

uCosminexus Application Server

Common Container Functionality Guide

3020-3-Y07-10(E)

■ Relevant program products

See the manual *uCosminexus Application Server Overview*.

■ Export restrictions

If you export this product, please check all restrictions (for example, Japan's Foreign Exchange and Foreign Trade Law, and USA export control laws and regulations), and carry out all required procedures.

If you require more information or clarification, please contact your Hitachi sales representative.

■ Trademarks

Active Directory is either a registered trademark or trademark of Microsoft Corporation in the United States and/or other countries.

AIX is a trademark of International Business Machines Corporation in the United States, other countries, or both.

AX2000 is a product name of A10 Networks, Inc.

F5, F5 Networks, BIG-IP and iControl are trademarks or registered trademarks of F5 Networks, Inc. in the U.S. and certain other countries.

All Borland brand and product names are trademarks or registered trademarks of Borland Software Corporation in the United States and other countries.

BSAFE is a registered trademark or a trademark of EMC Corporation in the United States and/or other countries.

CORBA is a registered trademark of Object Management Group, Inc. in the United States.

HP-UX is a product name of Hewlett-Packard Development Company, L.P. in the U.S. and other countries.

IIOp is a trademark of Object Management Group, Inc. in the United States.

Linux(R) is the registered trademark of Linus Torvalds in the U.S. and other countries.

Microsoft is either a registered trademark or trademark of Microsoft Corporation in the United States and/or other countries.

OMG, CORBA, IIOp, UML, Unified Modeling Language, MDA and Model Driven Architecture are either registered trademarks or trademarks of Object Management Group, Inc. in the United States and/or other countries.

Oracle and Java are registered trademarks of Oracle Corporation and/or its affiliates.

Red Hat is a trademark or a registered trademark of Red Hat Inc. in the United States and other countries.

RSA is a registered trademark or a trademark of EMC Corporation in the United States and/or other countries.

SOAP is an XML-based protocol for sending messages and making remote procedure calls in a distributed environment.

UNIX is a registered trademark of The Open Group in the United States and other countries.

VisiBroker is a trademark or registered trademark of Micro Focus IP Development Limited or its subsidiaries or affiliated companies in the United Kingdom, United States and other countries.

Windows, Windows Server, and Windows Vista are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

X/Open is a registered trademark of The Open Group in the U.K. and other countries.

Eclipse is an open development platform for tools integration provided by Eclipse Foundation, Inc., an open source community for development tool providers.

This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>).

The other company names and product names are either trademarks or registered trademarks of the respective companies. Throughout this document Hitachi has attempted to distinguish trademarks from descriptive terms by writing the name with the capitalization used by the manufacturer, or by writing the name with initial capital letters. Hitachi cannot attest to the accuracy of this information. Use of a trademark in this document should not be regarded as affecting the validity of the trademark.

■ Microsoft product name abbreviations

This manual uses the following abbreviations for Microsoft product names:

Abbreviation		Full name or meaning
Active Directory		Microsoft(R) Active Directory(R)
Microsoft IIS	Microsoft IIS 7.0	Microsoft(R) Internet Information Services 7.0
	Microsoft IIS 7.5	Microsoft(R) Internet Information Services 7.5
SQL Server	SQL Server 2005	Microsoft(R) SQL Server 2005
	SQL Server 2008	Microsoft(R) SQL Server 2008

Abbreviation		Full name or meaning
SQL Server	SQL Server 2008	Microsoft(R) SQL Server 2008 R2
	SQL Server 2012	Microsoft(R) SQL Server 2012
JDBC driver for SQL Server	SQL Server JDBC Driver	Microsoft(R) SQL Server JDBC Driver 3.0
		Microsoft(R) JDBC Driver 4.0 for SQL Server
Windows	Windows Server 2008	Windows Server 2008 x86
		Microsoft(R) Windows Server(R) 2008 Standard 32-bit
		Microsoft(R) Windows Server(R) 2008 Enterprise 32-bit
		Windows Server 2008 x64
		Microsoft(R) Windows Server(R) 2008 Standard
		Microsoft(R) Windows Server(R) 2008 Enterprise
	Windows Server 2008 R2	Microsoft(R) Windows Server(R) 2008 R2 Standard
		Microsoft(R) Windows Server(R) 2008 R2 Enterprise
		Microsoft(R) Windows Server(R) 2008 R2 Datacenter
	Windows Server 2012	Windows Server 2012 Standard
		Microsoft(R) Windows Server(R) 2012 Standard
	Windows Server 2012 Datacenter	
	Microsoft(R) Windows Server(R) 2012 Datacenter	
	Windows XP	Microsoft(R) Windows(R) XP Professional Operating System
	Windows Vista	Windows Vista Business
		Microsoft(R) Windows Vista(R) Business (32-bit)
Windows Vista Enterprise		
Microsoft(R) Windows Vista(R) Enterprise (32-bit)		
Windows Vista Ultimate		
Microsoft(R) Windows Vista(R) Ultimate (32-bit)		
Windows 7	Windows 7 x86	Microsoft(R) Windows(R) 7 Professional (32-bit)
		Microsoft(R) Windows(R) 7 Enterprise (32-bit)
		Microsoft(R) Windows(R) 7 Ultimate (32-bit)
	Windows 7 x64	Microsoft(R) Windows(R) 7 Professional (64-bit)
		Microsoft(R) Windows(R) 7 Enterprise (64-bit)
		Microsoft(R) Windows(R) 7 Ultimate (64-bit)
Windows 8	Windows 8 x86	Windows(R) 8 Pro (32-bit)
		Windows(R) 8 Enterprise (32-bit)
	Windows 8 x64	Windows(R) 8 Pro (64-bit)
		Windows(R) 8 Enterprise (64-bit)

Note that Windows 32 bit and Windows 64 bit are sometimes respectively referred to as Windows x86 and Windows x64.

■ Restrictions

Information in this document is subject to change without notice and does not represent a commitment on the part of Hitachi. The software described in this manual is furnished according to a license agreement with Hitachi. The license agreement contains all of the terms and conditions governing your use of the software and documentation, including all warranty rights, limitations of liability, and disclaimers of warranty.

Material contained in this document may describe Hitachi products not available or features not available in your country.

No part of this material may be reproduced in any form or by any means without permission in writing from the publisher.

Printed in Japan.

■ Issued

Aug. 2013: 3020-3-Y07-10(E)

■ Copyright

All Rights Reserved. Copyright (C) 2013, Hitachi, Ltd.

Summary of amendments

The following table lists changes in the manual 3020-3-Y07-10(E) for uCosminexus Application Server 09-50, uCosminexus Application Server(64) 09-50, uCosminexus Client 09-50, uCosminexus Developer 09-50, uCosminexus Service Architect 09-50, uCosminexus Service Platform 09-50, uCosminexus Service Platform(64) 09-50 and product changes related to the manual:

Changes	Location
A description is added for the Bean Validation functionality, and implementation methods and actions to be taken when an error occurs are specified.	<i>1.2.9, Chapter 10</i>
The functionality is added for deploying WAR applications, configured only with WAR files, on J2EE servers.	<i>2.4.3, 11.3.3, 11.3.4, 11.3.6, 11.4.6, 12.3, 12.5.2, 13.2, 13.4.4, 13.9</i>
The characters that can be used with the Portable Global JNDI name are added.	<i>2.4.3</i>
The RAR file for DB Connector, common to the SQL Server versions and JDBC driver for SQL Server, are added. <ul style="list-style-type: none"> • RAR file for DB Connector: <code>DBConnector_SQLServer_CP.rar</code> • JDBC driver for SQL Server: <code>SQL Server JDBC Driver</code> <p>With this, the description for the RAR file for DB Connector and JDBC driver for SQL Server is changed or deleted corresponding to the version.</p>	<i>3.3.2, 3.4.3, 3.6.4, 3.6.7</i>
A description is added for the J2EE components and functionality available for database connection.	<i>3.6.2</i>
HiRDB Type4 JDBC Driver and SQL Server JDBC driver compliant with the JDBC 4.0 specifications are now supported with DB Connector, and the JDBC 2.0 specifications are no longer supported.	<i>3.6.3</i>
SQL Server 2012 is now supported. Accordingly, the notes for each SQL Server version are added.	<i>3.6.7</i>
The SMTPS provider is added to the providers available with JavaMail. Also, the property to output a message when the <code>javax.mail.Transport</code> object is obtained is added to the session properties specific to the SMTP provider and SMTPS provider.	<i>8.1, 8.2, 8.2.1, 8.3</i>
Whether to read the library JAR annotations can now be selected with an option.	<i>12.3</i>
The annotations that cannot be updated with the reload functionality are added: <ul style="list-style-type: none"> • <code>@WebEndpoint</code> • <code>@WebServiceRef</code> 	<i>12.6.1</i>
The description of notes has been moved from Release Notes.	<i>3.3.13, 3.6.3, 3.6.6, 3.6.7, 3.14.4, 3.16.10, 4.4, 4.14.1, 4.15.2, 7.8.2, 7.10.1, 7.19, 8.3, 12.5.1, 13.2, 13.4.4, 13.8.13, 14.6, 15.2.1, 15.2.2, 15.2.3, 15.3, Appendix E</i>

In addition to the above changes, minor editorial corrections have been made.

Preface

For details on the prerequisites before reading this manual, see the *uCosminexus Application Server Overview*.

■ Non-supported functionality

Some functionality described in this manual is not supported. Non-supported functionality includes:

- Audit log functionality
- Compatibility functionality
- Cosminexus Component Transaction Monitor
- Cosminexus DABroker Library
- Cosminexus Reliable Messaging
- Cosminexus TPBroker and VisiBroker
- Cosminexus Web Service - Security
- Cosminexus XML Security - Core functionality
- JP1 linkage functionality
- Management portal functionality
- Migration functionality
- SOAP applications complying with specifications other than JAX-WS 2.1
- uCosminexus OpenTP1 linkage functionality
- Virtualized system functionality
- XML Processor high-speed parse support functionality

■ Non-supported compatibility functionality

"Compatibility functionality" in the above list refers to the following functionality:

- Basic mode
- Check of JSP source compliance (cjsp2java) with JSP1.1 and JSP1.2 specifications
- Database connection using Cosminexus DABroker Library
- EJB client application log subdirectory exclusive mode
- J2EE application test functionality
- Memory session failover functionality
- Servlet engine mode
- Simple Web server functionality
- Switching multiple existing execution environments
- Using EJB 2.1 and Servlet 2.4 annotation

Contents

1	Application Server Functionality	1
1.1	Classification of functionality	2
1.1.1	Functionality serving as an execution platform for applications	4
1.1.2	Functionality for operating and maintaining the execution platform of applications	5
1.1.3	Correspondence between the functionality and manuals	5
1.2	Correspondence between the purpose of a system and the functionality	8
1.2.1	Naming management functionality	8
1.2.2	Functionality for managing resource connections and transactions	9
1.2.3	Functionality for invoking Application Server from OpenTP1 (TP1 inbound integrated functionality)	12
1.2.4	JPA functionality used with Application Server	13
1.2.5	Cosminexus JPA Provider functionality	13
1.2.6	Cosminexus JMS Provider functionality	14
1.2.7	JavaMail functionality	15
1.2.8	CDI functionality used with Application Server	15
1.2.9	Bean Validation functionality used with Application Server	15
1.2.10	Managing application attributes	16
1.2.11	Annotation functionality	16
1.2.12	Functionality for formatting and deploying J2EE applications	17
1.2.13	Container extension library functionality	17
1.3	Description of the functionality mentioned in this manual	19
1.3.1	Meaning of the classification	19
1.3.2	Examples of tables describing the classification	19
1.4	Main functionality changes in Application Server 09-50	21
2	Naming Management	25
2.1	Organization of this chapter	26
2.2	Overview of naming management	27
2.2.1	Naming management functionality	27
2.2.2	Naming services	28
2.3	Binding and looking up objects in the JNDI name space	29
2.3.1	Types of names used for lookup	29
2.3.2	Mapping and looking up the JNDI name space	32
2.3.3	How to check the JNDI name space	35
2.3.4	Definitions in cosminexus.xml	36
2.3.5	Execution environment settings	36
2.4	Looking up with the Portable Global JNDI names	38
2.4.1	Types of JNDI name spaces	38
2.4.2	Automatically bound objects	39

2.4.3	Naming rules for the Portable Global JNDI names	40
2.4.4	Controlling the registration of Portable Global JNDI names	45
2.4.5	Looking up with the Portable Global JNDI names when the CTM is used	45
2.4.6	Resource reference names	46
2.4.7	Specifying the Portable Global JNDI names in annotations	49
2.4.8	Definitions in the DD	50
2.4.9	Execution environment settings	51
2.5	Looking up with names beginning with HITACHI_EJB	52
2.6	Assigning an optional name to Enterprise Beans or J2EE resources (User-specified name space functionality)	55
2.6.1	Objects that can be assigned an optional name	55
2.6.2	Rules for assigning the optional names	56
2.6.3	Timing for registering or deleting the optional name	58
2.6.4	Searching from the client	59
2.6.5	Setting the optional names for the Enterprise Beans	60
2.6.6	Setting the optional names for the J2EE resources	62
2.6.7	Execution environment settings	62
2.6.8	Precautions for using the user-specified name space functionality	64
2.7	Searching the CORBA Naming Service by using the round-robin policy	66
2.7.1	Scope of round-robin search	66
2.7.2	Operations when performing a round-robin search	67
2.7.3	Settings required for performing a round-robin search	68
2.7.4	Settings recommended for using the round-robin search functionality	72
2.7.5	Notes on performing a round-robin search	72
2.8	Caching with the naming management functionality	74
2.8.1	Procedure of caching	74
2.8.2	Clearing the Cache Used in Naming	75
2.8.3	Settings for using the caching functionality	76
2.8.4	Notes on caching in naming	77
2.9	Detecting errors in a naming service	78
2.9.1	What is the naming service error detection functionality	78
2.9.2	Concurrent use of the round-robin search functionality	79
2.9.3	Behavior of the naming service error detection functionality	80
2.9.4	Execution environment settings (When the Error Detection functionality is used)	81
2.9.5	Notes on the naming service error detection functionality	82
2.10	Switching the CORBA Naming Services	83
2.11	Re-using the EJB home object references (Functionality for re-connecting to the EJB home objects)	84
2.11.1	Execution environment settings (J2EE server settings)	84
2.11.2	Notes on re-using the EJB home object references	84

3

Resource Connections and Transaction Management	87
3.1 Organization of this chapter	88
3.2 Overview of resource connections and transaction management	89
3.3 Resource connections	90
3.3.1 How to connect to resources	90
3.3.2 Types of resource adapters	91
3.3.3 How to use the resource adapters	95
3.3.4 Resource adapter functionality	96
3.3.5 Functionality other than the resource adapter functionality	98
3.3.6 Implementation for connecting to the resources	99
3.3.7 How to set up resource adapters	99
3.3.8 Procedure for resource adapter settings (To deploy and use the resource adapter as a J2EE resource adapter)	100
3.3.9 Procedure for resource adapter settings (To include and use the resource adapter in the J2EE application)	103
3.3.10 Procedure for resource adapter settings (To use the resource adapter with Inbound)	107
3.3.11 Procedure for resource adapter settings (To use connection pool clustering)	109
3.3.12 Connection settings for using components other than the resource adapters	111
3.3.13 Notes on the resource adapters	111
3.4 Managing transactions	113
3.4.1 Transaction management methods for the resource connections	113
3.4.2 Local transaction and global transaction	114
3.4.3 Transaction types available for each resource	115
3.4.4 Functionality provided with the transaction services	117
3.4.5 Transaction operations during system exceptions	118
3.4.6 Obtaining the transaction manager	119
3.4.7 Overview of processing and the points to remember when using the container-managed transactions (CMT)	120
3.4.8 Overview of processing and the points to remember when using the UserTransaction interface	121
3.4.9 Overview of processing and the points to remember when using the resource adapter-specific transaction management interface	122
3.4.10 Overview of processing and the points to remember when transactions are not used	123
3.4.11 Notes on the JTA-based transaction implementation	124
3.4.12 Settings in the execution environment	124
3.5 Resource sign-on method	127
3.6 Connecting to a database	128
3.6.1 Overview of DB Connector-based connections	128
3.6.2 Available J2EE components and functionality	129
3.6.3 Connectable databases	130
3.6.4 Types of DB Connectors (RAR file)	132
3.6.5 Preconditions and notes on connecting to HiRDB	132
3.6.6 Preconditions and notes on connecting to Oracle	134

3.6.7	Preconditions and notes on connecting to SQL Server	136
3.6.8	Preconditions and notes on connecting to XDM/RD E2	140
3.6.9	Settings in the execution environment (resource adapter settings)	140
3.7	Connecting to a database queue	141
3.7.1	Overview of connections using DB Connector for Cosminexus RM and Cosminexus RM	141
3.7.2	Features of connections using DB Connector for Cosminexus RM and Cosminexus RM	142
3.7.3	Available functionality	145
3.7.4	Connectable databases	146
3.7.5	Types of DB Connector for Cosminexus RM (RAR file)	147
3.7.6	Preconditions for connecting to a HiRDB queue	147
3.7.7	Preconditions for connecting to an Oracle queue	148
3.7.8	Settings for connecting to a database queue	149
3.8	Outbound connection with OpenTP1 (SPP or TP1/Message Queue)	150
3.8.1	Connections using uCosminexus TP1 Connector	150
3.8.2	Connections using TP1/Message Queue - Access	151
3.8.3	Connection settings for OpenTP1 and Outbound	151
3.9	Inbound connection with OpenTP1	152
3.10	Connection with Cosminexus JMS Provider	153
3.11	Connecting to the SMTP server	154
3.12	Using the JavaBeans resources	155
3.12.1	JavaBeans resource functionality	155
3.12.2	Procedure for starting the JavaBeans resources	155
3.12.3	Implementing the JavaBeans resources	156
3.12.4	Setting up the JavaBeans resources	158
3.12.5	Replacing the JavaBeans resources	161
3.13	Connection with other resources	163
3.13.1	Resource adapters used for connection with other resources	163
3.13.2	Functionality available for other resource connections	163
3.14	Functionality for performance tuning	165
3.14.1	Connection pooling	165
3.14.2	Functionality available with connection pooling	168
3.14.3	Connection sharing or association	169
3.14.4	Statement pooling	173
3.14.5	Light transaction	176
3.14.6	In-process transaction service	176
3.14.7	Caching the DataSource objects	176
3.14.8	Optimizing the container-managed sign-on for DB Connector	177
3.14.9	Definitions in cosminexus.xml	178
3.14.10	Settings in the execution environment	178
3.15	Functionality for fault tolerance	181
3.15.1	Detecting the connection errors	181

3.15.2	Waiting for a connection when connections deplete	184
3.15.3	Retrying to obtain a connection	185
3.15.4	Displaying the connection pool information	186
3.15.5	Clearing the connection pool	187
3.15.6	Automatically closing the connections	187
3.15.7	Connection sweeper	189
3.15.8	Transaction timeout and statement cancellation	189
3.15.9	Transaction recovery	190
3.15.10	Output of the SQL statement for troubleshooting	193
3.15.11	Automatically closing the objects	195
3.15.12	Definitions in cosminexus.xml	195
3.15.13	Settings in the execution environment	195
3.16	Other resource adapter functionality (For the resource adapters conforming to the Connector 1.5 specifications)	198
3.16.1	Managing the resource adapter lifecycle	198
3.16.2	Managing the resource adapter work	202
3.16.3	Message inflow	211
3.16.4	Transaction inflow	215
3.16.5	Looking up Administered objects	216
3.16.6	Specifying multiple connection definitions	217
3.16.7	Application Server-specific Connector 1.5 API specifications	220
3.16.8	Settings for using the resource adapters conforming to the Connector 1.5 specifications	222
3.16.9	Examples of property file specification	227
3.16.10	Notes on using the resource adapters conforming to the Connector 1.5 specifications	233
3.17	Functionality for connection pool clustering	234
3.17.1	Connecting to Oracle using Oracle RAC	234
3.17.2	Overview of connection pool clustering	236
3.17.3	Resource adapters used	238
3.17.4	Connection pool clustering operations	241
3.17.5	Procedure for stopping or starting a connection pool manually	246
3.17.6	Settings required for clustering a connection pool	248
3.18	Connection test for resources	249
3.19	Functionality for operations in the firewall environment	252
3.19.1	Communication port for transaction recovery	252
3.19.2	Communication port used by Smart Agent	252
3.19.3	Settings for operations in the firewall environment	252
3.20	Notes on starting a transaction with the EJB client applications	254
3.20.1	Notes on application development	254
3.20.2	Notes on system setup	254
3.20.3	Notes on system operations	256

4	Invoking Application Server from OpenTP1 (TP1 Inbound Integrated Function) (INTENTIONALLY DELETED)	257
	4.1 INTENTIONALLY DELETED	258
5	How to Use JPA with Application Server	259
	5.1 Organization of this chapter	260
	5.2 Features of JPA	261
	5.2.1 Advantages of applications using JPA	261
	5.2.2 Entity class	262
	5.2.3 JPA provider	262
	5.3 JPA functionality that can be used with Application Server	264
	5.3.1 Available JPA providers	264
	5.3.2 Available components	265
	5.3.3 Supported application formats	266
	5.3.4 Supported class loader configuration	266
	5.3.5 available resource adapters	267
	5.4 EntityManager	268
	5.4.1 Methods provided with EntityManager	268
	5.4.2 Types of EntityManager	268
	5.4.3 Transaction control and EntityManager	269
	5.4.4 Persistence unit	269
	5.5 Persistence context	270
	5.5.1 EntityManager and persistence context	270
	5.5.2 Persistence context when the container-managed EntityManager is used	271
	5.5.3 Persistence context when the application-managed EntityManager is used	272
	5.6 How to obtain the container-managed EntityManager	274
	5.6.1 Method of injecting EntityManager in the application	274
	5.6.2 Method of looking up EntityManager from the application	276
	5.6.3 Overriding the @PersistenceContext definition using the DD	278
	5.7 How to obtain the application-managed EntityManager	279
	5.7.1 Method of injecting EntityManagerFactory in the application	279
	5.7.2 Method of looking up EntityManagerFactory from the application	280
	5.7.3 Overriding the @PersistenceUnit definition using the DD	282
	5.8 Definitions in persistence.xml	283
	5.8.1 Attributes specified in the <persistence-unit> tag	283
	5.8.2 Tags specified under the <persistence-unit> tag	283
	5.9 Allocating persistence.xml	288
	5.10 JPA interfaces	289
	5.10.1 javax.persistence.EntityManager interface	289
	5.10.2 javax.persistence.EntityManagerFactory interface	293

5.11 Notes on setting up applications	296
5.11.1 Notes on allocating the entity classes	296
5.11.2 Reference scope of the persistence unit name	296
5.11.3 Items checked when the application is deployed	297
5.11.4 Notes on using the JPA with Application Server	299
5.11.5 Notes when the Cosminexus JPA functionality is not used	299

6

Cosminexus JPA Provider	301
6.1 Organization of this chapter	302
6.2 Cosminexus JPA Provider	303
6.2.1 Processing in Cosminexus JPA Provider	303
6.2.2 Functionality provided by Cosminexus JPA Provider	304
6.2.3 Preconditions for using Cosminexus JPA Provider	305
6.2.4 Estimating the number of DB Connector connections	307
6.3 Updating a database using entities	308
6.4 Entity operations by EntityManager	309
6.4.1 Transition of entity states	309
6.4.2 persist operation for the entities	311
6.4.3 remove operation for the entities	312
6.4.4 Obtaining the entities from the database	312
6.4.5 Synchronization with the database	313
6.4.6 Separate and merge operations of an entity from the persistence context	315
6.4.7 managed entity	316
6.5 Defining the mapping information between the database and Java objects	318
6.6 Entity relationships	319
6.6.1 Relationship types	319
6.6.2 Annotations for relationships	319
6.6.3 Direction of relationships	320
6.6.4 Default mapping (bi-directional relationship)	321
6.6.5 Default mapping (unidirectional relationship)	324
6.7 Cache functionality of the entity objects	329
6.7.1 Processing of the cache functionality	329
6.7.2 Cache reference forms and cache types	331
6.7.3 Scope of the cache functionality	334
6.7.4 Notes on using the cache functionality	334
6.8 Auto-numbering of the primary key values	337
6.9 Database operations based on the query language	338
6.10 Optimistic lock	339
6.10.1 Optimistic lock processing	339
6.10.2 Exception processing when optimistic lock fails	340
6.10.3 Notes on using the optimistic lock	340

