </connection-pool>
    </database>
    . . .
</data-source>
. . .
</resources>
</domain>

```

## B.8. Tibero Configuration Example

### B.8.1. Tibero Connection Pool Data Source

The following example shows how to configure a Tibero connection pool data source.

Tibero Connection Pool Data Source: <domain.xml>

```

<domain>
    . . .
    <resources>
        <data-source>
            <database>
                <data-source-id>tibero_cpds</data-source-id>
                <export-name>tibero_cpds</export-name>
                <data-source-class-name>
                    com.tmax.tibero.jdbc.ext.TbConnectionPoolDataSource
                </data-source-class-name>
                <data-source-type>
                    ConnectionPoolDataSource
                </data-source-type>
                <vendor>tibero</vendor>
                <server-name>192.168.1.1</server-name>
                <port-number>8629</port-number>
                <database-name>testdb</database-name>
                <user>jeusdb1</user>
                <password>password</password>
                <property>
                    <name>driverType</name>
                    <type>java.lang.String</type>
                    <value>thin</value>
                </property>
                <connection-pool>
                    . . .
                </connection-pool>
            </database>
        </data-source>
    </resources>
</domain>

```



## B.9. MySQL 5.x Configuration Example

### B.9.1. MySQL Connector/J Connection Pool Data Source

The following example shows how to configure a MySQL Connector/J connection pool data source.

MySQL Connector/J Connection Pool Data Source: <domain.xml>

```
<domain>
  . . .
  <resources>
    <data-source>
      <database>
        <data-source-id>mysql_cpds</data-source-id>
        <export-name>mysql_cpds</export-name>
        <data-source-class-name>
          com.mysql.jdbc.jdbc2.optional.MysqlConnectionPoolDataSource
        </data-source-class-name>
        <data-source-type>
          ConnectionPoolDataSource
        </data-source-type>
        <vendor>mysql</vendor>
        <server-name>192.168.1.1</server-name>
        <port-number>3306</port-number>
        <database-name>testdb</database-name>
        <user>tester</user>
        <password>password</password>
        <connection-pool>
          . . .
        </connection-pool>
      </database>
    </data-source>
  </resources>
</domain>
```