6.11 Pessimistic lock in JPQL	342
6.12 Creating an entity class	343
6.12.1 Defining the mapping between an entity class and database	343
6.12.2 Requirements for creating entity classes	343
6.12.3 Specifying the access methods for the entity class fields	344
6.12.4 Creating the accessor method	345
6.12.5 Types of persistence fields and persistence properties of the entities	345
6.12.6 Specifying the primary key in the entities	346
6.12.7 Default mapping rules for the persistence fields and persistence properties	349
6.13 Procedure for inheriting an entity class	351
6.13.1 Inheritance class types	351
6.13.2 Inheritance mapping strategy	352
6.14 Procedure for using EntityManager and EntityManagerFactory	354
6.14.1 Entity lifecycle management with EntityManager	354
6.14.2 How to set up EntityManager and EntityManagerFactory	354
6.14.3 Notes on the API functions of EntityManager	354
6.15 Procedure for specifying the callback method	356
6.15.1 Location for specifying the callback method	356
6.15.2 Implementing the callback methods	357
6.15.3 Order of invoking the callback methods	358
6.16 Procedure for referencing and updating the database with the query language	359
6.16.1 Procedure for referencing and updating the database with JPQL	359
6.16.2 Procedure for referencing and updating the database with the native query	361
6.16.3 Specifying the range of query result items	365
6.16.4 Specifying the flush mode	366
6.16.5 Specifying a query hint	366
6.16.6 Notes on executing a query	366
6.17 JPQL coding method	367
6.17.1 JPQL syntax	367
6.17.2 SELECT statement	367
6.17.3 SELECT clause	368
6.17.4 FROM clause	369
6.17.5 WHERE clause	372
6.17.6 GROUP BY clause and HAVING clause	375
6.17.7 ORDER BY clause	376
6.17.8 Bulk UPDATE statement and Bulk DELETE statement	376
6.17.9 Notes on using JPQL	377
6.17.10 Exceptions thrown when queries are used	379
6.18 Defining persistence.xml	380
6.19 Settings in the execution environment	381

7

Cosminexus JMS Provider	383
7.1 Organization of this chapter	384
7.2 Overview of Cosminexus JMS Provider	385
7.2.1 Cosminexus JMS Provider	385
7.2.2 Location of Cosminexus JMS Provider within Application Server	385
7.2.3 Overview of the Cosminexus JMS Provider functionality	386
7.3 Allocating the CJMSP resource adapters and CJMSP Broker	388
7.3.1 Configuration in which one CJMSP Broker is allocated to one CJMSP resource adapter	388
7.3.2 Configuration in which one CJMSP Broker is allocated to multiple CJMSP resource adapters	388
7.4 Types of messaging models	390
7.4.1 PTP messaging model	390
7.4.2 Pub/Sub messaging model	391
7.5 Configuration of messages	395
7.6 Selecting the received messages using Message Selector	396
7.7 Mechanism for ensuring a highly-reliable message delivery	397
7.7.1 Types of problems occurring during message delivery and how to ensure reliability	397
7.7.2 Using transactions	397
7.7.3 Controlling the message flow rate	398
7.8 CJMSP Broker functionality	400
7.8.1 Connection services	400
7.8.2 Destination management and routing services	401
7.8.3 CJMSP Broker performance monitoring	404
7.8.4 Management information and message persistence services	404
7.9 CJMSP resource adapter functionality	408
7.10 Invoking a Message-driven Bean	409
7.10.1 Features of message processing using the Message-driven Beans	409
7.10.2 Procedure of invoking the Message-driven Beans	409
7.10.3 Settings required in the Message-driven Beans	410
7.10.4 Setting up the transaction context	411
7.11 Restrictions on implementing applications	412
7.12 Definitions in the DD	413
7.13 Flow of execution environment setup	415
7.14 CJMSP Broker settings	417
7.14.1 Setting up the common properties and the management command properties of CJMSP Broker	418
7.14.2 Creating CJMSP Broker	418
7.14.3 Setting up the individual properties of CJMSP Brokers	418
7.14.4 Starting CJMSP Broker	419
7.14.5 Creating destinations	419
7.15 CJMSP resource adapter settings	420
7.16 J2EE application settings	422
7.17 Starting and stopping the system when Cosminexus JMS Provider is used	423

7.17.1 System starting procedure	423
7.17.2 System stopping procedure	424
7.17.3 Checking the CJMSP Broker status	425
7.18 Checking the troubleshooting information	427
7.19 Notes on using Cosminexus JMS Provider	429

8

Using JavaMail	431
8.1 Organization of this chapter	432
8.2 Session properties	433
8.2.1 Session properties for connecting to the SMTP server	433
8.2.2 Session properties for connecting to the POP3 server	436
8.3 Notes on using JavaMail	438

9

Using CDI with Application Server	439
9.1 Organization of this chapter	440
9.2 Overview of the CDI	441
9.3 Application development using the CDI	442
9.3.1 CDI-target applications	442
9.3.2 Injective relationship of the CDI-target J2EE modules	442
9.3.3 Notes on development	444
9.4 Setting up the class path (Development environment settings)	445
9.4.1 File storage destinations	445
9.4.2 Class path settings	445
9.5 Settings for using the CDI (Changing the security settings)	446
9.6 Using the debug log (check development log)	447
9.7 Setting up the execution environment	448
9.8 Notes on using the CDI	449

10

Using Bean Validation with Application Server	451
10.1 Organization of this chapter	452
10.2 Overview of Bean Validation	453
10.3 Applicable scope of Bean Validation	454
10.4 Bean Validation functionality and Bean Validation operations	455
10.5 Procedures for using Bean Validation	459
10.5.1 Procedures for using Bean Validation from JSF	459
10.5.2 Procedures for using Bean Validation from CDI	460
10.6 Using the debug log (check development log)	462
10.7 Output of information when the validation.xml contents are invalid	463
10.8 Notes on the implementation of Bean Validation	464

11	Managing Application Attributes	465
11.1	Organization of this chapter	466
11.2	Managing attributes	467
11.3	Applications containing cosminexus.xml	469
11.3.1	cosminexus.xml	469
11.3.2	Advantages of using the applications containing cosminexus.xml	470
11.3.3	Creating the applications containing cosminexus.xml	472
11.3.4	Creating cosminexus.xml	472
11.3.5	Example of creating cosminexus.xml	474
11.3.6	Operations of the applications containing cosminexus.xml	476
11.3.7	Procedure of migrating from the applications not containing cosminexus.xml	481
11.4	Omitting the DD	482
11.4.1	Configuration of applications that can be executed with Application Server	482
11.4.2	Differences in functionality depending on the presence of application.xml	483
11.4.3	Rules for determining the modules when application.xml exists	485
11.4.4	Rules for determining the modules when application.xml does not exist	486
11.4.5	Operations for Web applications in which web.xml is omitted	488
11.4.6	Display name specified when the DD is omitted	489
11.4.7	Operations for creating application.xml when application.xml is omitted	490
11.4.8	Notes on adding resources in the J2EE applications when application.xml is omitted	491
12	Using Annotations	493
12.1	Organization of this chapter	494
12.2	Specifying annotations	495
12.2.1	Merits of using annotations and the annotations that can be specified	495
12.2.2	Using the library JAR class with a declared annotation	496
12.2.3	Implementing the Enterprise Beans when an annotation is specified	497
12.3	Classes to be loaded and the class path required for loading	498
12.4	Using the DI	501
12.4.1	Types of resources that can be specified in the @Resource annotation	501
12.4.2	Resolving the resource references using the @Resource annotation	502
12.4.3	Operations during DI failure	505
12.4.4	Notes	506
12.5	Controlling the annotation references	507
12.5.1	Purpose and scope of the functionality for controlling the annotation references	507
12.5.2	Timing for referencing annotations	509
12.5.3	Definitions in the DD (module settings)	510
12.5.4	Changing the settings for the functionality for controlling the annotation references	511
12.6	Updating the contents defined in the annotations	512
12.6.1	Updating the annotations	512

12.6.2	Overwriting the annotations using the DD	514
12.6.3	Referencing and updating the definitions with the server management commands	519
12.7	Notes on using annotations	521

13

Formats and Deployment of J2EE Applications		525
13.1	Organization of this chapter	526
13.2	Executable J2EE application formats	527
13.3	Archive-format J2EE applications	528
13.4	Exploded archive-format J2EE applications	529
13.4.1	Overview of the exploded archive format	529
13.4.2	Configuration of the application directory	531
13.4.3	Settings for using the exploded archive-format J2EE applications (changing the security settings)	537
13.4.4	Notes on using the exploded archive format	538
13.5	Deploying and un-deploying J2EE applications	540
13.5.1	Deploying and un-deploying J2EE applications in the archive format	540
13.5.2	Deploying and un-deploying J2EE applications in the exploded archive-format	540
13.5.3	Deploying the EAR files and ZIP files in the exploded archive format	541
13.5.4	Setting up the J2EE application properties	543
13.6	Replacing J2EE applications	545
13.7	Redeploying J2EE applications	546
13.7.1	Replacing J2EE applications by redeploying	546
13.7.2	Statuses and replacement of J2EE applications	548
13.7.3	Notes on replacing J2EE applications by redeploying	548
13.8	Detecting updates and reloading the J2EE applications	550
13.8.1	How to reload the J2EE applications	550
13.8.2	Scope of reloading	555
13.8.3	Class loader configuration used for reloading	556
13.8.4	Operations when an error occurs	557
13.8.5	Files for update detection	557
13.8.6	Update detection interval for J2EE applications	561
13.8.7	Interval for updating the J2EE application configuration file	562
13.8.8	Reloading the Web applications	564
13.8.9	Reloading the JSPs	566
13.8.10	Relationship with the other functionality	567
13.8.11	Reloading the J2EE applications using commands	570
13.8.12	Settings for detecting updates and reloading the J2EE applications	570
13.8.13	Notes and restrictions related to reloading	574
13.9	WAR applications	581
13.9.1	Support range of the operation commands for WAR applications	581
13.9.2	Executable WAR application formats	582
13.9.3	Replacing WAR applications	583

13.9.4 Names of WAR applications	583
13.9.5 Determining the context root	584
13.9.6 Reading the cosminexus.xml file	584

14 Container Extension Library	585
14.1 Organization of this chapter	586
14.2 Using the container extension library	587
14.3 Container extension library functionality	588
14.3.1 Determining the usage of the container extension library	588
14.3.2 Procedure of creating and using the container extension library	589
14.3.3 Settings for using the container extension library functionality	589
14.4 Server start and stop hook functionality	592
14.4.1 Invocation order of the server start and stop hook processing	592
14.4.2 Implementing the server start and stop hook functionality	593
14.4.3 Specifying the class path for using the server start and stop hook functionality	594
14.5 Invoking the CORBA objects through Smart Agent	595
14.5.1 Notes on the implementation of the CORBA object invocation processing	595
14.5.2 Notes on the packaging of the CORBA object invocation processing	595
14.6 Constraints on using the container extension library and server start and stop hook functionality	596

15 Notes on Implementation of Applications	597
15.1 Notes on using the thread-local variables	598
15.2 Notes on Developer's Kit for Java	599
15.2.1 Notes common to Application Server versions	599
15.2.2 Notes related to the differences in specifications with JDK 1.4.2 provided in Application Server Version 6	606
15.2.3 Notes related to the differences in specifications with JDK 5.0 provided in Application Server Version 7 and Version 8	609
15.3 Notes on using loggers beginning with sun.rmi	611

Appendix	613
A. Character Codes	614
A.1 Character codes handled in an application	614
A.2 Character code conversion between the browser and application	616
B. Configuration of the Class Loader	618
B.1 Default class loader configuration	618
B.2 Class loader configuration for local call optimization	620
B.3 Class path set in the class loader	621
C. Contract Between the JPA Provider and EJB Container	623
C.1 Runtime-related contract	623

C.2 Deployment-related contract	627
D. BNF for JPQL	631
E. Use Cases for Cosminexus JMS Provider	634
E.1 Preconditions common to all the use cases	635
E.2 Prerequisite process model	635
E.3 Environment setup for using Cosminexus JMS Provider	636
E.4 Adding applications that use Cosminexus JMS Provider	642
E.5 Deleting applications that use Cosminexus JMS Provider	647
E.6 Starting the Cosminexus JMS Provider services (for the initial startup)	649
E.7 Starting the Cosminexus JMS Provider services (for restarting a running system)	653
E.8 Checking the state of the CJMSP resource adapters and CJMSP Broker	655
E.9 Checking the message delivery status and the action to be taken for accumulated messages (Procedure for pausing CJMSP Broker)	656
E.10 Checking the message delivery status and the action to be taken for accumulated messages (Procedure for stopping an application)	659
E.11 Terminating the Cosminexus JMS Provider services	660
E.12 Compressing the destinations	661
E.13 Changing the destination size	663
E.14 Deleting the persistence subscribers	664
E.15 Monitoring the CJMSP Broker status	666
E.16 Checking the CJMSP Broker details	667
E.17 Checking the destination status	668
E.18 Checking the persistence subscriber status	669
E.19 Analysis of errors in the CJMSP resource adapters	669
E.20 Analysis when CJMSP Broker stops due to an error	670
E.21 Analysis when there is no response from a Cosminexus JMS Provider service	671
E.22 Recovery when an error occurs in a CJMSP resource adapter	673
E.23 Recovery when CJMSP Broker stops due to an error	674
E.24 Recovery when there is no response from a Cosminexus JMS Provider service	676
E.25 Deleting the Cosminexus JMS Provider service instances	678
F. Main Functionality Changes in Each Version	680
F.1 Main functionality changes in 09-00	680
F.2 Main functionality changes in 08-70	683
F.3 Main functionality changes in 08-53	685
F.4 Main functionality changes in 08-50	686
F.5 Main functionality changes in 08-00	689
G. Glossary	693

Index

695

1

Application Server Functionality

This chapter describes the classification and purpose of the Application Server functionality and the correspondence between the functionality and manuals. This chapter also describes the functionality that is changed in this version.

1.1 Classification of functionality

Application Server is a product used for setting up an execution environment for the applications focusing on a J2EE server compliant with Java EE 6 and for developing the applications to be operated on the execution environment. You can use various functionality such as the functionality compliant with the Java EE standard specifications and the functionality independently extended on Application Server. By selecting and using the functionality according to the purpose and intended use, you can set up and operate a highly reliable system with an excellent processing performance.

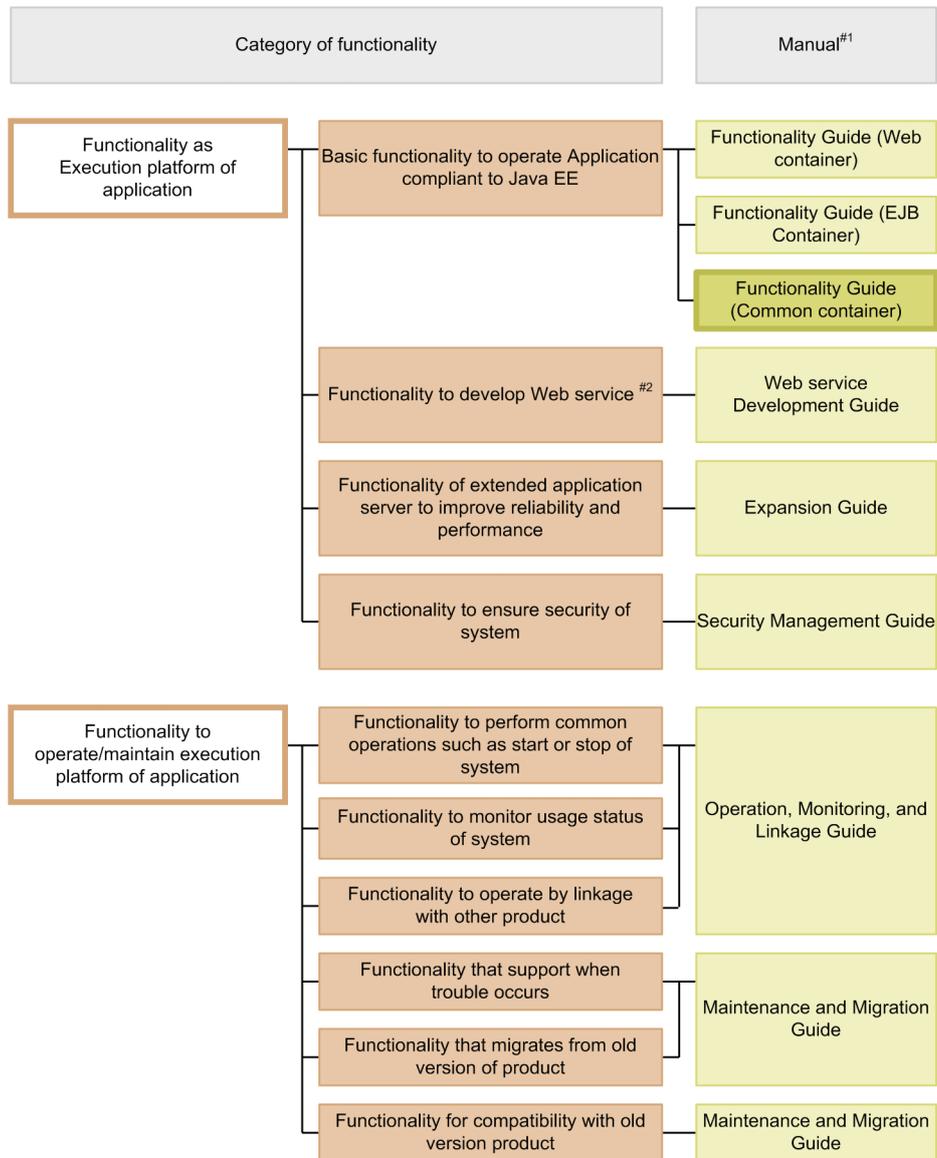
The Application Server functionality can be broadly classified as follows:

- Functionality serving as an execution platform of applications
- Functionality used for operating and maintaining the execution platform of the applications

The above functionality can be further classified according to the positioning and the intended use of the functionality. The Application Server manuals have been provided as per the classification of the functionality.

The following figure shows the classification of the Application Server functionality and the set of manuals corresponding to the functionality.

Figure 1-1: Classification of the Application Server functionality and the set of manuals corresponding to each functionality



Legend:

 : This manual

#1

Application Server has been omitted from the manual names.

#2

You can execute SOAP Web Services and RESTful Web Services with Application Server. Depending on the purpose, see the following manuals in addition to the *uCosminexus Application Server Web Service Development Guide*.

To develop and execute SOAP applications:

- *uCosminexus Application Server SOAP Application Development Guide*

To ensure the security for SOAP Web Services and SOAP applications:

- *uCosminexus Application Server XML Security - Core User Guide*
- *uCosminexus Application Server Web Service Security Users Guide*

To learn about XML processing in detail:

- *uCosminexus Application Server XML Processor User Guide*

The following subsections describe the classification of the functionality and the correspondence of the manuals to the functionality.

1.1.1 Functionality serving as an execution platform for applications

The functionality forms a base for executing online businesses and batch businesses implemented as applications. You choose the functionality you want to use according to the intended use of a system and requirements.

You need to determine whether to use the functionality serving as the execution platform for applications, even before you set up a system and develop applications.

The following points provide the classification-wise description of the functionality serving as the execution platform for applications:

(1) Basic functionality to operate applications (basic development functionality)

This functionality includes the basic functionality for operating the applications (J2EE applications). This functionality mainly includes the J2EE server functionality.

Application Server provides a Java EE 6-compliant J2EE server. The J2EE server provides the functionality unique to Application Server apart from the functionality compliant with the standard specifications.

The basic development functionality can be further classified into three types based on the form of the J2EE application using the functionality. The manuals of the Application Server functionality have been divided according to this classification.

The following subsections give an overview of the classification:

- **Functionality for executing Web applications (Web container)**
This includes the Web container functionality that serves as an execution platform for Web applications and the functionality that is executed by integrating Web containers and Web servers.
- **Functionality for executing Enterprise Beans (EJB container)**
This includes the EJB container functionality that serves as an execution platform for Enterprise Beans. This also includes the EJB client functionality for invoking Enterprise Beans.
- **Functionality used in both Web applications and Enterprise Beans (Common container functionality)**
This includes the functionality that can be used in the Web applications operating on Web containers and the Enterprise Beans operating on EJB containers.

(2) Functionality for developing Web Services

This includes the functionality for the execution and development environment of Web Service.

Application Server provides the following engines:

- JAX-WS engine that binds the SOAP messages in accordance with the JAX-WS specifications
- JAX-RS engine that binds the RESTful HTTP messages in accordance with the JAX-RS specifications

(3) Application server-specific functionality expanded for improving the reliability and performance (Expansion functionality)

This includes the functionality expanded uniquely on Application Server. This also includes the functionality implemented by using non-J2EE server processes, such as a batch server, CTM, and database.

With Application Server, various functionality are expanded to improve the reliability of the system and to implement stable operations. Furthermore, the functionality is also expanded to operate applications other than the J2EE applications (batch applications) in the Java environment.

(4) Functionality to ensure system security (Security management functionality)

This includes the functionality used for ensuring the security of Application Server-based systems. This includes the authentication functionality for preventing unauthorized access and the encryption functionality for preventing information leakage through a communication path.

1.1.2 Functionality for operating and maintaining the execution platform of applications

This functionality is used to efficiently operate and maintain the execution platform for the applications. You use this functionality as and when required after starting system operations. However, depending on the functionality, you need to implement some settings and applications in advance.

The following points classification-wise describe the functionality used for operating and maintaining the execution platform of applications:

(1) Functionality used for daily operations such as system start and stop (Operation functionality)

This classification includes the functionality used in daily operations, such as starting or stopping systems, starting or stopping applications, and replacing the applications.

(2) Functionality for monitoring system usage (Watch functionality)

This includes the functionality used for monitoring system operations and resource depletion. This also includes the functionality to output the information used in monitoring the system operation history.

(3) Functionality for operating the systems by integrating with other products (Linkage functionality)

This includes the functionality to be integrated and implemented with other products, such as JP1 and cluster software.

(4) Functionality for troubleshooting (Maintenance functionality)

This includes the functionality used for troubleshooting. This also includes the functionality used to output the information that is referenced for troubleshooting.

(5) Functionality for migrating from products of older versions (Migration functionality)

This includes the functionality used for migrating from old Application Server to a new Application Server.

(6) Functionality for compatibility with products of older versions (Compatibility functionality)

This includes the functionality used for compatibility with older versions of Application Server. We recommend you to migrate the compatibility functionality to the corresponding recommended functionality.

1.1.3 Correspondence between the functionality and manuals

The functionality guides for Application Server have been divided according to the classification of the functionality.

The following table describes the classification of the functionality and the manuals corresponding to the respective functionality.

Table 1-1: Classification of functionality and corresponding functionality guides

Classification	Functionality	Manual #1
Basic development functionality	Web container	<i>Web Container Functionality Guide</i>

1. Application Server Functionality

Classification	Functionality	Manual #1
Basic development functionality	Using JSF and JSTL	<i>Web Container Functionality Guide</i>
	Integrating with Web server	
	In-process HTTP server	
	Implementing servlets and JSPs	
	EJB container	<i>EJB Container Functionality Guide</i>
	EJB client	
	Notes on implementing Enterprise Beans	
	Naming management	<i>Common Container Functionality Guide</i> #2
	Managing resource connections and transactions	
	Invoking Application Server from OpenTP1 (TP1 inbound integrated function)	
	Using JPA with Application Server	
	Cosminexus JPA Provider	
	Cosminexus JMS Provider	
	Using JavaMail	
	Using CDI with Application Server	
	Using Bean Validation with Application Server	
	Managing application attributes	
	Using annotations	
	Formatting and deploying J2EE applications	
Container extension library		
Expansion functionality	Executing applications by using batch servers	<i>Expansion Guide</i>
	Scheduling and load balancing requests using CTM	
	Scheduling batch applications	
	Inheriting the session information between J2EE servers (Session failover functionality)	
	Database session failover functionality	
	EADs session failover functionality	
	Controlling the full garbage collection by using the Explicit Memory Management functionality	
	Output of application user logs	
	Asynchronous parallel processing of threads	
Security management functionality	Authentication using integrated user management	<i>Security Management Guide</i>
	Authentication using the application settings	
	Using the TLSv1.2 for the SSL/TLS communication	
	Controlling access with the management functionality for load balancers using API-based direct connections	

Classification	Functionality	Manual #1
Operation functionality	Starting and stopping a system	<i>Operation, Monitoring, and Linkage Guide</i>
	Operating J2EE applications	
Monitoring functionality	Monitoring the operation information (Statistics collection functionality)	
	Monitoring resource depletion	
	Audit log output functionality	
	Database audit trail integration functionality	
	Output of the operation information by using management commands	
	Automatic execution of processing by using Management event reporting and Management action	
	Collecting the statistical information of CTM	
	Output of console logs	
Linkage functionality	Operating a JP1-integrated system	
	Centralized monitoring of systems (Integrating with JP1/IM)	
	Automatic system operations by using jobs (Integrating with JP1/AJS)	
	Collecting and consolidating audit logs (Integrating with JP1/Audit Management - Manager)	
	Integration with cluster software	
	1-to-1 node switching systems (Integrating with cluster software)	
	Mutual node switching systems (Integrating with cluster software)	
	N-to-1 recovery systems (Integrating with cluster software)	
	Node switching system for the host unit management model (Integrating with cluster software)	
Maintenance functionality	Troubleshooting-related functionality	<i>Maintenance and Migration Guide</i>
	Performance analysis by using the trace-based performance analysis	
	JavaVM functionality of the product (referred to as JavaVM hereafter)	
Migration functionality	Migrating from old Application Server	
	Migrating to the recommended functionality	
Compatibility functionality	Basic mode	<i>Compatibility Guide</i>
	Servlet engine mode	
	Functionality for compatibility with the basic development functionality	
	Functionality for compatibility with the extension functionality	

#1: *uCosminexus Application Server* has been omitted from the manual names.

#2: This manual.

1.2 Correspondence between the purpose of a system and the functionality

With Application Server, you need to choose the applicable functionality according to the purpose of the system to be set up and operated.

This subsection describes the types of systems for the functionality that can be used in common by Web and EJB containers. The functionality-wise correspondence with the following items is described here:

- **Reliability**
 This functionality is best used for the systems that require high reliability.
 This functionality includes the functionality for enhancing the system availability (stable operations) and fault tolerance, and the functionality for enhancing the security, such as user authentication.
- **Performance**
 This functionality is best used for the systems that emphasize performance.
 This functionality includes the functionality used for system performance tuning.
- **Operation and maintenance**
 This functionality is best used when you want efficient operations and maintenance.
- **Extendibility**
 This functionality is best used when you need to expand or reduce the system scale and for flexible response for the changes in the configuration.
- **Others**
 The functionality is used for supporting other individual purposes.

Furthermore, the functionality that can be used in common by Web containers and EJB containers includes the Java EE standard functionality and the functionality independently extended by Application Server. When you select the functionality, check the compliance with the Java EE standards, as and when required.

1.2.1 Naming management functionality

The following table describes the naming management functionality. Select the functionality according to the purpose of the system. For details about the functionality, see the *reference location*.

Table 1-2: Correspondence between the naming management functionality and the purpose of a system

Functionality	Purpose of a system					Compliance with Java EE standards		Reference location
	Reliability	Performance	Operation and maintenance	Expansion	Others	Standard	Expanded	
Binding and looking up objects in the JNDI name space	N	N	N	Y	N	Y	Y	2.3
Looking up with the Portable Global JNDI names	N	N	N	Y	N	Y	Y	2.4
Looking up with names beginning with	N	N	N	Y	N	Y	Y	2.5

Functionality	Purpose of a system					Compliance with Java EE standards		Reference location
	Reliability	Performance	Operation and maintenance	Expansion	Others	Standard	Expanded	
HITACHI_EJB	N	N	N	Y	N	Y	Y	2.5
Assigning an optional name to the Enterprise Beans or J2EE resources (user-specified name space functionality)	N	N	N	Y	N	N	Y	2.6
Searching the CORBA Naming Service by using the round-robin policy	N	N	N	Y	N	N	Y	2.7
Caching with the naming management functionality	N	Y	N	N	N	N	Y	2.8
Detecting errors in a naming service	Y	N	N	N	N	N	Y	2.9
Switching the CORBA Naming Service	N	N	N	Y	N	N	Y	2.10
Re-using the EJB home object references (functionality for re-connecting to the EJB home objects)	N	N	N	Y	N	Y	Y	2.11

Legend:

Y: Supported

N: Not supported

Note: The functionality for which Y is specified in both *Standard* and *Expanded* columns of the *Compliance with Java EE standards* column indicates that the Application Server-specific functionality has been expanded to the Java EE standard functionality. The functionality for which Y is specified only in the *Expanded* column indicates the Application Server-specific functionality.

1.2.2 Functionality for managing resource connections and transactions

The following table describes the functionality for managing resource connections and transactions. Select the functionality according to the purpose of the system. For details about the functionality, see the *reference location*.

1. Application Server Functionality

Table 1-3: Correspondence between the functionality for managing the resource connections and transactions and the purpose of a system

Functionality name	Purpose of the system					Compliance with Java EE standards		Reference location
	Reliability	Performance	Operation and maintenance	Expansion	Others	Standard	Expanded	
Connection pooling	N	Y	N	N	N	Y	Y	3.14.1
Functionality available with connection pooling	N	Y	N	N	N	N	Y	3.14.2
Connection sharing and association	N	Y	N	N	N	Y	N	3.14.3
Statement pooling	N	Y	N	N	N	N	Y	3.14.4
Light transaction	N	Y	N	N	N	N	Y	3.14.5
In-process transaction service	N	Y	N	N	N	Y	N	3.14.6
DataSource object caching	N	Y	N	N	N	N	Y	3.14.7
Optimizing container management sign-on in DB Connector	N	Y	N	N	N	N	Y	3.14.8
Detecting connection errors	Y	N	N	N	N	Y ^{#1}	Y ^{#1}	3.15.1
Waiting to acquire connections in the case of connection depletion	Y	N	N	N	N	N	Y	3.15.2
Retrying to acquire connections	Y	N	N	N	N	N	Y	3.15.3
Displaying the connection pool information	Y	N	N	N	N	N	Y	3.15.4
Clearing the connection pool	Y	N	N	N	N	N	Y	3.15.5
Automatically closing a connection	Y	N	N	N	N	Y	N	3.15.6

Functionality name	Purpose of the system					Compliance with Java EE standards		Reference location
	Reliability	Performance	Operation and maintenance	Expansion	Others	Standard	Expanded	
Connection sweeper	Y	N	N	N	N	N	Y	3.15.7
Transaction timeout and statement cancellation	Y	N	N	N	N	Y	N	3.15.8
Transaction recovery	Y	N	N	N	N	Y	N	3.15.9
Issuing of SQL statements for troubleshooting	N	N	Y	N	N	N	Y	3.15.10
Automatic closing of objects	Y	N	N	N	N	Y	N	3.15.11
Managing the resource adapter life cycle #2	N	N	N	N	N	Y	N	3.16.1
Managing the resource adapter work #2	N	N	N	N	N	Y	N	3.16.2
Message in-flow #2	N	N	N	N	N	Y	N	3.16.3
Transaction in-flow #3	N	N	N	N	N	Y	N	3.16.4
Looking up Administered objects #2	N	N	N	N	N	Y	N	3.16.5
Specifying multiple connection definitions #2	N	N	N	N	N	Y	N	3.16.6
Connection pool clustering functionality (temporary stop, restart, and status display of the connection pool)	Y	N	N	N	N	N	Y	3.17
Resource connection test	N	N	Y	N	N	N	Y	3.18

1. Application Server Functionality

Functionality name	Purpose of the system					Compliance with Java EE standards		Reference location
	Reliability	Performance	Operation and maintenance	Expansion	Others	Standard	Expanded	
Functionality for operations in a firewall environment	N	N	Y	N	N	N	Y	3.19

Legend:

Y: Supported

N: Not supported

Note: The functionality for which Y is specified in both *Standard* and *Expanded* columns of the *Compliance with Java EE standards* column indicates that the Application Server-specific functionality has been expanded to the Java EE standard functionality. The functionality for which Y is specified only in the *Expanded* column indicates the Application Server-specific functionality.

#1: In the case of Connector 1.0, *Expanded*. In the case of Connector 1.5, *Java EE standards*.

#2: This functionality can be used only when you use a resource adapter compliant with the Connector 1.5 specifications.

#3: This functionality can be used only when you use the TP1 inbound adapter from the resource adapters compliant with the Connector 1.5 specifications.

1.2.3 Functionality for invoking Application Server from OpenTP1 (TP1 inbound integrated functionality)

The following table describes the TP1 inbound integrated functionality used for invoking Application Server from OpenTP1. Select the functionality according to the purpose of a system. For details about the functionality, see the *reference location*.

Table 1-4: Correspondence between Application Server invocation from OpenTP1 (TP1 inbound integrated functionality) and the purpose of a system

Functionality name	Purpose of a system					Compliance with Java EE standards		Reference location
	Reliability	Performance	Operation and maintenance	Expansion	Others	Standard	Expanded	
Connection management functionality	N	N	Y	N	Y	N	Y	4.6
RPC Connection functionality	N	N	Y	N	Y	N	Y	4.7
Scheduling function	N	N	Y	N	Y	N	Y	4.8
Transaction integration functionality	N	N	Y	N	Y	N	Y	4.9

Legend:

Y: Supported

N: Not supported

Note: The functionality for which Y is specified only in the *Expanded* column indicates the Application Server-specific functionality.

1.2.4 JPA functionality used with Application Server

The following table describes the JPA functionality used with Application Server. Select the functionality according to the purpose of a system. For details about the functionality, see the *reference location*.

Table 1-5: Correspondence between the JPA functionality used with Application Server and the purpose of a system

Functionality name	Purpose of a system					Compliance with Java EE standards		Reference location
	Reliability	Performance	Operation and maintenance	Expansion	Others	Standard	Expanded	
JPA	Y	N	Y	N	N	Y	Y	Chapter 5

Legend:

Y: Supported

N: Not supported

Note: The functionality for which Y is specified in both *Standard* and *Expanded* columns of the *Compliance with Java EE standards* column indicates that the Application Server-specific functionality has been extended to the Java EE standard functionality.

1.2.5 Cosminexus JPA Provider functionality

The following table describes the Cosminexus JPA Provider functionality. Select the functionality according to the purpose of a system. For details about the functionality, see the *reference location*.

Table 1-6: Correspondence between the Cosminexus JPA Provider functionality and the purpose of a system

Functionality name	Purpose of a system					Compliance with Java EE standards		Reference location
	Reliability	Performance	Operation and maintenance	Expansion	Others	Standard	Expanded	
Updating the database using entities	Y	Y	N	N	N	Y	Y	6.3
Entity operations by EntityManager	N	N	N	N	Y	Y	Y	6.4
Defining the mapping information between a database and Java objects	N	N	N	N	Y	Y	Y	6.5
Entity relationship	N	Y	N	N	N	Y	Y	6.6
Functionality for caching entity objects	N	Y	N	N	N	N	Y	6.7
Auto-numbering of primary key values	N	N	N	N	Y	Y	Y	6.8

1. Application Server Functionality

Functionality name	Purpose of a system					Compliance with Java EE standards		Reference location
	Reliability	Performance	Operation and maintenance	Expansion	Others	Standard	Expanded	
Database operations based on query language	N	N	Y	N	N	Y	Y	6.9
Optimistic lock	Y	N	N	N	N	Y	Y	6.10
Pessimistic lock in the JPQL	Y	N	N	N	N	N	Y	6.11

Legend:

Y: Supported

N: Not supported

Note: The functionality for which Y is specified in both *Standard* and *Expanded* columns of the *Compliance with Java EE standards* column indicates that the Application Server-specific functionality has been expanded to the Java EE standard functionality. The functionality for which Y is specified only in the *Expanded* column indicates the Application Server-specific functionality.

1.2.6 Cosminexus JMS Provider functionality

The following table describes the Cosminexus JMS Provider functionality. Select the functionality according to the purpose of a system. For details about the functionality, see the *reference location*.

Table 1-7: Correspondence between the Cosminexus JMS Provider functionality and the purpose of a system

Functionality name	Purpose of a system					Compliance with Java EE standards		Reference location
	Reliability	Performance	Operation and maintenance	Expansion	Others	Standard	Expanded	
Sending and receiving the messages compliant with the JMS specifications	N	N	N	N	Y	Y	N	7.4, 7.5, 7.6
System for ensuring highly reliable message delivery	Y	N	N	N	N	Y	N	7.7
CJMSP Broker functionality	N	N	Y	N	N	Y	N	7.8
CJMSP resource adapter functionality	N	N	N	N	Y	Y	N	7.9

Functionality name	Purpose of a system					Compliance with Java EE standards		Reference location
	Reliability	Performance	Operation and maintenance	Expansion	Others	Standard	Expanded	
Invoking the Message-driven Beans	N	N	N	N	Y	Y	N	7.10

Legend:

Y: Supported

N: Not supported

1.2.7 JavaMail functionality

The following table describes the JavaMail functionality. Select the functionality according to the purpose of a system. For details about the functionality, see the *reference location*.

Table 1-8: Correspondence between the JavaMail functionality and the purpose of a system

Functionality name	Purpose of a system					Compliance with Java EE standards		Reference location
	Reliability	Performance	Operation and maintenance	Expansion	Others	Standard	Expanded	
JavaMail	N	N	N	N	Y	Y	N	Chapter 8

Legend:

Y: Supported

N: Not supported

1.2.8 CDI functionality used with Application Server

The following table describes the CDI functionality used with Application Server. Select the functionality according to the purpose of a system. For details about the functionality, see the *reference location*.

Table 1-9: Correspondence between the CDI functionality and the purpose of a system

Functionality name	Purpose of a system					Compliance with Java EE standards		Reference location
	Reliability	Performance	Operation and maintenance	Expansion	Others	Standard	Expanded	
CDI	N	N	N	N	Y	Y	N	Chapter 9

Legend:

Y: Supported

N: Not supported

1.2.9 Bean Validation functionality used with Application Server

The following table describes the Bean Validation functionality used with Application Server. Select the functionality according to the purpose of a system. For details on the functionality, see the *Reference location* column.

Table 1–10: Correspondence between the Bean Validation functionality and the purpose of a system

Functionality name	Purpose of a system					Compliance with Java EE standards		Reference location
	Reliability	Performance	Operation and maintenance	Expansion	Others	Standard	Expanded	
Bean Validation	--	--	--	--	Y	Y	--	<i>Chapter 10</i>

Legend:

Y: Supported

--: Not supported

1.2.10 Managing application attributes

The following table describes the functionality for managing application attributes. Select the functionality according to the purpose of a system. For details about the functionality, see the *reference location*.

Table 1–11: Correspondence between the functionality for managing the application attributes and the purpose of a system

Functionality name	Purpose of a system					Compliance with Java EE standards		Reference location
	Reliability	Performance	Operation and maintenance	Expansion	Others	Standard	Expanded	
Applications containing <code>cosminexus.xml</code>	N	N	N	Y	N	N	Y	<i>11.3</i>
Omitting a DD	N	N	N	Y	N	Y	Y	<i>11.4</i>

Legend:

Y: Supported

N: Not supported

Note: The functionality for which Y is specified in both *Standard* and *Expanded* columns of the *Compliance with Java EE standards* column indicates that the Application Server-specific functionality has been expanded to the Java EE standard functionality. The functionality for which Y is specified only in the *Expanded* column indicates the Application Server-specific functionality.

1.2.11 Annotation functionality

The following table describes the annotation functionality. Select the functionality according to the purpose of a system. For details about the functionality, see the *reference location*.

Table 1–12: Correspondence between the annotation functionality and the purpose of a system

Functionality name	Purpose of the system					Compliance with Java EE standards		Reference location
	Reliability	Performance	Operation and maintenance	Expansion	Others	Standard	Expanded	
Annotation	N	N	N	Y	N	Y	Y	<i>Chapter 12</i>

Legend:

Y: Supported

N: Not supported

Note: The functionality for which Y is specified in both *Standard* and *Expanded* columns of the *Compliance with Java EE standards* column indicates that the Application Server-specific functionality has been expanded to the Java EE standard functionality.

1.2.12 Functionality for formatting and deploying J2EE applications

The following table describes the functionality for formatting and deploying J2EE applications. Select the functionality according to the purpose of the system. For details about the functionality, see the *reference location*.

Table 1-13: Correspondence between the functionality for formatting and deploying J2EE applications and the purpose of a system

Functionality name	Purpose of the system					Compliance with Java EE standards		Reference location
	Reliability	Performance	Operation and maintenance	Expansion	Others	Standard	Expanded	
Deploying and undeploying J2EE applications	N	N	N	N	N	Y	Y	13.5
Replacing J2EE applications	N	N	N	N	N	Y	N	13.6
Redeploying J2EE applications	N	N	Y	N	N	N	Y	13.7
Detecting updates and reloading J2EE applications	N	N	Y	N	N	N	Y	13.8

Legend:

Y: Supported

N: Not supported

Note: The functionality for which Y is specified in both *Standard* and *Expanded* columns of the *Compliance with Java EE standards* column indicates that the Application Server-specific functionality has been expanded to the Java EE standard functionality. The functionality for which Y is specified only in the *Expanded* column indicates the Application Server-specific functionality.

1.2.13 Container extension library functionality

The following table describes the container extension library functionality. Select the functionality according to the purpose of a system. For details about the functionality, see the *reference location*.

1. Application Server Functionality

Table 1-14: Correspondence between the container extension library functionality and the purpose of a system

Functionality name	Purpose of the system					Compliance with Java EE standards		Reference location
	Reliability	Performance	Operation and maintenance	Expansion	Others	Standard	Expanded	
Container extension library functionality	N	N	N	Y	N	N	Y	14.3
Server start/stop hook functionality	N	N	N	Y	N	N	Y	14.4
Invoking the CORBA objects through Smart Agent	N	N	N	Y	N	N	Y	14.5

Legend:

Y: Supported

N: Not supported

Note: The functionality for which Y is specified only in the *Expanded* column indicates the Application Server-specific functionality.

1.3 Description of the functionality mentioned in this manual

This subsection describes the meaning of the classification used in the description of the functionality in this manual and the examples of the tables describing the classification.

1.3.1 Meaning of the classification

The description of the functionality in this manual is classified into the following five points. You can select and read the required location depending on the purpose of referencing the manual.

- **Explanation**
This is the explanation about the functionality. This section explains the purpose, features, and mechanism of the functionality. Read this section if you want an overview of the functionality.
- **Implementation**
This section describes the methods such as the coding method and the DD writing method. Read this section when developing applications.
- **Settings**
This section describes the required property settings for the system setup. Read this section when setting up the system.
- **Operations**
This section describes operation methods. This section describes the operating procedures and the execution examples of the commands to be used. Read this section when operating the system.
- **Notes**
This section describes the general precautions for using the functionality. Make sure that you read the notes.

1.3.2 Examples of tables describing the classification

The following table lists the classification of the functionality description. The title of the table is "organization of this chapter" or "organization of this section".

The following is an example of a table describing the classification of the functionality.

Example of a table describing the classification of the functionality

Table X-1: Organization of this chapter (YY functionality)

Category	Title	Reference location
Explanation	What is the YY functionality	X.1
Implementation	Implementation of applications	X.2
	Definitions in the DD and <code>cosminexus.xml</code> #	X.3
Settings	Settings in the execution environment	X.4
Operations	Operations using the YY functionality	X.5
Notes	Notes on using the YY functionality	X.6

#

For details on `cosminexus.xml`, see 11. *Managing Application Attributes*.

Tip

Property settings for applications that do not contain `cosminexus.xml`

1. Application Server Functionality

With the applications that do not contain `cosminexus.xml`, you set up or change the properties after importing the applications into the execution environment. You can also change the specified properties in the execution environment.

You specify the application settings in the execution environment using the server management commands and the property files. For details on the application settings with the server management commands and the property files, see 3.5.2 *How to set up the J2EE application properties* in the *uCosminexus Application Server Application Setup Guide*.

The tags specified in the property files correspond to a DD or `cosminexus.xml`. For details on the correspondence between the DD or `cosminexus.xml` and the property file tags, see 2.2 *Details of attributes specified in the Cosminexus application property file (cosminexus.xml)* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

Note that the properties specified with each property file can also be specified with the HITACHI Application Integrated Property File.

1.4 Main functionality changes in Application Server 09-50

This section describes the main changes in the functionality of Application Server 09-50 and the purpose of each change.

The contents described in this section are as follows:

- This section gives an overview and describes the changes in the main functionality of Application Server 09-50. For details on the functionality, see the description in the reference location. The *Reference manual* and *Reference section* columns describe the main locations where that functionality has been described.
- *uCosminexus Application Server* is omitted from the manual names mentioned in the *Reference manual* column.

(1) Improving development productivity

The following table describes the items that are changed to improve development productivity.

Table 1-15: Changes made for improving development productivity

Item	Overview of changes	Reference manual	Reference section
Simplification of Eclipse setup	GUI can now be used to set up the Eclipse environment.	<i>Application Development Guide</i>	1.1.5, 2.4
Supporting debug using the user-extended trace-based performance analysis	The user-extended trace-based performance analysis setup file can now be created in the development environment.	<i>Application Development Guide</i>	1.1.3, 6.5

(2) Simplifying installation and setup

The following table describes the items that are changed to simplify installation and setup.

Table 1-16: Changes made for simplifying installation and setup

Item	Overview of changes	Reference manual	Reference section
Expanding the system configuration patterns in the virtual environment	More tier types are now available in the virtual environment (<code>http-tier</code> , <code>j2ee-tier</code> , and <code>ctm-tier</code>). Accordingly, the following system configuration patterns can now be set up: <ul style="list-style-type: none"> • Pattern of allocating the Web server and J2EE server on different hosts • Pattern of allocating the front end (servlets or JSPs) and back end (EJBs) separately • Pattern of using the CTM 	<i>Virtual System Setup and Operation Guide</i>	1.1.2

(3) Supporting standard and existing functionality

The following table describes the items that are changed to support standard and existing functionality.

Table 1-17: Changes made for supporting standard and existing functionality

Item	Overview of changes	Reference manual	Reference section
Supporting the JDBC 4.0 specifications	HiRDB Type4 JDBC Driver and SQL Server JDBC driver complying with the JDBC 4.0 specifications are now supported with DB Connector.	This manual	3.6.3

1. Application Server Functionality

Item	Overview of changes	Reference manual	Reference section
Relaxation of the naming rules for the Portable Global JNDI names	Characters that can be used in the Portable Global JNDI names are added.	This manual	2.4.3
Supporting the Servlet 3.0 specifications	The changes in the HTTP Cookie name, and URL path parameter name in Servlet 3.0 can now also be used in Servlet 2.5 or earlier.	<i>Web Container Functionality Guide</i>	2.7
Extended use of applications that can be integrated with Bean Validation	Bean Validation can now be used for validation in the CDI and user applications as well.	This manual	Chapter 10
Supporting JavaMail	The mail sending and receiving functionality using JavaMail 1.4-compliant APIs is now available.	This manual	Chapter 8
Extended use of the OS that can use the <code>javacore</code> command	The <code>javacore</code> command can now be used to obtain the Windows thread dump.	<i>Command Reference Guide</i>	<i>javacore (Obtaining the thread dump/in Windows)</i>

(4) Maintaining and improving reliability

The following table describes the items that are changed for maintaining and improving reliability.

Table 1-18: Changes made for maintaining and improving reliability

Item	Overview of changes	Reference manual	Reference section
Avoiding the depletion of the code cache area	The size of the code cache area used in the system can now be checked, and the threshold value can be changed to avoid the depletion of area before the area is depleted.	<i>System Design Guide</i>	7.1.2
		<i>Maintenance and Migration Guide</i>	5.7.2, 5.7.3
		<i>Definition Reference Guide</i>	16.1, 16.2, 16.4
Supporting the efficient application of the Explicit Memory Management functionality	<p>The functionality that can control the objects transferred to the Explicit heap is added as the functionality for reducing the automatic release processing time and applying the Explicit Memory Management functionality efficiently.</p> <ul style="list-style-type: none"> Functionality for controlling the transfer of objects to the Explicit memory block Functionality for specifying the classes to be excluded from the Explicit Memory Management functionality Output of the object release rate information in the Explicit heap information 	<i>System Design Guide</i>	7.13.6
		<i>Expansion Guide</i>	8.2.2, 8.6.5, 8.10, 8.13.1, 8.13.3
		<i>Maintenance and Migration Guide</i>	5.5
Expanding the range of the class-wise statistical information that is output	The <code>static</code> field-based reference relationships can now be output to the extended thread dump containing class-wise statistical information.	<i>Maintenance and Migration Guide</i>	9.6

(5) Maintaining and improving operability

The following table describes the items that are changed with the purpose of maintaining and improving operability.

Table 1-19: Changes made for maintaining and improving operability

Item	Overview of changes	Reference manual	Reference section
Supporting the EADs session failover functionality	The EADs session failover functionality that implements the session failover functionality by integrating with the EADs is now supported.	<i>Expansion Guide</i>	<i>Chapter 5, Chapter 7</i>
Operations using the WAR files	The WAR applications configured only with WAR files can now be deployed on J2EE servers.	<i>Web Container Functionality Guide</i>	2.2.1
		This manual	13.9
		<i>Command Reference Guide</i>	<i>cjimportwar</i> (Importing WAR applications)
Starting and stopping the management functionality using synchronous execution	A synchronous execution option is added for starting and stopping the management functionality (Management Server and Administration Agent).	<i>Operation, Monitoring, and Linkage Guide</i>	2.6.1, 2.6.2, 2.6.3, 2.6.4
		<i>Command Reference Guide</i>	<i>adminagentctl</i> (Starting and stopping Administration Agent), <i>mngautorun</i> (Setting up and cancelling the settings for auto-start and auto-restart), <i>mngsvrctl</i> (Starting, stopping, and setting up Management Server)
Releasing the Explicit memory block forcefully with the Explicit Memory Management functionality	The release processing of the Explicit memory block can now be executed at any time with the <code>javagc</code> command.	<i>Expansion Guide</i>	8.6.1, 8.9
		<i>Command Reference Guide</i>	<i>javagc</i> (Forced garbage collection)

(6) Other purposes

The following table describes the items that are changed for other purposes.

Table 1-20: Changes made for other purposes

Item	Overview of changes	Reference manual	Reference section
Obtaining definition information	The <code>snapshot</code> (collecting the snapshot log) command can now be used to collect only definition files.	<i>Maintenance and Migration Guide</i>	2.3
		<i>Command Reference Guide</i>	<i>snapshotlog</i> (Collecting the

1. Application Server Functionality

Item	Overview of changes	Reference manual	Reference section
Obtaining definition information	The <code>snapshot</code> (collecting the snapshot log) command can now be used to collect only definition files.	<i>Command Reference Guide</i>	<i>snapshot log)</i>
Output of the <code>cjenvsetup</code> command log	The execution information for the Component Container Administrator setup (the <code>cjenvsetup</code> command) is now output to the message log.	<i>System Setup and Operation Guide</i>	4.1.4
		<i>Maintenance and Migration Guide</i>	4.20
		<i>Command Reference Guide</i>	<i>cjenvsetup (Setting up Component Container Administrator)</i>
Supporting BIG-IP v11	BIG-IP v11 is now added to the available load balancer types.	<i>System Setup and Operation Guide</i>	4.7.2
		<i>Virtual System Setup and Operation Guide</i>	2.1
Output of the CPU time to the event log of the Explicit Memory Management functionality	The CPU time taken for the Explicit memory block release processing is now output to the event log of the Explicit Memory Management functionality.	<i>Maintenance and Migration Guide</i>	5.11.3
Extending the functionality of the user-extended trace-based performance analysis	<p>The following functionality is added in the user-extended trace-based performance analysis:</p> <ul style="list-style-type: none"> • The trace target specification method can now be specified for packages or classes in addition to the usual trace target specification method for methods. • The range of available event IDs is expanded. • The restrictions on the number of lines that can be specified in the user-extended trace-based performance analysis setup file are relaxed. • The trace collection level can now be specified in the user-extended trace based performance analysis setup file. 	<i>Maintenance and Migration Guide</i>	7.5.2, 7.5.3, 8.28.1
Improving the analysis of information when asynchronous Session Bean invocation is used	The root application information of the PRF trace can now be used to compare the invocation source and the invocation destination requests.	<i>EJB Container Functionality Guide</i>	2.17.3

2

Naming Management

This chapter describes the naming management functionality. The naming management is one of the functionality provided with the J2EE services. You use the naming management functionality to resolve the names for the Enterprise Bean references or the resource references.

2.1 Organization of this chapter

The *naming management* is one of the functionality provided with the J2EE services. A J2EE service is a component functionality of the J2EE container.

The following table describes the naming management functionality and the reference locations.

Table 2-1: Naming management functionality and reference locations

Functionality	Reference location
Overview of naming management	2.2
Binding and looking up objects in the JNDI name space [#]	2.3
Looking up with the Portable Global JNDI names	2.4
Looking up with names beginning with <code>HITACHI_EJB</code>	2.5
Assigning an optional name to the Enterprise Beans or J2EE resources (user-specified name space functionality)	2.6
Searching the CORBA Naming Service by using the round-robin policy	2.7
Caching with the naming management functionality	2.8
Naming service error detection	2.9
Switching the CORBA Naming Services	2.10
Re-using the EJB home object references (functionality for re-connecting to the EJB home objects)	2.11

[#]: There are various types of lookup methods. For details on the lookup names and methods, see 2.3.1 *Types of names used for lookup*.

2.2 Overview of naming management

This section describes the naming management functionality and the naming services used for naming management.

2.2.1 Naming management functionality

The naming management manages the name and storage location of an object (EJB home objects, business interface references, and J2EE resources corresponding to the Enterprise Beans). By using the naming management functionality, the EJB client is able to use the necessary objects from the names of the Enterprise Beans or resources to be invoked, without knowing their storage locations.

Also, when you use a resources adapter compliant with the Connector 1.5 specifications, the objects to be managed are also managed by the naming management. The objects to be managed are the objects used for sending and simultaneously receiving messages from within the J2EE application. For details on the objects to be managed, see *3.16.5 Looking up Administered objects*.

In the JNDI of the naming management functionality, the objects other than the CORBA object references (RMI-IIOP remote objects and JDBC data source objects) are handled as follows:

- The objects other than the CORBA object references are registered by converting the target objects into CORBA objects and by registering the CORBA object references into the CORBA Naming Service.
- To search for objects other than the CORBA objects, the functionality searches for the CORBA object references, reversely convert the CORBA objects into the original objects, and acquire the original objects.

The naming management functionality provided with Application Server includes the functionality provided in J2EE with Application Server-specific enhancements and the functionality that is unique to Application Server. To determine whether the functionality is unique to Application Server, see *1. Application Server Functionality*.

The following table describes the relation between the naming management functionality provided by Application Server and the target objects.

Table 2-2: Relation between the naming management functionality provided by Application Server and the target objects

Functionality	Enterprise Bean		J2EE resource
	EJB home object	Business interface	
Binding and looking up objects in the JNDI name space	Y	Y	Y ^{#1}
Looking up with the Portable Global JNDI names	Y	Y	Y
Looking up with names beginning with HITACHI_EJB	Y	Y	N
Assigning an optional name to the Enterprise Beans or J2EE resources (user-specified name space functionality)	Y	Y	Y
Searching the CORBA Naming Service by using the round-robin policy	Y	Y	N
Caching objects with the naming management functionality	Y	N ^{#2}	N
Switching the CORBA Naming Service	Y	N	N

Legend:

Y: Available

N: Not available

#1: The names beginning with HITACHI_EJB cannot be looked up.

#2: If the J2EE server is restarted by setting the `ejbserver.jndi.cache` property to "on" while a business interface is being used, a `javax.ejb.EJBException` might occur when the business method is executed.

Important note

When a custom error page is set up in a Web application, you cannot use the JNDI from the set error page.

2.2.2 Naming services

When using the naming management functionality, the name and storage location of the object is managed by the *naming service*.

Application Server supports the JNDI that uses the *CORBA Naming Service* as the naming management functionality. When the JNDI interfaces, such as object registration, deletion, or search, are invoked, the interface of the corresponding CORBA Naming Service is invoked. Consequently, to use the naming management functionality, you must specify the protocol, host name, and port number to connect to the CORBA Naming Service.

Reference note

You can start the CORBA Naming Service as an in-process of the J2EE server. By starting the CORBA Naming Service as an in-process, a separate start and stop process becomes redundant, thereby enhancing the operability.

2.3 Binding and looking up objects in the JNDI name space

The objects to be looked up are associated with the names in the JNDI name space and managed.

To reference the objects, such as the resource manager you want to use and the Enterprise Bean you want to invoke, execute a search (lookup) in the JNDI name space with the application to be executed in the J2EE server.

This section describes the types of names used for lookup, rules for bound names, and the lookup mechanism. This section also describes how to check the JNDI name space.

The following table describes the organization of this section.

Table 2-3: Organization of this section (Binding and looking up objects in the JNDI name space)

Category	Title	Reference location
Explanation	Types of names used for lookup	2.3.1
	Mapping and looking up the JNDI name space	2.3.2
	How to check the JNDI name space	2.3.3
Implementation	Definitions in <code>cosminexus.xml</code>	2.3.4
Settings	Execution environment settings	2.3.5

Note: The functionality-specific explanation is not available for "Operations".

2.3.1 Types of names used for lookup

With Application Server, you can use the following four types of names for lookup:

- Looking up with the Portable Global JNDI names
A lookup method for which you specify a name beginning with `java:global`, `java:app`, or `java:module` (Portable Global JNDI name) defined in Java EE. You can use this method to look up the Enterprise Beans or J2EE resources.
- Looking up with names using `java:comp/env`
A lookup method for which you specify a name that uses `java:comp/env` defined in Java EE. You can use this method to look up the Enterprise Beans or J2EE resources.
- Looking up with names beginning with `HITACHI_EJB`
A lookup method for which you specify a name that is automatically bound using the Application Server-specific naming rules. You can use this method to look up Enterprise Beans.
- Looking up with the optional names assigned by the user-specified name space functionality
A lookup method for which you specify the optional name registered by using an Application Server-specific functionality (user-specified name space functionality). You can use this method to look up the Enterprise Beans or J2EE resources.

The following table describes the lookup method that can be used and the recommended lookup method for each form of EJB client.

Table 2-4: Types of lookup methods

Form of EJB Client	Lookup methods			
	Looking up with the Portable Global JNDI names	Looking up with names using <code>java:comp/env</code>	Looking up with names beginning with <code>HITACHI_EJB</code>	Looking up with the optional names assigned by the user-specified name space functionality
EJB client applications	R	N	Y	Y

Form of EJB Client	Lookup methods			
	Looking up with the Portable Global JNDI names	Looking up with names using <code>java:comp/env</code>	Looking up with names beginning with <code>HITACHI_EJB</code>	Looking up with the optional names assigned by the user-specified name space functionality
JSPs, servlets	Y	R	Y	Y
EJBs	Y	R	Y	Y

Legend:

R: Usage is recommended

Y: Available

N: Not available

The following points describe the lookup names and the methods of lookup using those names:

(1) Looking up with the Portable Global JNDI names

The following points describe how to define the names, the lookup range, and features for looking up with the Portable Global JNDI names:

- **Definition method**
A lookup using the Portable Global JNDI names includes lookup with automatically bound names and lookup with names specified in the resource reference definitions.
 - To look up automatically bound names
A name is automatically assigned to the object reference when a J2EE application is deployed on Application Server. This name is defined using the naming rules provided in Java EE and begins with `java:global`, `java:app`, or `java:module`. This name is called the *Portable Global JNDI name*.
 - To look up names specified in the resource reference definitions
You specify the Portable Global JNDI name in the resource reference definition for the DD.
- **Range that can be looked up**
You can look up the Enterprise Beans or J2EE resources in all the J2EE servers or EJB client applications that use the same CORBA Naming Service.
- **Features**
When a J2EE application is deployed, the EJB home object reference corresponding to the Enterprise Bean, and the business interface reference are automatically bound to names beginning with `java:global`, `java:app`, or `java:module`. A name space conflict can be avoided because the standard application name or standard module name is assigned to the bound name. Also, the Portable Global JNDI names are common to Application Servers created by different companies, so you need not amend the source code and the DD when the J2EE applications are migrated between Application Servers.

For details on looking up with the Portable Global JNDI names, see *2.4 Looking up with the Portable Global JNDI names*. Also, for details on the standard application name and standard module name, see *2.4.2 Automatically bound objects*.

(2) Looking up with names using `java:comp/env`

The following points describe how to define the names, the lookup range, and features for looking up names using `java:comp/env`:

- **Definition method**
Defines the mapping of the links between the reference names defined in Java EE using `java:comp/env` and the actual names in the DD resource references. This resolves the link between the reference names and the actual names during deployment.
- **Range that can be looked up**

You can look up in one component. However, in the case of Web applications, you can look up in one Web application.

- Features

`java:comp/env` is the context root of the name space defined in Java EE. You can use `java:comp/env` to look up between different EJB-JAR and Web applications in a J2EE application and to look up J2EE resources.

The specifications for looking up names using `java:comp/env` follow the Java EE provisions.

(3) Looking up with names beginning with HITACHI_EJB

The following points describe how to define the names, the lookup range, and features for looking up names beginning with `HITACHI_EJB`:

- Definition method

A name is automatically assigned to the object reference when a J2EE application is deployed on Application Server. This name is defined using the Application Server-specific naming rules.

- Range that can be looked up

You can look up the Enterprise Beans in all the J2EE servers or EJB applications that use the same CORBA Naming Service.

- Features

When a J2EE application is deployed, the EJB home object reference corresponding to the Enterprise Bean, and the business interface reference are automatically bound to names beginning with `HITACHI_EJB`. A name space conflict can be avoided because the server name and application name of the J2EE application is assigned to the bound name.

For details on looking up with names beginning with `HITACHI_EJB`, see *2.5 Looking up with names beginning with HITACHI_EJB*.

(4) Looking up with the optional names assigned by the user-specified Name Space functionality

The following points describe how to define the names, the lookup range, and features for looking up with the optional names assigned by the user-specified Name Space functionality:

- Definition method

Uses the **user-specified name space functionality** of Application Server to define an optional name for the Enterprise Beans or J2EE resources.

- Range that can be looked up

You can look up the Enterprise Beans or J2EE resources in all the J2EE servers that use the same CORBA Naming Service.

- Features

In the case of the Enterprise Beans, if you use the user-specified Name Space functionality, you can register the optional names for the EJB home object references or business interface references combined with binding the name to the JNDI name. You can also register the optional names for the J2EE resources (resource adapter, mail configuration, and JavaBeans resource).

By registering an optional name, you can look up the Enterprise Beans and J2EE resources with any user-specified name.

For details on assigning optional names for the Enterprise Beans or J2EE resources, see *2.6 Assigning an optional name to the Enterprise Beans or J2EE resources (User-specified name space functionality)*.

Reference note

The Java EE specifications recommend looking up with the names using `java:comp/env`.

2.3.2 Mapping and looking up the JNDI name space

This subsection describes the mechanism of JNDI name space mapping, and the relationship of the name specified for lookup with the JNDI name space and DD.

(1) Mechanism for referencing the Enterprise Beans and the method of usage (ejb-ref)

To invoke the Enterprise Beans with the applications executed in J2EE servers, you need to resolve the names of the Enterprise Bean references. To be precise, you resolve the names such as the names of the EJB home interface, EJB local home interface, and business interface corresponding to the Enterprise Beans.

The following two types of operations are required for resolving the names of the EJB home interfaces or business interfaces:

- **Operations for creating the referencing applications**

You decide the reference name when you create the application and specify that reference name in the `lookup` argument of the programs such as Enterprise Beans and servlets. Also, when using `java:comp/env`, you specify the decided reference name in `<ejb-ref>` of the DD.

- **Operations for deploying the referencing applications**

When you deploy the created application on a J2EE server, customize the Enterprise Beans and link the reference name and the actual name with `linked-to`.

You can use the server management commands to execute this operation.

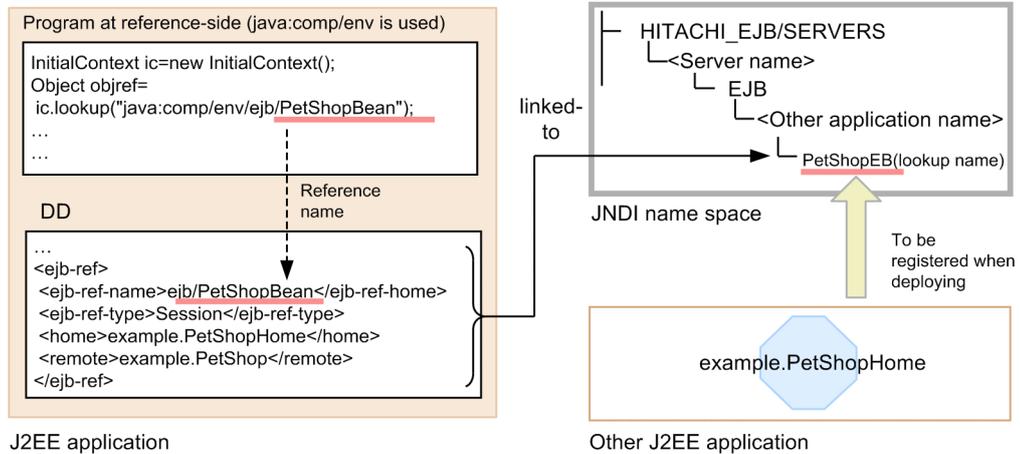
The mapping mechanism differs based on whether the referencing J2EE application is the same as or differs from the referenced J2EE application. The examples of resolving the names of the EJB home interface when you reference the Enterprise Bean of a different J2EE application and when you reference the Enterprise Bean of the same J2EE application are as follows:

(a) When referencing the Enterprise Beans of a different J2EE application

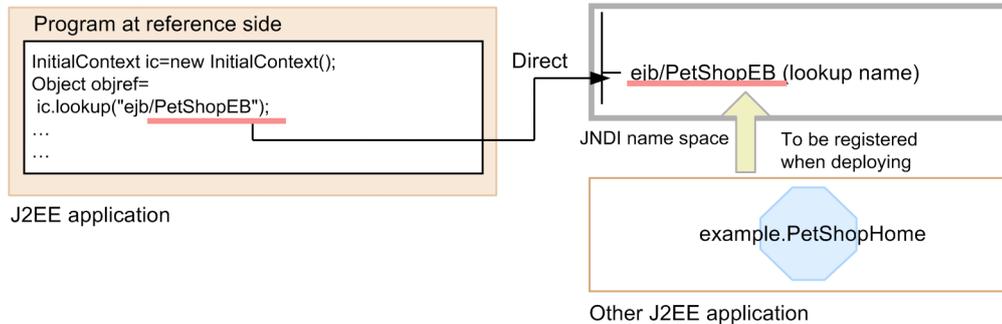
The following figure shows an example of referencing the Enterprise Beans included in a J2EE application from a different J2EE application.

Figure 2-1: Example of referencing the Enterprise Beans included in a J2EE application from a different J2EE application

- When using `java:comp/env`



- When using the user specified name space functionality



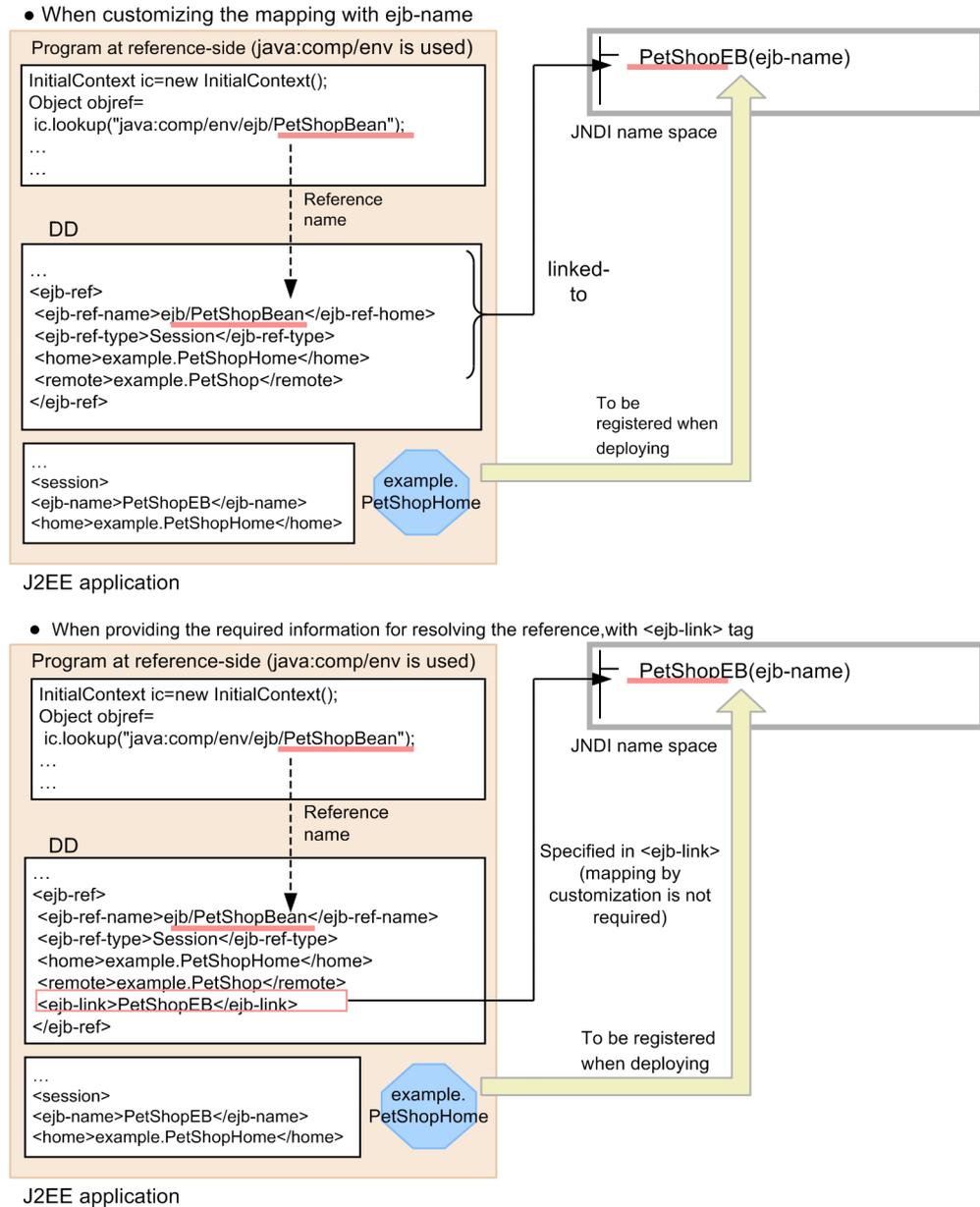
When using `java:comp/env`, the reference name in `java:comp/env` that is specified in the `lookup` argument of the referencing program, and the `lookup` name that is actually registered in the JNDI Name Space are linked by default (`linked-to`). Note that to specify a `lookup` name that is different from the name registered in the JNDI Name Space, you must use the server management commands for customization. The above figure shows a case of customization for invoking `PetShopEB` as `ejb/PetShopBean`.

When using the user-specified name space functionality, you can specify the optional name in the `lookup` argument of the referencing program. You need not define `<ejb-ref>`.

(b) When referencing the Enterprise Beans of the same J2EE application

The following figure shows an example of referencing the Enterprise Beans included in a J2EE application from the same J2EE application.

Figure 2-2: Example of referencing the Enterprise Beans included in the same J2EE application



To invoke the Enterprise Beans of the same application, resolve the references by linking (`linked-to`) `<ejb-ref-name>` and `ejb-name` instead of resolving the references with the `lookup` name. Note that for the same J2EE applications, the information required for referencing is identified when the application is developed, so you can directly write `ejb-name` in the `<ejb-link>` tag of the DD.

(2) Mechanism for referencing the resources and the method of usage (`resource-ref`)

To use a resource adapter in the applications executed on a J2EE server, you need to use lookup to resolve the names of the resource references. To be precise, you resolve the name of the factory that creates a connection to the resource manager.

The following two types of operations are required for resolving the names of the factory that creates the connection to the resource manager:

- Operations for creating the referencing applications

You decide the reference name when you create the application and specify that reference name in the `lookup` argument of the programs such as Enterprise Beans and servlets. Also, specify the decided reference name in `<resource-ref>` of the DD.

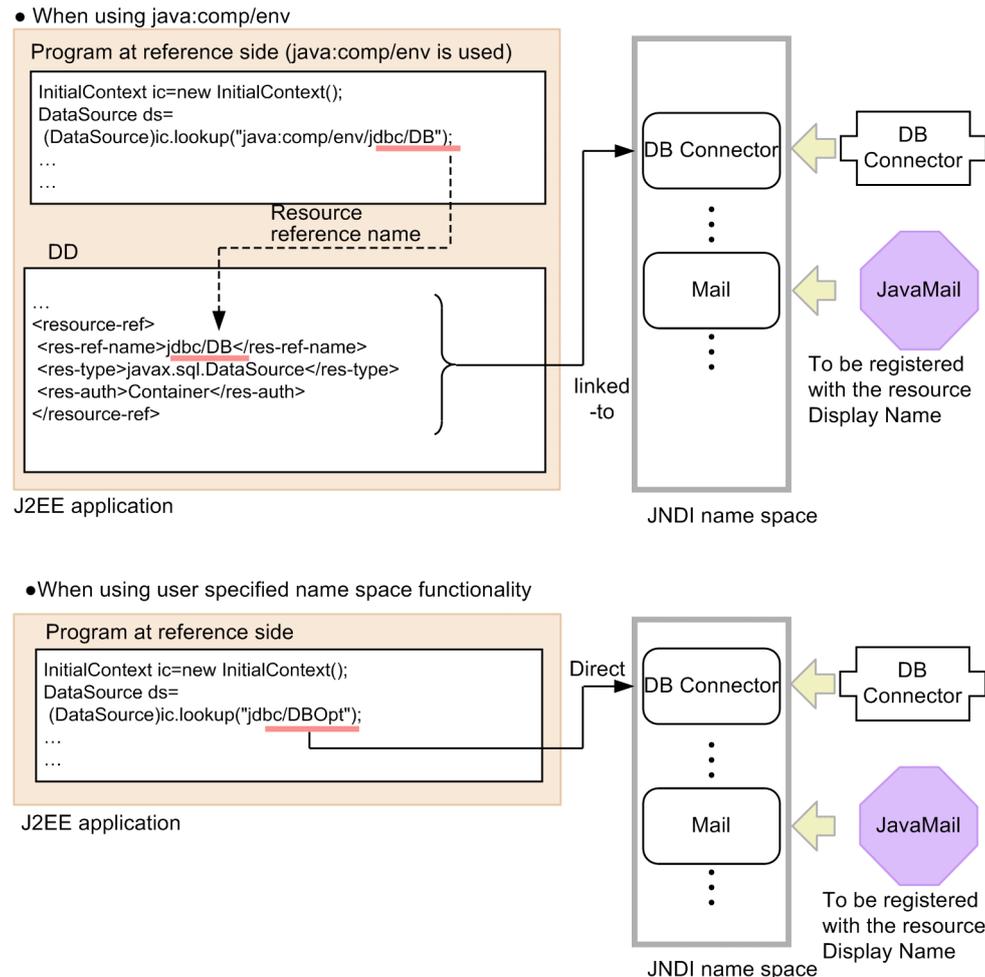
- **Operations for deploying the referencing applications**

When you deploy the created application on a J2EE server, you customize the resource and use `linked-to` to link the reference name and the resource registered on the J2EE server.

You can use the server management commands to execute this operation.

The following figure shows an example of accessing the database via DB Connector.

Figure 2-3: Example of accessing the database via DB Connector



In this example, the reference name and the display name of the resource are linked.

2.3.3 How to check the JNDI name space

You can use commands to display and check the name identified by the CORBA Naming Service. Note that you cannot check the optional names of the J2EE resources and the registered information of the EJB local home object references. The `nsutil` command is used. For details on the usage method and the usage conditions of the `nsutil` command, see the *Borland(R) Enterprise Server VisiBroker(R) Developers Guide*.

The following is an example of execution when the list of contents in the JNDI name space `HITACHI_EJB/SERVERS/MyServer/EJB` is displayed:

```
# nsutil -VBJprop ORBInitRef=NameService=corbaname::localhost:900 list HITACHI_EJB/SERVERS/
MyServer/EJB
Bindings in HITACHI_EJB/SERVERS/MyServer/EJB
Context: Appl
Context: App2
Context: App3
```

2.3.4 Definitions in cosminexus.xml

Define the naming management functionality in the `<ejb-jar>` tag of `cosminexus.xml`. The following table describes the definitions used for naming management in `cosminexus.xml`.

Table 2-5: Definition of the naming management functionality in `cosminexus.xml`

Specified tags	Settings
<code><session>-<lookup-name></code> tag	Specifies the lookup name.
<code><entity>-<lookup-name></code> tag	

For details on `cosminexus.xml`, see 2. *Application Property File (cosminexus.xml)* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

2.3.5 Execution environment settings

When you use the naming management functionality, you must set up the J2EE server and the J2EE applications and customize the server management commands.

(1) Setting up the J2EE server

You specify the J2EE server settings with the Easy Setup definition file. You define the naming management functionality in the `<configuration>` tag of the logical J2EE server (`j2ee-server`) in the Easy Setup definition file. The following table describes the definition of the naming management functionality in the Easy Setup definition file.

Table 2-6: Definition of the naming management functionality in the Easy Setup definition file

Items	Specified parameters	Settings
Basic settings	<code>ejbserver.naming.host#</code>	Specifies the host name of the CORBA Naming Service.
	<code>ejbserver.naming.port#</code>	Specifies the port number of the CORBA Naming Service.
	<code>ejbserver.naming.startupMode</code>	Specifies the startup mode of the CORBA Naming Service.
Timeout in the communication with the naming service	<code>ejbserver.jndi.request.timeout</code>	Specifies the timeout value for the communication with the naming service.

By default, the J2EE server automatically starts and uses the CORBA Naming Service with the host name `localhost` and port number `900` as an in-process. Change these settings as needed.

For details on the Easy Setup definition file and parameters, see 4.6 *Easy Setup definition file* in the *uCosminexus Application Server Definition Reference Guide*.

(2) Customizing the server management commands

You can customize the settings for operating the server management commands. This point describes the settings for the CORBA Naming Service used by the server management commands.

To customize the server management commands for using the naming management functionality, specify the settings in `usrconf.properties` (system property file for the server management commands). The settings are as follows. For details on the keys and the description of the keys not listed here, see *5. Files Used with the Server Management Commands* in the *uCosminexus Application Server Definition Reference Guide*.

Table 2-7: Customizing the server management commands for using the naming management functionality

Items	Specified keys	Settings
Name of the host on which the CORBA Naming Service is operated	<code>ejbserver.naming.host</code>	Specifies the host on which the CORBA Naming Service used by the server management commands is operated.
Port number of the CORBA Naming Service	<code>ejbserver.naming.port</code>	Specifies the port number of the CORBA Naming Service used by the server management commands.
Access protocol for the CORBA Naming Service	<code>ejbserver.naming.protocol</code>	Specifies the access protocol for the CORBA Naming Service used by the server management commands.

(3) Setting up the J2EE applications

You implement the J2EE application settings for the execution environment using the server management commands and property files. To define the naming management functionality, use the Session Bean property file or Entity Bean property file.

The tags specified in these property files correspond to `cosminexus.xml`. For details about the definitions in `cosminexus.xml`, see *2.3.4 Definitions in cosminexus.xml*.

2.4 Looking up with the Portable Global JNDI names

When you deploy a J2EE application, Portable Global JNDI names are automatically bound to the JNDI names of the EJB home object reference and business interface reference. The bound names are used for lookup.

This section describes the name spaces defined in Java EE, objects bound by the Portable Global JNDI names, naming rules, definitions in the DD, and the execution environment settings.

The following table describes the organization of this section.

Table 2-8: Organization of this section (Looking up with the Portable Global JNDI names)

Category	Title	Reference location
Explanation	Types of JNDI name spaces	2.4.1
	Automatically bound objects	2.4.2
	Naming rules for the Portable Global JNDI names	2.4.3
	Controlling the registration of Portable Global JNDI names	2.4.4
	Looking up the Portable Global JNDI names when the CTM is used	2.4.5
	Resource reference names	2.4.6
	Specifying the Portable Global JNDI names in annotations	2.4.7
Implementation	Definitions in the DD	2.4.8
Settings	Execution environment settings	2.4.9

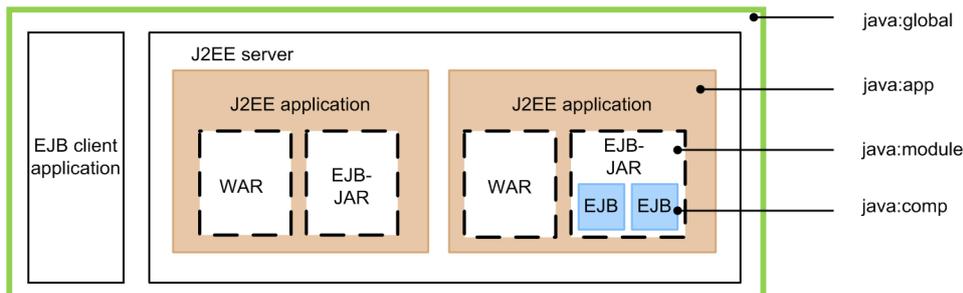
Note: The functionality-specific explanation is not available for "Operations" and "Notes".

2.4.1 Types of JNDI name spaces

This subsection describes the types of JNDI name spaces defined in Java EE and the range that can be looked up.

The following figure shows the range of the JNDI name space.

Figure 2-4: Range of the JNDI name space



Legend:

WAR: Web application
EJB: Enterprise Bean

- `java:global`
This name space is shared by all the J2EE applications. With Application Server, all the J2EE servers or EJB client applications using the same CORBA Naming Service share this name space.
- `java:app`
This name space is shared within a J2EE application. The name space is shared by all the components in one J2EE application (Enterprise Beans, servlets, JSPs, filters, and resource adapters).
- `java:module`

This name space is shared among various modules (EJB-JARs, Web applications, or resource adapters). The name space is shared by all the components in one module (Enterprise Beans, servlets, JSPs, filters, and resource adapters).

- `java:comp`

This name space is shared among various components (Enterprise Beans, servlets, JSPs, filters, or resource adapters). The name space is shared within one component only. However, in the case of a Web application, the name space is shared by all the servlets or JSPs in the Web application.

The name space that can be used depends on whether the component to be looked up is included in the same process, application, module, or component as the lookup source. The following table describes the relationship with the components to be looked up, and the possibility of lookup in each name space.

Table 2-9: Possibility of lookup in each name space

No.	Relationship with the component to be looked up				Name space to be looked up			
	Server (Process)	Application	Module	Component	java:global	java:app	java:module	java:comp
1	Same	Same	Same	Same	Y	Y	Y	Y
2	Same	Same	Same	Different	Y	Y	Y	C #1
3	Same	Same	Different	Different	Y	Y	N	N
4	Same	Different	Different	Different	Y	N	N	N
5	Different#2	Different	Different	Different	Y#3	N	N	N

Legend:

Y: Can be looked up.

C: Can be looked up depending on the component type.

N: Cannot be looked up.

#1

Can be looked up in the case of components in the Web application (servlets, JSPs, or filters).

#2

Indicates that the server (process) is different, but the same CORBA Naming Service is used. Includes the lookup of J2EE applications in a J2EE server from the EJB client application.

#3

Only the automatically bound EJB references can be looked up (Application Server-specific specifications).

2.4.2 Automatically bound objects

With Application Server, the standard application name, standard module name, and EJB reference are automatically bound to the names defined in Java EE. Note that the standard application name and standard module name are defined on Application Server as follows:

- Standard application name

This name uniquely identifies the J2EE application introduced in Java EE 6. With Application Server, the name is called *standard application name* to distinguish from the "J2EE application name" or "application display name" specified in `<display-name>` in `application.xml`.

- Standard module name

This name uniquely identifies various modules in a J2EE application introduced in Java EE 6 (EJB-JAR, Web application, or resource adapter). With Application Server, the name is called *standard module name* to distinguish from the "module name" indicating the names of the EJB-JAR files or WAR files.

The following table describes the automatically bound objects and names.

Table 2-10: Automatically bound objects and names

Bound objects		Bound names
Standard application name (<code>java.lang.String</code> type)		<code>java:app/AppName#1</code>
Standard module name (<code>java.lang.String</code> type)		<code>java:module/ModuleName#1</code>
EJB reference #2	Session Bean home object	<code>java:global[/standard-application-name]/standard-module-name/Enterprise-Bean-name[!fully-qualified-class-name]</code>
		<code>java:app/standard-module-name/Enterprise-Bean-name[!fully-qualified-class-name]</code>
		<code>java:module/Enterprise-Bean-name[!fully-qualified-class-name]</code>
	Session Bean business interface	<code>java:global[/standard-application-name]/standard-module-name/Enterprise-Bean-name[!fully-qualified-class-name]</code>
		<code>java:app/standard-module-name/Enterprise-Bean-name[!fully-qualified-class-name]</code>
		<code>java:module/Enterprise-Bean-name[!fully-qualified-class-name]</code>

#1

Can be looked up only from an EJB-JAR or Web application.

#2

With the EJB 3.1 specifications, the class name omission format in which the fully-qualified class name is omitted can be used for lookup when there is one EJB reference (home object and business interface) for one Enterprise Bean.

Furthermore, with Application Server, the reference registered first can be looked up using the class name omission format in the following order of priority even if there are multiple references:

1. Remote home object
2. Local home object
3. Local business interface (including when the business interface is omitted)
4. Remote business interface

However, we do not recommend lookup in the class name omission format because the specifications related to the class name omission format for multiple references are not provided in Java EE.

If an object that can be looked up with a name beginning with `HITACHI_EJB` exists in an application, you can also use the Portable Global JNDI name for lookup.

Note that when the application starts, the Portable Global JNDI name is output to message `KDJE47701-I` for each bound EJB reference. When the class name or application name omission format of the Enterprise Bean is enabled, all the formats that can be looked up are enumerated using comma and space delimiters (,).

2.4.3 Naming rules for the Portable Global JNDI names

The Portable Global JNDI name includes the standard application name, standard module name, and Enterprise Bean name. Java EE defines the naming rules and characters that can be used for these names.

This subsection describes the naming rules for the standard application name, standard module name, and Enterprise Bean name used in the Portable Global JNDI name, and the characters that can be used.

(1) Naming rules for the standard application name

The following table describes the rules for setting up the standard application name.

Table 2-11: Naming rules for the standard application name

Preconditions for the application				Standard application name
J2EE application (EAR file)	<code>application.xml</code> exists.	The <code><application-name></code> tag exists.	The value exists.	Value of the <code><application-name></code> tag #1

Preconditions for the application				Standard application name
J2EE application (EAR file)	application.xml exists.	The <application-name> tag exists.	The value is null #2.	Default standard application name #3
		The <application-name> tag does not exist.	--	
	application.xml does not exist.	--	--	
WAR application (WAR file)	A value is specified in the -name option of the <code>cjimportwar</code> command			Value of the -name option of the <code>cjimportwar</code> command #1
	No value is specified in the -name option of the <code>cjimportwar</code> command			Default standard application name #3

Legend:

--: Not applicable.

#1

The space characters before and after the value (single-byte spaces, tabs, and linefeeds) are removed. The space characters between the words are not removed.

#2

Indicates a zero length string. A value consisting of only a space character is also determined to be null.

#3

The default standard application name differs depending on the J2EE application format. The following table describes the naming rules for the default standard application name.

Table 2-12: Naming rules for the default standard application name

Precondition for applications		Default standard application name
J2EE application (EAR file)	Archive format	String obtained by removing the extension from the name of the EAR file. However, the extension is not removed if a period "." exists only at the beginning of the file name.
	Exploded archive format	<ul style="list-style-type: none"> If <code>application.xml</code> exists J2EE application name (value of the <display-name> tag). If <code>application.xml</code> does not exist Name of the application directory.
WAR application (WAR file)	Archive format	String obtained by removing the extension from the name of the WAR file. However, the extension is not removed if a period "." exists only at the beginning of the file name.
	Exploded archive format	WAR directory name.

The values that can be specified in the standard application name are as follows:

Standard application name specified in the <application-name> tag

The standard application name is registered if the name contains single-byte alphanumeric characters (0 to 9, A to Z, and a to z) and the following special characters:

Space (), exclamation mark (!), double quotation mark ("), hash mark (#), dollar sign (\$), percent sign (%), ampersand (&), single quotation mark ('), barren (()), asterisk (*), plus sign (+), comma (,), hyphen (-), period (.), colon (:), semicolon (;), less-than sign (<), equal sign (=), greater-than sign (>), question mark (?), at mark (@), square brackets ([]), backslash (\), caret (^), underscore (_), grave accent mark (`), curly brackets ({ }), vertical bar (|), and tilde (~)

However, the following names are not registered:

- Names that begin or end with a period (.)
- Names containing a period (.) only
- Names with a string length of 256 characters or more

- Names matching with `env`

Standard application name specified in the `<display-name>` tag

The standard application name (value of the `<display-name>` tag) is registered if the name contains single-byte alphanumeric characters (0 to 9, A to Z, and a to z), plus (+), hyphen (-), period (.), underscore (_), and caret (^).

However, the following names are not registered:

- Names that begin or end with a period (.)
- Names containing a period (.) only
- Names with a string length of less than 1 character
- Names with a string length of 256 characters or more
- Names matching with `env`

Examples of names that are not registered

`foo/bar`, `.foo`, `.foobar.`, `env`

In the case of names for which the standard application name cannot be registered, `KDJE47710-W` is output when the application starts and the start processing of the application continues. However, the Portable Global JNDI names are not registered for all the objects included in that application, so the Portable Global JNDI names cannot be used for lookup. To look up with the Portable Global JNDI names, as and when required, change the name of the DD or EAR file, or the name of the application directory according to the naming rules.

(2) Naming rules for the standard module name

The following table describes the rules for setting up the standard module name.

Table 2-13: Naming rules for the standard module name

Preconditions for the module				Standard module name
EJB-JAR module (EJB-JAR file)	<code>ejb-jar.xml</code> exists	The <code><module-name></code> tag exists.	The value exists.	Value of the <code><module-name></code> tag #1
			The value is null #2.	Default standard module name #3
		The <code><module-name></code> tag does not exist.	--	
	<code>ejb-jar.xml</code> does not exist	--	--	
Web module (WAR file)	<code>web.xml</code> exists	The <code><module-name></code> tag exists.	The value exists.	Value of the <code><module-name></code> tag #1
			The value is null #2.	Default standard module name #3
		The <code><module-name></code> tag does not exist.	--	
	<code>web.xml</code> does not exist	--	--	
Resource adapter module (RAR file)		--		Default standard module name #3

Legend:

--: Not applicable.

#1

The space characters before and after the value (single-byte spaces, tabs, and linefeeds) are removed. The space characters between the words are not removed.

#2

Indicates a zero length string. A value consisting of only a space character is also determined to be null.

#3

The default standard module name differs depending on the module and J2EE application format. The following table describes the naming rules for the default standard module name.

Table 2-14: Naming rules for the default standard module name

Module	J2EE application format	Default standard module name
EJB-JAR module (EJB-JAR file)	Archive format	String obtained when you remove the extension from the relative path from the root directory of the application package up to the EJB-JAR file. However, the extension is not removed if a period "." exists only at the beginning of the EJB-JAR file name. If the path delimiter is \, the character is converted to /.
	Exploded archive format	<ul style="list-style-type: none"> If <code>application.xml</code> exists The path specified in the <code><module>/<ejb></code> tag of <code>application.xml</code>. If <code>application.xml</code> does not exist String obtained when you remove the last <code>_jar</code> from the relative path from the application directory up to the EJB-JAR directory. If the path delimiter is \, the character is converted to /.
Web module (WAR file)	Archive format	String obtained when you remove the extension from the relative path from the root directory of the application package up to the WAR file. However, the extension is not removed if a period "." exists only at the beginning of the WAR file name. If the path delimiter is \, the character is converted to /.
	Exploded archive format	<ul style="list-style-type: none"> If <code>application.xml</code> exists The path specified in the <code><module>/<web>/<web-uri></code> tag of <code>application.xml</code>. If <code>application.xml</code> does not exist String obtained when you remove the last <code>_war</code> from the relative path from the application directory up to the WAR directory. If the path delimiter is \, the character is converted to /.
Resource adapter module	Archive format	String obtained when you remove the extension from the relative path from the root directory of the application package up to the RAR file. However, the extension is not removed if a period "." exists only at the beginning of the RAR file name. If the path delimiter is \, the character is converted to /.

Note that the standard module name (value of the `<module-name>` tag) is registered if the name contains single-byte alphanumeric characters (0 to 9, A to Z, and a to z) and the following special characters:

Space (), exclamation mark (!), double quotation mark ("), hash mark (#), dollar sign (\$), percent sign (%), ampersand (&), single quotation mark ('), barren (()), asterisk (*), plus sign (+), comma (,), hyphen (-), period (.), forward slash as a delimiter (/), colon (:), semicolon (;), less-than sign (<), equal sign (=), greater-than sign (>), question mark (?), at mark (@), square brackets ([]), backslash (\), caret (^), underscore (_), grave accent mark (`), curly brackets ({}), vertical bar (|), and tilde (~)

However, the following names are not registered:

- Names that begin or end with a forward slash (/)
- Names containing a forward slash (/) only
- Names with a series of forward slashes (/)
- Names that begins or ends with a period (.)
- Names containing a period (.) only

2. Naming Management

- Names with a series of forward slashes (/) and periods (.)
- Names with a string length of 256 characters or more
- Names matching with `env`
- Names beginning with `env/`
- Names matching with `AppName`
- Names beginning with `AppName/`

Examples of names that are not registered

```
/foo, /foobar/, /, foo//bar, bar., .foobar., foo/.bar, AppName, AppName/foo,
env/foo/bar
```

In the case of names for which the standard module name cannot be registered, KDJE47711-W is output for each module when the application starts and the start processing of the application continues. However, the Portable Global JNDI names are not registered for all the objects included in that module, so the Portable Global JNDI names cannot be used for lookup. To look up with the Portable Global JNDI names, as and when required, change the name of the DD or the name of the application directory according to the naming rules.

(3) Naming rules for the Enterprise Bean name

A name is registered as the Enterprise Bean name if the name contains single-byte alphanumeric characters (0 to 9, A to Z, and a to z) and the following special characters:

Space (), exclamation mark (!), double quotation mark ("), hash mark (#), dollar sign (\$), percent sign (%), ampersand (&), single quotation mark ('), barren (() ()), asterisk (*), plus sign (+), comma (,), hyphen (-), period (.), colon (:), semicolon (;), less-than sign (<), equal sign (=), greater-than sign (>), question mark (?), at mark (@), square brackets ([]), backslash (\), caret (^), underscore (_), grave accent mark (`), curly brackets ({} {}), vertical bar (|), and tilde (~)

However, the following names are not registered:

- Names that begin or end with a period (.)
- Names containing a period (.) only
- Names with a string length of 256 characters or more
- Names matching with `ModuleName`
- Names matching with `env`

Examples of names that are not registered

```
.foo, .foobar., ModuleName, env
```

In the case of names for which the Enterprise Bean name cannot be registered, KDJE47712-W is output for each Enterprise Bean when the application starts and the start processing of the application continues. However, the registration to the Portable Global JNDI names is not performed for all the objects included in that Enterprise Bean. Note that the names for which the Enterprise Bean name cannot be registered cannot be looked up with the Portable Global JNDI names, but all the other application functionality operate as before.

Furthermore, the specifiable string length is checked for each Enterprise Bean interface (home interface or business interface) as well. For example, if the following string length is 256 characters or more, KDJE47713-W is output for each interface, and the name is not registered in the Portable Global JNDI name with the format specifying that interface name.

```
Length-of-Enterprise-Bean-name + length-of-interface-class-name# + 1
```

Fully-qualified class name containing the package name

Note that in EJB 3.1, if the interface is omitted, the Enterprise Bean class name is applicable instead of the interface class name.

(4) Operations when the standard application name or standard module name are duplicated

With Application Server, if the same standard application name is already registered for a name space when an application starts, KDJE47720-W is output, and the name is not registered for the name space. Similarly, if the same standard module name is already registered for a name space in the same application, the following message is output and the name is not registered for the name space. However, the start processing of the application continues.

- For an EJB-JAR: KDJE47721-W
- For a Web application: KDJE47722-W
- For a resource adapter: KDJE47723-W

Note that the modules are registered in the following order with Application Server:

1. Resource adapter
2. EJB-JAR
3. Web application

If the registered object and name match in the hierarchy demarcated with forward slashes (/), even if the name does not exactly match the registered standard application name or standard module name, the name might be determined to be a duplicate. The examples are as follows:

Example 1

If the applications are started in the following order, the standard application name is determined to be duplicated:

1. An application with the standard application name `foo/bar` is started
2. An application with the standard application name `foo` is started

Example 2

If the applications are started in the following order, the standard application name is determined to be duplicated:

1. An application with the standard application name `foo` and the standard module name `bar` is started
2. An application with the standard application name `foo/bar` is started

2.4.4 Controlling the registration of Portable Global JNDI names

In Application Server version 09-00 and later, if you do not want to look up with the Portable Global JNDI names, specify the following settings in the `<configuration>` tag of the logical J2EE server (`j2ee-server`) in the Easy Setup definition file. When you specify these settings, the Portable Global JNDI name is not registered when the application starts. Furthermore, the information and warnings related to the Portable Global JNDI name are also not displayed.

- Specify `false` in the `ejbserver.jndi.global.enabled` parameter.

2.4.5 Looking up with the Portable Global JNDI names when the CTM is used

If the system configuration uses the CTM, you can acquire the EJB remote home object by looking up the CORBA Naming Service that is connected to the CTM daemon from the EJB client application. When you look up the EJB, you can use the names beginning with `HITACHI_EJB`, the Portable Global JNDI name, and the optional name assigned by using the user-specified name space functionality.

During EJB lookup when the CTM is used, the optional name is used if the optional name is specified for the EJB. If the optional name is not specified, the default lookup name is used. The default lookup name is switched using the `ejbserver.ctm.useGlobalJNDI` parameter in the `<configuration>` tag of the logical J2EE server (`j2ee-server`) in the Easy Setup definition file.

The following table describes the names used for EJB lookup when the CTM is used.

Table 2-15: Names used for EJB lookup when the CTM is used

Preconditions				Lookup name
Specification of the optional name for EJB	Value of the <code>ejbserver.jndi.global.enabled</code> parameter #1	Value of the <code>ejbserver.ctm.useGlobalJNDI</code> parameter #2	Can the Portable Global JNDI name be registered #3	
Yes	--	--	--	Optional name for the EJB
No	true (default)	true	Can be registered	Portable Global JNDI name
			Cannot be registered	Name beginning with <code>HITACHI_EJB</code>
	false (default)	--		
	false	--	--	

Legend:

--: Not applicable

#1

Indicates the value specified in the `ejbserver.jndi.global.enabled` parameter of the Easy Setup definition file. If the value is `true`, the Portable Global JNDI name is registered when the application starts. If the value is `false`, the Portable Global JNDI name is not registered.

#2

Indicates the value specified in the `ejbserver.ctm.useGlobalJNDI` parameter of the Easy Setup definition file. If the value is `true`, the Portable Global JNDI name is used for EJB lookup in a system configuration where the CTM is used. If the value is `false`, a name beginning with `HITACHI_EJB` is used.

#3

Indicates whether the EJB can be registered with the Portable Global JNDI name. If the standard application name or standard module name is duplicated, or if the string cannot be registered, the Portable Global JNDI name cannot be registered.

2.4.6 Resource reference names

In Java EE 6 and later, apart from the names using `java:comp/env`, you can specify the fully qualified JNDI name in the resource reference name. By specifying the fully qualified JNDI name in the resource reference name, you can define the resource reference shared by all the components in one application (Enterprise Beans, servlets, JSPs, filters, and resource adapters) and the resource reference shared by all the components in the EJB-JAR or Web application.

Also, you can look up the other resource references with the fully qualified JNDI name by specifying the `<lookup-name>` tag in the DD (`web.xml`), or the `lookup` attribute of the `@Resource` and `@EJB` annotations.

This subsection describes the specification of the fully qualified JNDI name for the resource reference name, and the look up of the other resource references.

! Important note

If the registration of the Portable Global JNDI names is controlled by specifying `false` in the `ejbserver.jndi.global.enabled` parameter, do not specify a name beginning with `java:` in the resource reference name.

(1) Specifying the fully qualified JNDI name for the resource reference name

With Application Server, if you specify a name beginning with `java:` in the resource reference name when the registration of the Portable Global JNDI name is enabled, the fully qualified JNDI name is determined to have been specified and the specified name is handled as the JNDI name as is.

If the registration of the Portable Global JNDI name is disabled, or if you specify a name beginning with a string other than `java:`, the name is handled as a name beginning with `java:comp/env` or as an optional name of the resource adapter.

The following table describes the resource reference types for which you can specify the fully qualified JNDI name as the resource reference name in the DD (`web.xml`).

Table 2-16: Resource reference types for which you can specify the fully qualified JNDI name (DD)

No	Resource reference types	Tags specifying the resource reference name
1	Environment entry	<code><env-entry></code> - <code><env-entry-name></code>
2	Resource reference	<code><resource-ref></code> - <code><res-ref-name></code>
3	Resource environment variable reference	<code><resource-env-ref></code> - <code><resource-env-ref-name></code>
4	Remote EJB reference	<code><ejb-ref></code> - <code><ejb-ref-name></code>
5	Local EJB reference	<code><ejb-local-ref></code> - <code><ejb-local-ref-name></code>
6	Persistence context reference	<code><persistence-context-ref></code> - <code><persistence-context-ref-name></code>
7	Persistence unit reference	<code><persistence-unit-ref></code> - <code><persistence-unit-ref-name></code>

The following table describes the resource reference annotations for which you can specify the fully qualified JNDI name in the `name` attribute.

Table 2-17: Resource reference types for which you can specify the fully qualified JNDI name (Annotations)

No	Resource reference types	Annotations specifying the resource reference name
1	Environment entry	name attribute of <code>@Resource</code>
2	Resource reference	
3	Resource environment variable reference	
4	Remote EJB reference	name attribute of <code>@EJB</code>
5	Local EJB reference	
6	Persistence context reference	name attribute of <code>@PersistenceContext</code>
7	Persistence unit reference	name attribute of <code>@PersistenceUnit</code>

If you specify the fully qualified JNDI name in the resource reference name when the registration of the Portable Global JNDI name is enabled, depending on the combination of whether the specified name space, and standard application name and standard module name violate the naming rules, KDJE47730-E is output when the application is starting and the application fails to start.

The following table describes the specified name space of the fully qualified JNDI name and whether the application can be started. Note that even if you specify a name that begins with `java:`, but without a specifiable name space, KDJE47730-E is output when the application is starting and the application fails to start.

Table 2-18: Specified name space of the fully qualified JNDI name and can the application be started

Preconditions		Specified name space			
Standard application name of the application defining the resource reference	Standard module name of the EJB-JAR or Web application defining the resource reference	java:comp	java:module	java:app	java:global
Invalid value or duplication	--	Y	N	N	Y

2. Naming Management

Preconditions		Specified name space			
Standard application name of the application defining the resource reference	Standard module name of the EJB-JAR or Web application defining the resource reference	java:comp	java:module	java:app	java:global
Normal	Invalid value or duplication	Y	N	Y	Y
	Normal	Y	Y	Y	Y

Legend:

Y: The application can be started.

N: The application cannot be started.

--: Not applicable.

With Application Server, there are no constraints on the sub-context name when you specify the resource reference name for the fully qualified JNDI name. However, if a context or object is already registered with the same name, KDJE47721-W is output when the application is starting and the application fails to start.

Note that even when you define the Portable Global JNDI name in the resource reference name, that resource reference cannot be looked up from the J2EE server or EJB client application of another process. If you look up a resource reference on another process in which the Portable Global JNDI name is defined, the `NameNotFoundException` exception occurs.

(2) Looking up the other resource references

With Application Server, by specifying the `<lookup-name>` tag in `web.xml`, or the `lookup` attribute in the `@Resource` and `@EJB` annotations, in addition to specifying the resource references that have been registered using the names beginning with `java:comp/env`, you can look up the resources shared by the EJB-JAR or Web applications and the resources shared by an application through the DI functionality and resource references.

For example, the DI functionality can also be executed for the environment entry defined in the `java:app` name space in `web.xml` within a Web application from the EJB in the EJB-JAR that is a separate module.

You can use the resource adapter name and the EJB optional name in the names that can be looked up from the resource references.

The following table describes the names that can be looked up from the resource references.

Table 2-19: Names that can be looked up from the resource references

Lookup source resource reference		Name that can be looked up
Annotation	DD (web.xml)	
lookup attribute of <code>@Resource</code> #1	<code><env-entry></code> - <code><lookup-name></code> #2	Environment entry name (fully qualified JNDI name)
	<code><resource-ref></code> - <code><lookup-name></code> #1	Resource reference name (fully qualified JNDI name) Optional name of the resource adapter
	<code><resource-env-ref></code> - <code><lookup-name></code> #1	Resource environment variable reference name (fully qualified JNDI name)
lookup attribute of <code>@EJB</code>	<code><ejb-ref></code> - <code><lookup-name></code>	EJB remote reference name (fully qualified JNDI name) Portable Global JNDI name of the EJB remote object Optional name of the EJB remote object

Lookup source resource reference		Name that can be looked up
Annotation	DD (web.xml)	
lookup attribute of @EJB	<ejb-local-ref> - <lookup-name>	Local EJB reference name (fully qualified JNDI name)
		Portable Global JNDI name of the local EJB object
		Optional name of the local EJB object

#1

According to the Java EE specifications, if the data type in the injection destination or the data type set in the `<resource-ref-type>` tag and `<resource-env-ref-type>` tag of `web.xml` is one of the following types, the settings in the lookup attribute of the `@Resource` annotation or the `<lookup-name>` tag are ignored (the operation is the same as when the settings are not specified):

```
org.omg.CORBA.ORB
org.omg.CORBA_2_3.ORB
javax.ejb.EJBContext
javax.ejb.SessionContext
javax.ejb.TimerService
javax.transaction.UserTransaction
javax.validation.Validator
javax.validation.ValidatorFactory
javax.enterprise.inject.spi.BeanManager
```

#2

According to the Java EE specifications, if a value is set in the `<env-entry-value>` tag, the settings in the lookup attribute of the `@Resource` annotation or the `<lookup-name>` tag are ignored (the operation is the same as when the settings are not specified).

Note that if the `<linked-to>` tag, `<linked-queue>` tag, or `<linked-adminobject>` tag is specified in the property file, and the `mappedName` attribute is specified in the `@Resource` annotation, the settings in the lookup attribute of the `@Resource` annotation and the `<lookup-name>` tag in `web.xml` are ignored.

! Important note

Do not set the lookup destination resource reference as the lookup source and do not set a circular reference. If you define the resource references so that they form a circular reference, an infinite recursive call occurs when you look up from an application or when you execute the DI functionality, the `java.lang.StackOverflowError` exception occurs, and the application does not return a response.

An example of a circular reference is as follows:

```
<env-entry>
<env-entry-name>java:app/env/sample1</env-entry>
<lookup-name>java:app/env/sample2</env-entry>
</env-entry>
<env-entry>
<env-entry-name>java:app/env/sample2</env-entry>
<lookup-name>java:app/env/sample1</env-entry>
</env-entry>
```

For this definition, if you look up `java:app/env/sample1` from the application, an infinite recursive call occurs, and the `java.lang.StackOverflowError` exception is thrown.

2.4.7 Specifying the Portable Global JNDI names in annotations

With Application Server, you can specify the Portable Global JNDI name in the `lookup` and `mapped` attributes of the `@EJB` annotation, according to the Java EE specifications. Due to this, you can directly inject the EJB business interface reference or home object reference.

If you specify settings matching the following conditions in the `@EJB` annotation, regardless of the injected Session Bean type, the DI functionality is implemented every time a business method is invoked, or the timeout callback method is invoked.

- If the format `<module-name>/<bean-name>` is specified in the `beanName` attribute
- If the Portable Global JNDI name is specified in the `lookup` attribute
- If the Portable Global JNDI name is specified in the `mappedName` attribute

2.4.8 Definitions in the DD

This subsection describes the necessary DD definitions when you use the Portable Global JNDI names for look up.

You specify the definitions for look up with the Portable Global JNDI names in `application.xml`, `ejb-jar.xml`, and `web.xml`.

The following table describes the definitions in the DD related to looking up with the Portable Global JNDI names.

Table 2-20: Definitions in the DD for looking up with the Portable Global JNDI names

DD type	Tag name	Settings
<code>application.xml</code>	<code><application></code> - <code><application-name></code>	Specifies the standard application name.
<code>ejb-jar.xml</code>	<code><ejb-jar></code> - <code><module-name></code>	Specifies the standard module name.
<code>web.xml</code>	<code><web-app></code> - <code><module-name></code>	Specifies the standard module name. If the tag is specified multiple times, the value set in the tag specified first is applied, and the values set in the second and subsequent tags are ignored.
	<code><web-app></code> - <code><env-entry></code> - <code><lookup-name></code>	Specifies the Portable Global JNDI name that references another <code>env-entry</code> .
	<code><web-app></code> - <code><resource-ref></code> - <code><lookup-name></code>	Specifies the Portable Global JNDI name that references another <code>resource-ref</code> .
	<code><web-app></code> - <code><resource-env-ref></code> - <code><lookup-name></code>	Specifies the Portable Global JNDI name that references another <code>resource-env-ref</code> .
	<code><web-app></code> - <code><ejb-ref></code> - <code><lookup-name></code>	Specifies the Portable Global JNDI name that references another <code>ejb-ref</code> , or the Portable Global JNDI name that binds the remote business interfaces.
	<code><web-app></code> - <code><ejb-local-ref></code> - <code><lookup-name></code>	Specifies the Portable Global JNDI name that references another <code>ejb-local-ref</code> , or the Portable Global JNDI name that binds the local business interfaces.

! Important note

The contents specified in the `<module-name>` tag and `<lookup-name>` tag existing in `cosminexus.xml` and HITACHI Application Integrated Property File differ from the contents specified in the `<module-name>` tag and `<lookup-name>` tag set in the above `ejb-jar.xml` or `web.xml`.

2.4.9 Execution environment settings

To look up with the Portable Global JNDI names, you must specify the J2EE server settings.

You implement the J2EE server settings using the Easy Setup definition file. You specify the definition for looking up with the Portable Global JNDI names in the `<configuration>` tag of the logical J2EE server (`j2ee-server`) in the Easy Setup definition file. The following table describes the settings in the Easy Setup definition file.

Table 2-21: Definitions in the Easy Setup definition file for looking up with the Portable Global JNDI names

Item	Specified parameter	Settings
Registering the Portable Global JNDI name	<code>ejbserver.jndi.global.enabled</code>	Specifies whether to register the Portable Global JNDI name for the naming service when the application starts.
Default lookup name when the CTM is used	<code>ejbserver.ctm.useGlobalJNDI</code>	Specifies the default lookup name to be used when the optional name is not specified in the EJB, when the CTM is used.

Reference note

If you specify the settings to register the Portable Global JNDI name, the information about the registered name and the warnings, such as the duplication of the standard application name, are displayed on the console when the application starts.

For details on the Easy Setup definition file and parameters, see *4.6 Easy Setup definition file* in the *uCosminexus Application Server Definition Reference Guide*.

2.5 Looking up with names beginning with HITACHI_EJB

This section describes the names beginning with `HITACHI_EJB`. When you deploy a J2EE application, a name beginning with `HITACHI_EJB` is automatically bound to the JNDI name of the EJB home object reference and business interface reference. The bound name is used for lookup.

For details on the mapping mechanism and usage of the JNDI name space, see *2.3.2 Mapping and looking up the JNDI name space*.

! Important note

The names beginning with `HITACHI_EJB` are not bound when a local interface is used. Use another method for lookup.

(1) Name that automatically binds the EJB home object reference

When you start (deploy) a J2EE application, the EJB home object reference of the Enterprise Bean is bound to the JNDI name with the following name:

`HITACHI_EJB/SERVERS/server-name/EJB/J2EE-application-name/Enterprise-Bean-name`

server-name

Name of the J2EE server

J2EE-application-name

Lookup name of the J2EE application

Enterprise-Bean-name

Lookup name of the Enterprise Bean

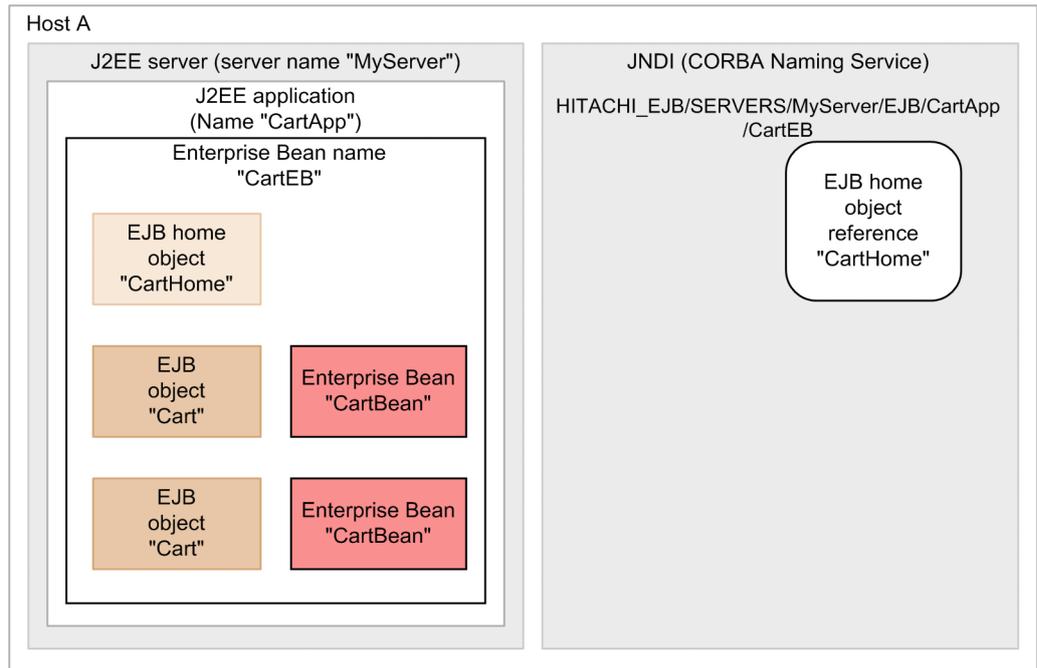
When Enterprise Beans are invoked between the J2EE applications, or Enterprise Beans are invoked from the EJB client application, the client looks up the EJB home object reference with the bound JNDI name.

The following figure shows that when a J2EE application is started with the below-mentioned conditions, an EJB home object implementing the "CartHome" interface is generated, and that reference is bound to the JNDI name "HITACHI_EJB/SERVERS/MyServer/EJB/CartApp/CartEB".

Conditions

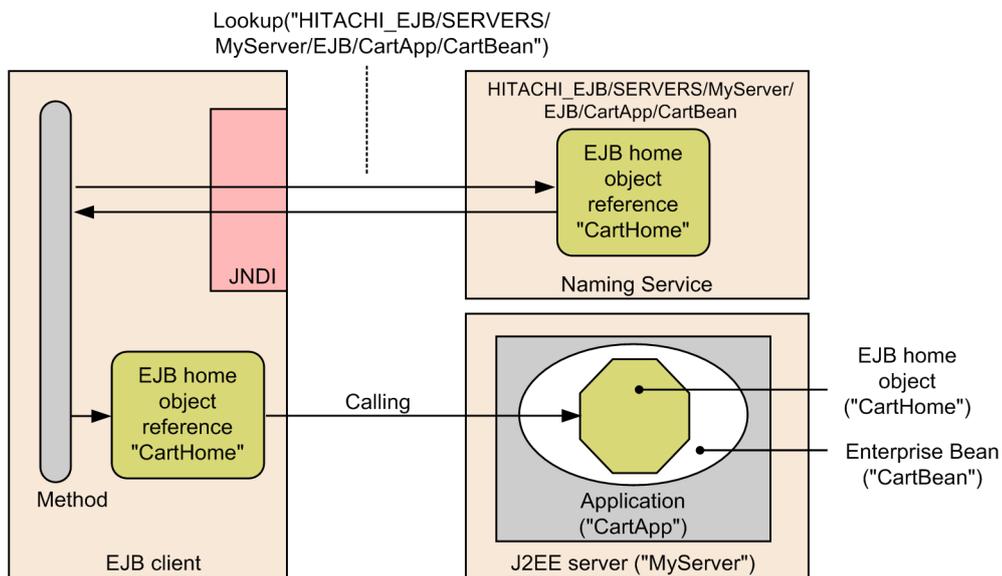
- Enterprise Bean: "CartBean"
- Remote interface name: "Cart"
- Home interface: "CartHome"
- Server name: "MyServer"
- Lookup name of the J2EE application: "CartApp"
- Lookup name of the Enterprise Bean: "CartEB"

Figure 2-5: Binding the EJB home object reference to the JNDI name space



The following figure shows the procedure of lookup and obtaining of objects when the EJB home object reference is looked up using a name beginning with HITACHI_EJB.

Figure 2-6: Procedure of lookup and obtaining of objects using a name beginning with HITACHI_EJB



(2) Name that automatically binds the business interface reference

When you start (deploy) a J2EE application, the business interface reference is bound to the JNDI name with the following name:

`HITACHI_EJB/SERVERS/server-name/EJBBI/J2EE-application-name/Enterprise-Bean-name`

server-name

Name of the J2EE server

2. Naming Management

J2EE-application-name

Lookup name of the J2EE application

Enterprise-Bean-name

Lookup name of the Enterprise Bean

When Enterprise Beans are invoked between the J2EE applications, or Enterprise Beans are invoked from the EJB client application, the client looks up the business interface reference with the bound JNDI name.

2.6 Assigning an optional name to Enterprise Beans or J2EE resources (User-specified name space functionality)

The functionality with which the user assigns an optional name and registers the Enterprise Beans or J2EE resources in JNDI name space is called the *user-specified name space functionality*. This functionality enables you to lookup the Enterprise Beans or J2EE resources using any specified name.

Note that you must customize the settings for operating the server management commands to assign an optional name. For details on how to specify the settings, see 2.6.7 *Execution environment settings*.

! Important note

In the Java EE specifications, we recommend that you perform lookup with the name using `java:comp/env`.

The following table describes the organization of this section.

Table 2-22: Organization of this section (User-specified name space functionality)

Category	Title	Reference location
Explanation	Objects that can be assigned an optional name	2.6.1
	Rules for assigning the optional names	2.6.2
	Timing for registering or deleting the optional name	2.6.3
	Searching from the client	2.6.4
Implementation	Setting the optional names for the Enterprise Beans	2.6.5
	Setting the optional names for the J2EE resources	2.6.6
Settings	Execution environment settings	2.6.7
Notes	Precautions for using the user-specified name space functionality	2.6.8

Note: the function-specific explanation is not available for "Operations".

2.6.1 Objects that can be assigned an optional name

This subsection describes the objects that can be assigned an optional name.

You assign optional names to the Enterprise Beans or J2EE resources.

(1) Optional names for Enterprise Beans

An optional name is assigned to Enterprise Beans with the following interfaces:

- Remote home interface
- Local home interface
- Remote business interface
- Local business interface

Hereafter, the remote home interface and remote business interface are together called the *remote interface*. Also, the local home interface and local business interface are together called the *local interface*.

Reference note

The optional names for the remote interface and local interface are set as different attributes. For details, see 2.6.5 *Setting the optional names for the Enterprise Beans*.

(2) Optional names for J2EE resources

The following table describes the J2EE resources to which you can assign optional names.

Table 2-23: J2EE resources for which optional names can be assigned

Types of J2EE resources		Possibility of assigning an optional name
Resource adapter	DB Connector	Y#1
	DB Connector for Cosminexus RM	Y
	Cosminexus RM	Y#2
	uCosminexus TP1 Connector	Y
	TP1/Message Queue - Access	Y#2
	Other resources adapters	Y#3
Mail configuration		Y
JavaBeans resource		Y

Legend:

Y: Optional name is assigned

#1 When the connection pool clustering functionality is used, optional names are not assigned to the member resource adapters.

#2 An optional name is assigned to the `javax.jms.ConnectionFactory` object specified in `<resource-env-ref>` of the property file. An optional name is not assigned to the `javax.jms.Destination` object specified in `<resource-env-ref>`.

#3 For the resource adapters compliant with the Connector 1.5 specifications, you cannot assign an optional name to an administered object.

2.6.2 Rules for assigning the optional names

This subsection describes the rules for assigning the optional names to Enterprise Beans and J2EE resources.

(1) Specifiable names

This subsection describes the characters that can be specified as the optional name and the specification constraints.

(a) Characters that can be specified in an optional name

You can specify a name containing the following characters as an optional name:

- Alphanumeric characters (A-Z, a-z, 0-9)
- Underscore (`_`)
- Forward slash (`/`)
- Period (`.`)
- Hyphen (`-`)

However, you can use the forward slash (`/`) in the name only when the forward slash is used as a delimiter.

(b) Constraints on optional name specification

You cannot specify the following names as optional names. If you specify the following names, the J2EE applications or J2EE resources cannot be started:

- Name that begins or ends with a forward slash (`/`) or period (`.`).
- Name that only contains a forward slash (`/`) or period (`.`).
- Name with a series of forward slashes (`/`).

- Name with a series of forward slashes (/) and periods (.).
- Name that begins with HITACHI_EJB (case-sensitive).
- Name with a string length longer than 255 bytes.

Apart from the above, for the J2EE resources, the definition specified later becomes valid when the same name is specified.

(2) Duplication of optional names

The following table describes whether the duplication of the optional name is allowed when the optional names are assigned to Enterprise Beans and J2EE resources.

Table 2-24: Duplication of the optional names for Enterprise Beans and J2EE resources

Target for assigning the optional name	Enterprise Beans with remote interface	Enterprise Beans with local interface	Resource adapters included in the J2EE application	J2EE resources #1
Enterprise Beans with remote interface	N	Y#2	N	N
Enterprise Beans with local interface	Y#2	C	C	N
Resource adapters included in the J2EE application	N	C	C	N
J2EE resources #1	N	N	N	N

Legend:

- Y: Can be duplicated
- C: Can be duplicated if the J2EE application is different
- N: Cannot be duplicated

#1 Excluding the resource adapters included in the J2EE application.

#2 Can be duplicated regardless of whether the local call optimization functionality is used.

Note that for the combinations in which the optional names cannot be duplicated, the specification of the optional name might be disallowed even if the name does not match completely.

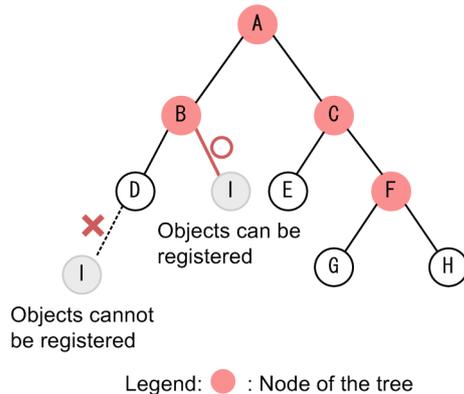
This is described with the following example:

Example

The naming management functionality manages the objects registered in the CORBA Naming Service by names. The name assigned to the object analyzes the forward slash (/) as a delimiter of the hierarchical structure and is managed in a tree structure. In the tree structure, you can register a new object below the nodes that have child nodes.

In the following figure, nodes A, B, C, and F have child nodes. Therefore, new objects can be registered below these nodes.

Figure 2-7: Example wherein the optional name cannot be specified although the name does not match completely



In such a case, if the optional name 'A/B/D' is already in use, you can specify the optional name 'A/B/I'. However, new objects cannot be registered in nodes such as D, E, G, and H. Therefore, the optional name 'A/B/D/I' cannot be specified.

If an optional name that cannot be duplicated is specified, an error occurs at the following timings in each object where the optional name is specified:

- In the case of the Enterprise Beans, the J2EE application fails to start.
- In the case of J2EE resources other than mail configuration, the resource fails to start.
- In the case of mail configuration, an error occurs during attribute settings.

2.6.3 Timing for registering or deleting the optional name

This subsection describes how to set the optional names for the Enterprise Beans or J2EE resources, the timing for registering the optional name, and the timing for deleting the optional name.

(1) Timing for registering or deleting the optional names of Enterprise Beans

The optional name specified in the Enterprise Bean object is registered in the name space when the J2EE application is started or when the J2EE server is started.

The optional name specified in the Enterprise Bean object is deleted from the name space when the J2EE application is stopped or when the J2EE server is stopped.

Reference note

When the log level is set to "Warning", you can check whether the optional name is registered or deleted in the message log.

- When the optional name is registered: KDJE47605-I is output.
- When the optional name is deleted: KDJE47606-I is output.

However, these messages are not output in the default log level settings. For details on the log level settings, see 3.3.6 *Log collection settings for J2EE servers* in the *uCosminexus Application Server Maintenance and Migration Guide*.

(2) Timing for registering or deleting the optional names of the J2EE resources

The optional name for a J2EE resource is registered in the name space when the J2EE resource is started.

The optional name for the J2EE resource is deleted from the name space when the J2EE resource is stopped.

Reference note

You can check whether the optional name was registered or deleted through the message log.

- When the optional name is registered: KDJE47602-I is output.

- When the optional name is deleted: KDJE47603-I is output.

2.6.4 Searching from the client

This subsection describes how to search the Enterprise Beans or J2EE resources that are assigned optional names, from the client.

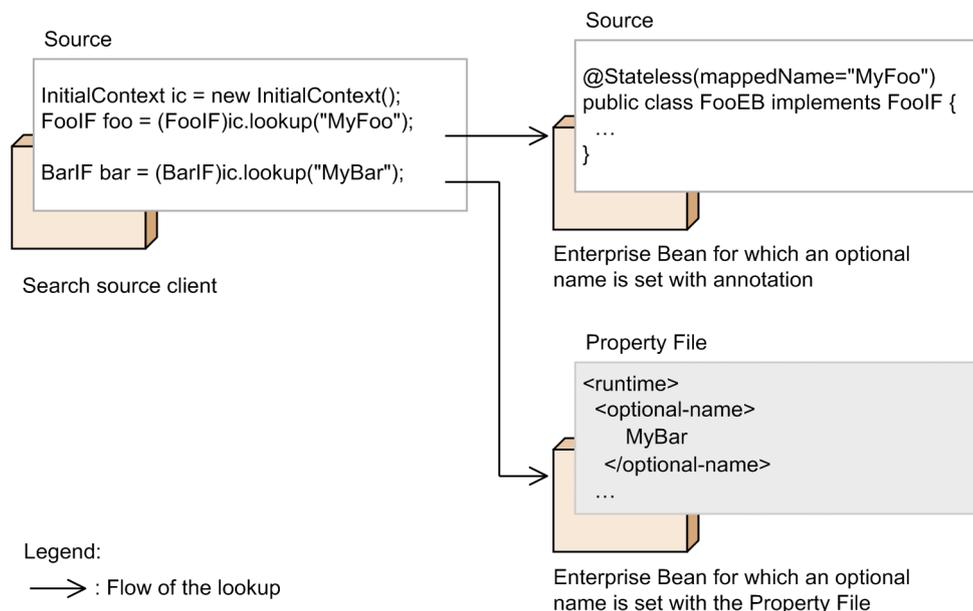
(1) Relation between the settings for the client source and the search destination object

This subsection describes the specification method in the client source and the settings in the search destination object when an Enterprise Bean or J2EE resource that is assigned an optional name is searched.

In the client source, you specify the optional name of the search destination object as the name to be looked up. In the search destination object, you use the annotation or property file to set up the corresponding optional name.

The following figure shows the relation between the coding of the source at the search source and the settings in the search destination object.

Figure 2-8: Relation between the settings for the client source and the search destination object



(2) Searching the Enterprise Beans

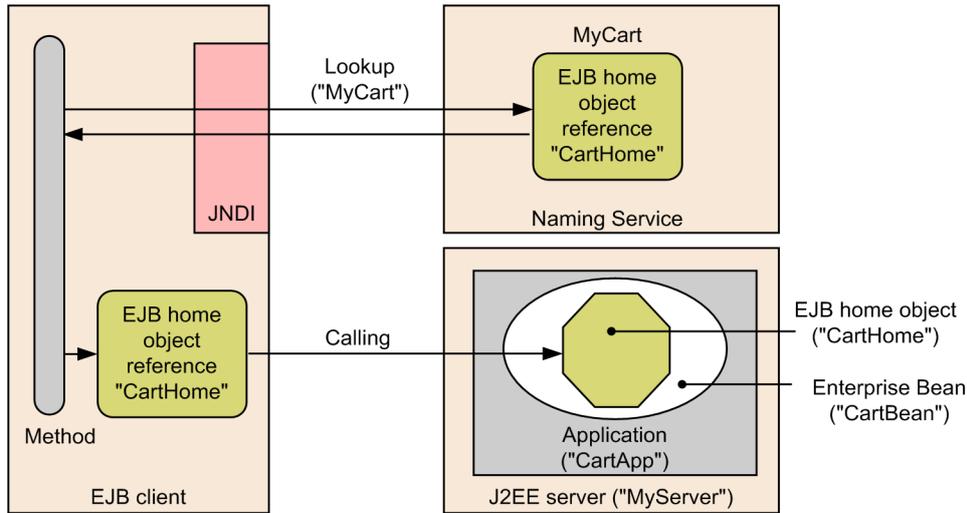
The following is an example of coding in the client when an optional name is used to search an Enterprise Bean. In this example, an EJB home object specifying an optional name "MyCart" is searched.

```
...
javax.naming.Context ctx = new javax.naming.InitialContext();
Object obj = ctx.lookup("MyCart");
SampleHome home =
(SampleHome)javax.rmi.PortableRemoteObject.narrow(obj, SampleHome.class);
Sample mybean = home.create(); // Generating a remote object
String name = mybean.ping(); //Executing the business method
...
```

The operations after acquiring the EJB home objects are similar to the operations for the EJB home objects acquired without using the optional names.

The following figure shows the procedure of lookup and acquisition of objects when you look up the EJB home object references.

Figure 2-9: Procedure of lookup and acquisition of objects for the EJB home object references



(3) Searching the J2EE resources

The following is an example of coding in the client when an optional name is used to search a J2EE resource. In this example, a J2EE resource specifying an optional name "jdbc/DBOpt" is searched.

```
Context initCtx = new InitialContext();
DataSource ds = (DataSource) initCtx.lookup("jdbc/DBOpt");
```

Note that the same operations are executed after acquiring the J2EE resource objects that are executed for the objects acquired without using the optional names.

! Important note

You cannot search the J2EE resources from the EJB client applications. If you perform such a search, the exception `javax.naming.NameNotFoundException` occurs.

2.6.5 Setting the optional names for the Enterprise Beans

This section describes how to set up an optional name for Enterprise Beans.

You can set up an optional name for the Enterprise Beans using the following methods:

- Method of specifying the settings in `cosminexus.xml`
- Method of specifying the settings in annotations

This subsection describes the respective methods.

Note that when you use this functionality, you must first specify whether you want to use the optional name, in `usrconf.properties` of the server management commands. For details on the settings, see [2.6.7 Execution environment settings](#).

(1) Method of specifying the settings in `cosminexus.xml`

To set up the optional name for an Enterprise Bean, you specify the settings in the `<ejb-jar>` tag of `cosminexus.xml`. The following table describes the optional name settings for the Enterprise Beans in `cosminexus.xml`.

Table 2-25: Setting the optional names for the Enterprise Beans in cosminexus.xml

Items	Specified tags	Settings
Remote interface	<session> - <optional-name> tag	Specifies the optional name for the remote interface of the Session Bean.
	<entity> - <optional-name> tag	Specifies the optional name for the remote interface of the Entity Bean.
Local interface	<session> - <local-optional-name> tag	Specifies the optional name for the local interface of the Session Bean.
	<entity> - <local-optional-name> tag	Specifies the optional name for the local interface of the Entity Bean.

For details on the specified tags, see 2.2 *Details of attributes specified in the Cosminexus application property file (cosminexus.xml)* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

Reference note

You can also specify the optional name for the Enterprise Beans using annotations. If the optional name is specified using the annotations and if a different optional name is set up using the server management commands, the value set up with the server management commands becomes valid.

For details, see (3) *When an optional name is specified in both server management commands and annotations*.

(2) Method of specifying the settings in annotations

You specify the optional name in the `mappedName` attribute of `@Stateless`, `@Stateful`, or `@Singleton`.

The following is an example of coding when the optional name is specified using annotations. In this example, the optional name "MyFoo" of the Stateless Session Bean is specified in the `mappedName` attribute of `@Stateless`.

```
@Stateless(mappedName="MyFoo")
public class FooEB implements FooIF {
    ...
}

public interface FooIF {
    ...
}
```

The optional name specified in the `mappedName` attribute of `@Stateless`, `@Stateful`, or `@Singleton` is set up in the `<hitachi-session-bean-property><mapped-name>` tag of the Session Bean property file.

(3) When an optional name is specified in both server management commands and annotations

If the optional name is specified by specifying the `mappedName` attribute of `@Stateless`, `@Stateful`, or `@Singleton`, and if the optional name is specified in the `<optional-name>` tag or `<local-optional-name>` tag in `cosminexus.xml`, the specification in the `<optional-name>` tag and `<local-optional-name>` tag becomes valid.

The following table lists the tags that become valid when the optional name is specified in both server management commands and annotations.

Table 2-26: Tags that become valid when the optional name is specified in both server management commands and annotations

Target for setting up the optional name	Only the mappedName attribute of @Stateless, @Stateful, or @Singleton is specified	Only the <optional-name> tag or <local-optional-name> tag is specified	When both mappedName attribute of @Stateless, @Stateful, or @Singleton, and the <optional-name> tag or <local-optional-name> tag are specified
Remote interface	<mapped-name> tag #	<optional-name> tag	<optional-name> tag
Local interface	<mapped-name> tag #	<local-optional-name> tag	<local-optional-name> tag

#: The tag in which the value specified in the mappedName attribute of @Stateless, @Stateful, or @Singleton is set up.

2.6.6 Setting the optional names for the J2EE resources

This subsection describes how to set an optional name for a J2EE resource.

You can set the optional name for a J2EE resource in `cosminexus.xml`.

Note that when you use this functionality, you must first specify whether you want to use the optional name, in `usrconf.properties` of the server management commands. For details about the settings, see *2.6.7 Execution environment settings*.

To set up the optional names for the J2EE resources, specify the settings in the `<rar>` tag of `cosminexus.xml`. The following table describes the optional name settings for the J2EE resources in `cosminexus.xml`.

Table 2-27: Setting the optional names for the J2EE resources in `cosminexus.xml`

Items	Specified tags	Settings
Resource adapter	<resource-external-property> - <optional-name> tag	Specifies the optional name for the resource adapter.

Note: You specify the optional name for the mail configuration and JavaBeans resources using the property files. For details on the settings, see *2.6.7 Execution environment settings*.

For details about the specified tags, see *2.2 Details of attributes specified in the Cosminexus application property file (cosminexus.xml)* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

2.6.7 Execution environment settings

To use the user-specified name space functionality, you must customize the server management commands and set up the J2EE applications.

(1) Customizing the server management commands

You can customize the settings for operating the server management commands. This subsection describes the settings for the user-specified name space functionality used by the server management commands.

You customize the server management commands using `usrconf.properties` (system property file for the server management commands). The following table describes the settings.

Table 2-28: Customizing the server management commands for using the user-specified name space functionality

Items	Specified keys	Settings
Assigning the optional names to the EJB home	<code>ejbserver.cui.optionalname.enabled</code>	Specifies whether to set up the optional names for the EJB home objects.

Items	Specified keys	Settings
object references (User-specified name space functionality)	<code>ejbserver.cui.optionalname.enabled</code>	Also, for assigning an optional name to an EJB home object, in addition to specifying this key, you must also specify the optional name [#] for the EJB home object.
Assigning the optional names to the J2EE resources (User-specified name space functionality)	<code>ejbserver.cui.optionalname.enabled</code>	Specifies whether to set up the optional names for the J2EE resources. Also, for assigning an optional name to a J2EE resource, in addition to specifying this key, you must also specify the optional name [#] for the J2EE resource.

[#]: You specify an optional name, when you define the J2EE application properties by using the server management commands. For details about referencing and changing the name registered in the JNDI name space, see *9.13 Referencing and changing the names registered in the JNDI name space in the uCosminexus Application Server Application Setup Guide*.

(2) Setting up the J2EE applications

You implement the J2EE application settings in the execution environment using the server management commands and property files. This subsection describes the settings for specifying the optional name for the Enterprise Beans and J2EE resources separately. Note that the tags described here correspond to `cosminexus.xml`. For details about the definitions in `cosminexus.xml`, see *2.6.5 Setting the optional names for the Enterprise Beans* or *2.6.6 Setting the optional names for the J2EE resources*.

(a) Setting the optional names for the Enterprise Beans

To set up an optional name for an Enterprise Bean, use the Session Bean property file or Entity Bean property file. The tag names differ depending on the interface type. The property files and tags used for each interface type are as follows:

- Optional name for the remote interface
You specify the optional name in the Session Bean property file for the Session Bean and in the `<optional-name>` tag of the Entity Bean property file for the Entity Bean.
- Optional name for the local interface
You specify the optional name in the Session Bean property file for the Session Bean and in the `<local-optional-name>` tag of the Entity Bean property file for the Entity Bean.

An example of specification of an optional name in a property file is as follows. In this example, the `SessionBean` property file is used to set up an optional name for the Stateful Session Beans.

```
<hitachi-session-bean-property>
<display-name>MyAdder</display-name>
<session-type>Stateful</session-type>
<transaction-type>Container</transaction-type>
<runtime>
<lookup-name>MyAdder</lookup-name>
<optional-name>user/Adder</optional-name>
<local-optional-name>user/localAdder</local-optional-name>
<maximum-sessions>0</maximum-sessions>
<stateful>
<maximum-active-sessions>0</maximum-active-sessions>
<inactivity-timeout>0</inactivity-timeout>
<removal-timeout>0</removal-timeout>
```

In this example, `"user/Adder"` is set as the optional name for the remote interface and `"user/localAdder"` is set up as the optional name for the local interface.

(b) Setting the optional names for the J2EE resources

To set up an optional name for a J2EE resource, specify the name as an attribute of the DB Connector, mail configuration, or JavaBeans resource. The commands and property files used in the settings differ for each resource type. The following table lists the resource types and the commands and property files for setting the optional names.

Table 2-29: Resource types and the commands and property files for setting the optional names

Types of resource adapters	Commands	Property file	Specified tags
Resource adapter	<code>cjsetrarprop</code> command	Connector property file	<code><resource-external-property></code> - <code><optional-name></code> tag
Mail configuration	<code>cjsetresprop</code> command	Mail property file	
JavaBeans resource	<code>cjsetjbprop</code> command	JavaBeans resource property file	<code><resource-env-external-property></code> - <code><optional-name></code> tag

Note: You cannot set up the optional names for the mail configuration and JavaBeans resources in `cosminexus.xml`. You specify the optional name in a property file.

The following is an example of specification of an optional name in a property file. In this example, the Connector property file is used to set up the optional name for the resource adapters.

```
<hitachi-connector-property>
<description></description>
<display-name>DB_Connector_for_Oracle</display-name>
<icon />
<vendor-name>Hitachi, Ltd.</vendor-name>
:
<connector-runtime>
<resource-external-property>
<description></description>
<optional-name>jdbc/TestDB1</optional-name>
<res-auth>Container</res-auth>
<res-sharing-scope>Shareable</res-sharing-scope>
</resource-external-property>
</connector-runtime>
</hitachi-connector-property>
```

In this example, "jdbc/TestDB1" is set as the optional name for DB Connector.

2.6.8 Precautions for using the user-specified name space functionality

This subsection describes the precautions for using the user-specified name space functionality.

(1) Notes on executing a search using an optional name

- An Enterprise Bean with a local interface cannot be searched from outside the J2EE application.
- The optional name of the Enterprise Bean with a local interface cannot be searched for the naming contexts.
- You can set up the same optional name for the Enterprise Bean with a remote interface and the Enterprise Bean with a local interface. However, if you look up the optional name that is duplicated in such a case, the Enterprise Bean with a local interface will necessarily be looked up.

(2) Notes on the naming service

- When one CORBA Naming Service is shared by multiple J2EE servers, you cannot use the user-specified name space functionality.
- Specify settings so that the J2EE server and the CORBA Naming Service start and stop at the same time. If either the J2EE server or the CORBA Naming Service is down, restart the J2EE server and the CORBA Naming Service together.
- If the CORBA Naming Service is shared, you cannot use 'Cosminexus' as the optional name to be set up in the user-specified name space functionality.

(3) Notes on specifying the optional names for the J2EE resources

- When you stop, delete, or change the attributes (for a JavaMail session) of the J2EE resource for which the optional name is registered, first stop all the J2EE applications running on the J2EE server.
- When using the user-specified name space functionality of a J2EE resource, specify the same string as the host name of the provider URL (`java.naming.provider.url`), specified in `InitialContext` generated by the J2EE application and the value of the `ejbserver.naming.host` key specified in the J2EE server definition. Note that if this condition holds true, you need not specify the provider URL for `InitialContext` that is generated by the J2EE application.
 - Specify `localhost` (default value) in the `ejbserver.naming.host` key of the user property file for J2EE servers.
 - Connect to the naming service on the same J2EE server.

If `localhost` is specified in the `ejbserver.naming.host` key, and if you want to specify the provider URL for `InitialContext` generated by the J2EE application, specify the value that can be acquired using the following API in the host name of the provider URL:

```
java.net.InetAddress.getLocalHost().getHostName();
```

- Even if the optional name in the user-specified name space functionality of a J2EE resource contains a forward slash (/), you cannot perform a naming context search. Only the specified optional name can be used as the lookup name.

The examples of JNDI lookup names that can and cannot be used for the J2EE resources (DB Connector) that are deployed and started with "jdbc/TestDB" assigned as an optional name are as follows:

Example of a lookup name that can be used

```
DataSource ds = (DataSource) initCtx.lookup("jdbc/TestDB");
```

Example of a lookup name that cannot be used

```
Context ctx = (Context) initCtx.lookup("jdbc");
```

2.7 Searching the CORBA Naming Service by using the round-robin policy

You can look up the EJB home objects or business interface references with the same names (optional name) registered in multiple CORBA Naming Services by following the round-robin policy. This is called the *round-robin search*.

When you perform a round-robin search for the EJB home objects or business interface references using the relevant names from the JNDI name space, the client application can acquire the EJB home objects or business interface references selected by the round-robin policy from the EJB home objects or business interface references that exist in multiple CORBA Naming Services. As a result, you can distribute the load out by starting the J2EE server in a cluster configuration. Furthermore, you can invoke the Enterprise Beans of the J2EE server from the EJB client regardless of the cluster configuration.

The following table describes the organization of this section.

Table 2-30: Organization of this section (Searching the CORBA Naming Service by using the round-robin policy)

Category	Title	Reference location
Explanation	Scope of round-robin search	2.7.1
	Operations when performing a round-robin search	2.7.2
Settings	Settings required for performing a round-robin search	2.7.3
	Settings recommended for using the round-robin search functionality	2.7.4
Notes	Notes on performing a round-robin search	2.7.5

Note: The function-specific explanation is not available for "Implementation" and "Operations".

Tip

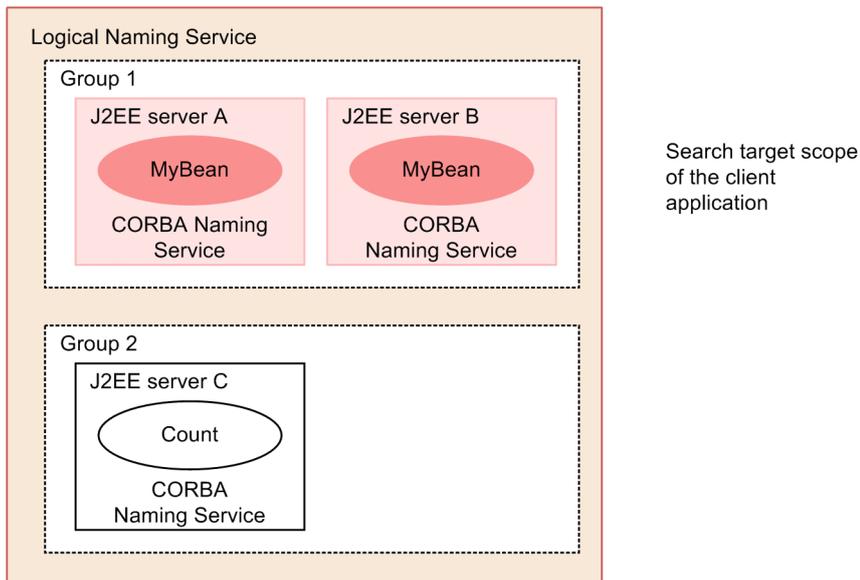
Note that to perform a round-robin search, the EJB home objects or business interface references to be searched must be registered with the same names in the CORBA Naming Service. Therefore, use the user-specified name space functionality to specify the same names for the EJB home objects or business interface references.

2.7.1 Scope of round-robin search

The scope of round-robin search using the JNDI includes groups within a *logical naming service*.

The logical naming service is made up of one or more groups. Each group contains one or more CORBA Naming Services. The following figure shows the configuration of a logical naming service.

Figure 2-10: Configuration of a logical naming service



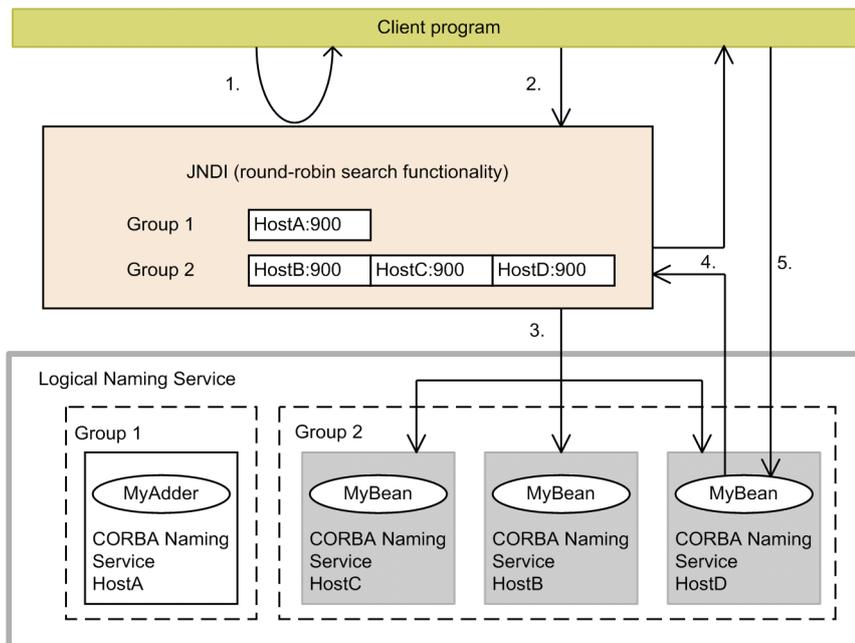
You can specify the same CORBA Naming Service for the group members in multiple groups.

2.7.2 Operations when performing a round-robin search

The following figure shows the operations in a round-robin search when performed from a client program.

You assume that the EJB home object to be searched using the round-robin search is registered with the name "MyBean" in each of the CORBA Naming Services shown in the figure. The same name is set up using the user-specified name space functionality in the multiple EJB home objects to be searched.

Figure 2-11: Operations during the round-robin search



The following is a description of the figure:

1. The client application uses the group name of the logical naming services to be searched, as an argument, and generates the `InitialContext` instance for round robin.
2. The client application specifies the name "My Bean" and requests a search to the generated `InitialContext` instance for round robin.
3. The round-robin Search functionality uses the round-robin policy to determine the search destination of the CORBA Naming Services belonging to the group to be searched.
4. If the search is successful, the result is returned to the client application.
Note that if the search fails, another CORBA Naming Service belonging to that group is searched.
5. The client application accesses the searched EJB home objects.

! Important note

When `InitialContext` is issued for the first time, an attempt is made to connect to the provider URLs of all the groups. Therefore, if a non-connectable provider URL is mentioned, the processing might be delayed.

2.7.3 Settings required for performing a round-robin search

This subsection describes the settings required for performing a round-robin search and the naming rules of the group names.

The following settings are required when performing a round-robin search:

- Logical naming service group for which the round-robin search will be performed
- Root position of the CORBA Naming Services that belong to each group
- Class to which the implementation of `InitialContextFactory` is delegated

To use the round-robin search functionality, you specify the settings in the system properties. Note that in addition to the system property settings, you can also specify the classes to which the implementation of `InitialContextFactory` is delegated, as arguments when generating `InitialContext` of each application. In the arguments used for generating `InitialContext`, you can select and specify a specific naming service from the group of logical naming services specified in the system properties.

Note that when the settings are only specified in the system properties, you cannot perform a round-robin search specifying a specific group. The naming services of all the groups existing on the logical naming service are included in the scope of search.

The following is an overview of each setting method:

(1) Identifying the groups from the system property settings and the root position of the CORBA Naming Service belonging to that group

When executing the round-robin search, you specify the group of logical naming services to be searched by a round-robin search and the root position of the naming services belonging to the group, in the system properties. Also, you must specify

```
java.naming.factory.initial=com.hitachi.software.ejb.jndi.GroupContextFactory
```

as the class to which the implementation of `InitialContextFactory` is delegated.

The method of setting up the system properties differs at the following locations for the various types of applications that use the round-robin search functionality.

(2) For the J2EE applications (Enterprise Beans or servlets) running on the J2EE server

You customize and set up the J2EE server properties. Specify the settings in the user properties for J2EE servers in the Easy Setup definition file.

You specify the definition for the round-robin search in the `<configuration>` tag of the logical J2EE server (`j2ee-server`) in the Easy Setup definition file. The following table describes the definitions for performing a round-robin search in the Easy Setup definition file.

Table 2-31: Definition for performing a round-robin search in the Easy Setup definition file

Specified parameters	Settings
<code>ejbserver.jndi.namingservice.group.list</code>	Specify the CORBA Naming Service group.
<code>ejbserver.jndi.namingservice.group.Specify-group-name.providerurls</code>	Specify the root position of the CORBA Naming Services belonging to each group.
<code>java.naming.factory.initial</code>	Specify the class to which the implementation of <code>InitialContextFactory</code> is delegated.

Note that the round-robin search presumes the use of the user-specified name space functionality. When you use the user-specified name space functionality, you need to customize the settings for operating the server management commands and define the J2EE application properties. For details on the settings, see *2.6.7 Execution environment settings* and *2.6.5 Setting the optional names for the Enterprise Beans*.

(3) For the EJB client applications running on servers other than the J2EE server

You use one of the following methods to specify the settings:

- Specify the settings as properties when starting the EJB client application.
- Specify the settings in the application by using the `System.setProperty` method.

Note that the method of setting the EJB client application properties differs depending on the commands used for starting the EJB client application. This subsection describes how to set the EJB client application properties and the examples of specification.

(a) Methods of setting the EJB client application properties

The method of setting the properties differs when the `cjclstartap` command is used and when the `vbj` command is used.

- **For the `cjclstartap` command**

When the `cjclstartap` command is used, you set the properties in the EJB client application properties file (`usrconf.properties`). For properties that can be specified, see *(2) For the J2EE applications (Enterprise Beans or servlets) running on the J2EE server*.

- **For the `vbj` command**

When the `vbj` command is used, you set the properties in the batch file/ shell script file, or in the command arguments.

(b) Example of specifying the properties

An example of specification is as follows. This is an example of specifying the properties in `usrconf.properties`. For details on each key, see *14.3 usrconf.properties (User property file for Java applications)* in the *uCosminexus Application Server Definition Reference Guide*.

```
# Define the logical naming service configuration
ejbserver.jndi.namingservice.group.list=g1;g2;g3
ejbserver.jndi.namingservice.group.g1.providerurls=corbaname::hostA:900;corbaname::hostB:900
ejbserver.jndi.namingservice.group.g2.providerurls=corbaname::hostD:700;corbaname::hostE:700
ejbserver.jndi.namingservice.group.g3.providerurls=corbaname::hostF:800;corbaname::hostG:800
# Specify the class to which the implementation of InitialContextFactory is delegated
java.naming.factory.initial=com.hitachi.software.ejb.jndi.GroupContextFactory
:
```

The following contents are specified in the `ejbserver.jndi.namingservice.group.list` key, `ejbserver.jndi.namingservice.group.Specify-group-name.providerurls` key, and `java.naming.factory.initial` key respectively in the example of specification:

`ejbserver.jndi.namingservice.group.list` key

When performing a round-robin search, define the group of the logical naming services that is to be searched. Specify each group name so that it can be uniquely identified in logical naming.

2. Naming Management

`ejbserver.jndi.namingservice.group.Specify-group-name.providerurls` key

You specify the root position of the naming service belonging to each group in the provider URL. In *Specify-group-name*, you specify the group name specified in `ejbserver.jndi.namingservice.group.list`.

`java.naming.factory.initial` key

You specify the class to which the implementation of `InitialContextFactory` is delegated.

If you specify `com.hitachi.software.ejb.jndi.GroupContextFactory` in the `java.naming.factory.initial` key, a round-robin search is implemented. If you do not specify `com.hitachi.software.ejb.jndi.GroupContextFactory` in the `java.naming.factory.initial` key, the CORBA Naming Service used by the J2EE server is searched.

Note that if you specify the `java.naming.factory.initial` key as an argument when generating `InitialContext` in the method described in (4) *Selecting the group to be searched based on the argument specified for generating InitialContext*, you need not specify a value for this key in the system properties.

(4) Selecting the group to be searched based on the argument specified for generating InitialContext

When settings are specified for executing a round-robin search, you can select a specific group to be looked up in the round-robin search targets by specifying the group as an argument for generating `InitialContext` in the client application. Note that the specification of an argument for generating `InitialContext` is optional.

An example of specification is as follows:

```
:
Hashtable env = new Hashtable();
env.put("ejbserver.jndi.namingservice.groupname", "g1");
env.put("java.naming.factory.initial",
"com.hitachi.software.ejb.jndi.GroupContextFactory");
InitialContext ic = new InitialContext(env);
:
```

The following contents are specified in the `ejbserver.jndi.namingservice.groupname` key and the `java.naming.factory.initial` key respectively in the example of specification:

`ejbserver.jndi.namingservice.groupname` key

You specify the group name to be searched in the 'g/' portion. Specify the group name that is already defined in system properties (`ejbserver.jndi.namingservice.group.list` key of `usrconf.properties`). Note that `ejbserver.jndi.namingservice.groupname` key does not have a default value. When you do not specify a group name, all the groups set in the system properties are searched.

`java.naming.factory.initial` key

You specify the class to which the implementation of `InitialContextFactory` is delegated. If you specify `com.hitachi.software.ejb.jndi.GroupContextFactory` in the `java.naming.factory.initial` key, a round-robin search is implemented. When you omit the specification in the `java.naming.factory.initial` key of the system properties and if you do not specify this key in the argument, the group specified in the `ejbserver.jndi.namingservice.groupname` key is not searched and the CORBA Naming Service used by the J2EE server is searched.

(5) Naming rules for the group names

You can use the following characters in a group name:

- Alphanumeric characters (A-Z, a-z, 0-9)
- Underscore (`_`)

Make sure to specify a group name that is unique in the logical naming service.

(6) Settings in the properties

The EJB client application properties are used when the Enterprise Beans are invoked from the Java application. Set the required properties according to the contents of the J2EE application. For details on the properties that can be set, see the *uCosminexus Application Server Definition Reference Guide*.

Example of contents that can be set in the properties

- **Log settings for the EJB client applications**

You use the keys beginning with `ejbserver.client.log` and the keys beginning with `ejbserver.logger` to change the output destination and the log level of the system log output by the system and the user log output by the EJB client applications. For details on the system log, see 3.8 *System log output by the EJB client applications* in the *uCosminexus Application Server EJB Container Functionality Guide*, and for details on the user log, see 9. *User log output by the applications* in the *uCosminexus Application Server Expansion Guide*.

- **Transactions settings for the EJB client applications**

You use the keys beginning with `ejbserver.client.transaction` to specify whether to use transactions with the EJB client applications and to specify the client name to be used by the transaction service. Note that when you use uCosminexus Client to set up the EJB client environment, you cannot use the EJB client application transactions. For details, see 3.20 *Notes on starting a transaction with the EJB client applications*.

- **EJB client operations in the case of communication error in the EJB remote interface**

In the `ejbserver.container.rebindpolicy` key, you can specify the re-connect operation for the connections and the re-send operation for the requests in the EJB client.

- **Settings for executing the round-robin search from the EJB client applications**

In the `ejbserver.jndi.namingservice.group.list` key, `ejbserver.jndi.namingservice.group.specify-group-name.providerurls` key, and `java.naming.factory.initial` key, you can specify the CORBA Naming Service group, the root position of the CORBA Naming Service belonging to each group, and the class to which the implementation of `InitialContextFactory` is delegated. Note that the round-robin search is enabled when you specify the user-specified name space functionality during the customization of the server management commands for the J2EE server.

- **Setting the priority of requests from the EJB client applications to the CTM**

In the `ejbserver.client.ctm.RequestPriority` key, you can set the priority for the requests to be sent from the EJB client applications to the CTM.

(7) Properties with different specification requirements depending on the commands (for the EJB client applications)

This subsection describes the properties with different specification requirements depending on the EJB client application commands (`vbj` command). The following table lists the property keys with different specification requirements depending on the EJB client application commands.

Table 2-32: Property keys with different specification requirements depending on the EJB client application commands

Property keys	Type	Commands	
		cjclstartap	vbj
<code>org.omg.CORBA.ORBClass</code>	Fixed	N	N
<code>org.omg.CORBA.ORBSingletonClass</code>	Fixed	N	N
<code>javax.rmi.CORBA.UtilClass</code>	Fixed	N	Y
<code>javax.rmi.CORBA.StubClass</code>	Fixed	N	N
<code>javax.rmi.CORBA.PortableRemoteObjectClass</code>	Fixed	N	Y
<code>java.endorsed.dirs</code>	Variable	N	N

Legend:

Fixed: Value for the applicable key is fixed and must be specified

Variable: Value must be specified according to the system execution environment

Y: Key must be specified for the command

N: Key need not be specified for the command

2.7.4 Settings recommended for using the round-robin search functionality

When you use the round-robin search functionality, we recommend that you also use the naming service error detection functionality.

For the example of settings when the round-robin search functionality is combined with the naming service error detection functionality, see *2.9.4 Execution environment settings (When the error detection functionality is used)*.

2.7.5 Notes on performing a round-robin search

This subsection describes the notes on performing a round-robin search.

- The context acquired for a round-robin search only supports the `lookup` method. You cannot use the other APIs defined in `javax.naming.Context`.
- If you execute a round-robin search using an optional name, the Enterprise Beans are searched in the following order:
 1. An Enterprise Bean with a local interface is searched from the name space of the J2EE server executing the search.
 2. If the Enterprise Bean is not found in 1, an Enterprise Bean with a remote interface is searched by the round-robin search.

The following table describes whether the Enterprise Beans can be searched for each class to which the processing of `InitialContextFactory` is delegated.

Table 2-33: Searchability of the Enterprise Beans for each class to which the processing of `InitialContextFactory` is delegated

Search target	GroupContextFactory	InsContextFactory
Enterprise Bean with a remote interface	Y [#]	Y
Enterprise Bean with a remote interface (search by optional name)	Y	Y
Enterprise Bean with a local interface	Y [#]	Y
Enterprise Bean with a local interface (search by optional name)	Y	Y
J2EE resource	Y [#]	Y
J2EE resource (search by optional name)	N	Y

Legend:

Y: Can be searched

N: Cannot be searched

[#] When you perform a lookup in `java:comp/env`, the round-robin search is not executed. `java:comp/env` is searched only from the J2EE server executing the lookup.

You use the following methods to specify the class to which the implementation of `InitialContextFactory` will be delegated. If both the methods are specified, the method specified in the argument is enabled.

- Specify the class in the `java.naming.factory.initial` key of `usrconf.properties`.
- Specify the class in the `java.naming.factory.initial` key with the argument (`Hashtable`) used when `InitialContext` is generated.

2.8 Caching with the naming management functionality

The naming management functionality of the J2EE services has a caching functionality. The caching functionality is a functionality whereby when you search an EJB home object reference through the JNDI, the relevant object is temporarily stored in the cache and returned in the subsequent searches for the same object.

This section describes the procedure of caching and the clearing of the cache area.

You specify the settings for caching with the naming management functionality as a property of the J2EE server or EJB client application.

The following table describes the organization of this section.

Table 2-34: Organization of this section (Caching with the naming management functionality)

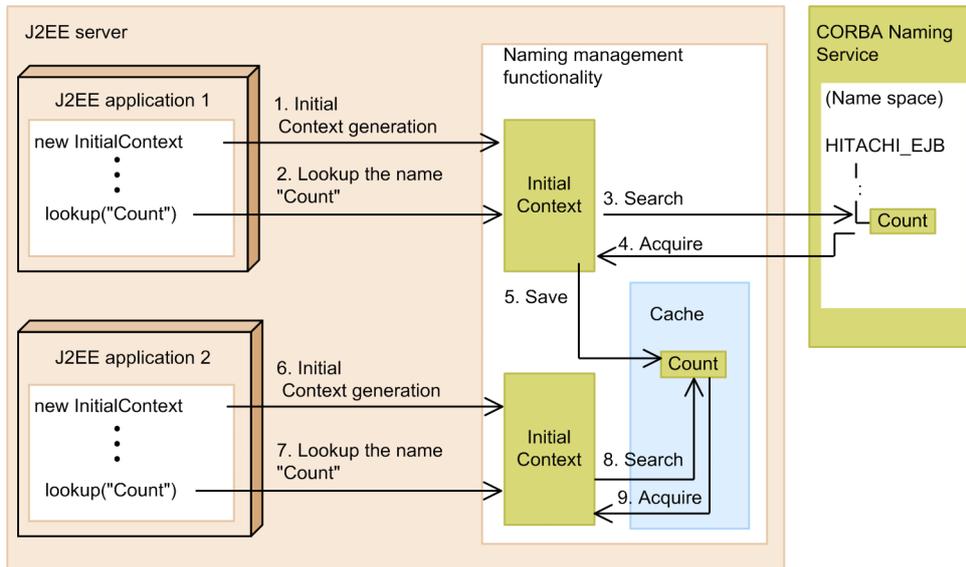
Category	Title	Reference location
Explanation	Procedure of caching	2.8.1
	Clearing the cache used in naming	2.8.2
Settings	Settings for using the caching functionality	2.8.3
Notes	Notes on caching in naming	2.8.4

Note: The function-specific explanation is not available for "Implementation" and "Operations".

2.8.1 Procedure of caching

The following figure shows the procedure by which the naming is cached.

Figure 2-12: Procedure of caching in naming



The following is a description of the caching procedure. In this procedure, the EJB home object references with the same name 'Count' are looked up from two J2EE applications on the same J2EE server. The points 1 through 5 indicate processing executed from the J2EE application 1 and points 6 through 9 indicate processing executed from the J2EE application 2.

1. J2EE application 1 generates an instance of the JNDI `javax.naming.InitialContext` class.
2. J2EE application 1 requests a search (lookup) of the EJB home object references for the instance of the `javax.naming.InitialContext` class. At this time, 'Count' is specified in the name.

3. The naming management functionality that receives the request searches the EJB home object references from the name space of the CORBA Naming Service.
4. The naming management functionality obtains the EJB home object references as the search results.
5. The naming management functionality stores the obtained EJB home object references in the cache.
6. J2EE application 2 existing on the same process generates the instance of the JNDI `javax.naming.InitialContext` class.
7. J2EE application 2 requests a search (lookup) of the EJB home object references for the instance of the `javax.naming.InitialContext` class. At this time, the same name 'Count' as that in 2 is specified as the name.
8. The naming management functionality that receives the request searches the EJB home object references from the cache.
9. The naming management functionality obtains the EJB home object references from cache as the search results.

2.8.2 Clearing the Cache Used in Naming

You can clear the cache used in naming. However, you cannot specify the size of the cache to be cleared. This subsection describes when the cache is cleared and the range that is cleared.

(1) Timing for clearing the cache

The contents of the cache are cleared at one of the following timings:

- If an exception occurs in the JNDI and RMI-IIOP APIs, the cache is forcefully cleared.
- The cache is cleared at the interval specified in the system properties (by default, the value is 0 seconds and the cache is not cleared).

(2) Cache-clearing range

This subsection describes the range in which the cache is cleared in a naming service.

The cache-clearing range is as follows:

1. Clearing the complete cache area.
2. Clearing only the invalid cache area.

In the first case, the complete cache area is cleared. On the other hand, in the second case, the objects stored in the cache are periodically checked for validity and only the invalid objects are cleared from the cache. Furthermore, in the second case, when the cache is cleared, the status of the CORBA Naming Services that have been searched once is also monitored. As a result, a CORBA Naming Service that has been searched once is not included in the search if the CORBA Naming Service is down. If you restart the CORBA Naming Service, the search of that CORBA Naming Service starts automatically.

For details on how to specify the settings for clearing the cache, see *2.8.3 Settings for using the caching functionality*.

Also, if you want to use the error detection functionality, see *2.9.4 Execution environment settings (When the Error Detection functionality is used)*.

Reference note

When the re-connect functionality of the EJB home object is used, the cache does not become invalid even after the J2EE server is restarted.

Therefore, even if you specify settings to clear only the invalid cache area, the object references of the EJB home objects for the CORBA Naming Service are not deleted from the cache area.

The object references that are not deleted from the cache can be used as are to search (lookup) the EJB home object references.

2.8.3 Settings for using the caching functionality

This subsection describes the settings for using the caching functionality. The locations for the settings differ in the J2EE applications and in the EJB client applications.

(1) When the caching functionality is used in the J2EE applications

You specify the definition for using the caching functionality in the J2EE applications in the `<configuration>` tag of the logical J2EE server (`j2ee-server`) in the Easy Setup definition file. The following table describes the definitions for using the caching functionality specified in the Easy Setup definition file.

Table 2-35: Definitions for using the caching functionality

Specified parameters	Settings
<code>ejbserver.jndi.cache</code>	Specify whether to enable caching in naming.
<code>ejbserver.jndi.cache.interval</code>	Specify the interval (unit: seconds) for clearing the cache when caching is performed in naming.
<code>ejbserver.jndi.cache.interval.clear.option</code>	Determine the operations (cache-clearing range) for the cache area in naming after the lapse of the interval.

The following is an example of settings when the cache is cleared periodically (when the physical tier is defined).

Example:

```
<configuration>
  <logical-server-type>j2ee-server</logical-server-type>
  <param>
    <param-name>ejbserver.jndi.cache</param-name>
    <param-value>on</param-value>
  </param>
  <param>
    <param-name>ejbserver.jndi.cache.interval</param-name>
    <param-value>60</param-value>
  </param>
  <param>
    <param-name>ejbserver.jndi.cache.interval.clear.option</param-name>
    <param-value>check</param-value>
  </param>
  ...
</configuration>
```

(2) When the caching functionality is used in the EJB client applications

You specify the settings as properties when the EJB client application starts.

The method of setting the EJB client application properties differs depending on the commands used for starting the EJB client application.

- **For the `cjclstartap` command**

When the `cjclstartap` command is used, you set up the properties in the properties file for the EJB client applications (`usrconf.properties`).

- **For the `vbj` command**

When the `vbj` command is used, you set up the properties in the batch file/ shell script file, or the command arguments.

The following is an example of settings when the cache is cleared periodically.

Example of system property settings for an EJB client

```
...
# Cache settings
ejbserver.jndi.cache=on
ejbserver.jndi.cache.interval=60
ejbserver.jndi.cache.interval.clear.option=check
```

2.8.4 Notes on caching in naming

This sub-section describes the notes related to caching of naming.

- We recommend that you disable the caching in naming when the EJB home object references and the JDBC data source are cached in the application.
- To clear the cache periodically, specify the settings in the Easy Setup definition file. Specify the following parameters in the `<configuration>` tag of the logical J2EE server (`j2ee-server`):
 - `ejbserver.jndi.cache.interval.clear.option`
Specifies the range for clearing the cache.
 - `ejbserver.jndi.cache`
Specifies whether to execute the cache. Specify 'ON'.
 - `ejbserver.jndi.cache.interval`
Specifies the cache-clearing interval.
- If `check` is set in the property `ejbserver.jndi.cache.interval.clear.option` in the `<configuration>` tag of the logical J2EE server (`j2ee-server`), the CORBA Naming Service is monitored only at the cache-clearing time specified in the property `ejbserver.jndi.cache.interval`. After the CORBA Naming Service restarts, the value specified in the property `ejbserver.jndi.cache.interval` is the maximum time required for detecting the recovery of the CORBA Naming Service.
- When you use the caching functionality, if the J2EE server that holds the EJB home objects is down or if the J2EE application is redeployed when the cache contains stored EJB home object references, the object references of the EJB home object stored in the cache become invalid. In this condition, if a search request (lookup) for an EJB home object is received, the invalid object references in the cache are returned to the requester. If methods such as the `javax.rmi.PortableRemoteObject.narrow()` or `create` methods are executed for such object references, a CORBA exception (such as `org.omg.CORBA.OBJECT_NOT_EXIST`) might occur. Note that when a CORBA exception occurs, all the cache information is deleted. In the next search request (lookup), valid information is obtained by connecting to the CORBA Naming Service.
- When the CTM is used and if only the invalid cache area is cleared at the specified interval, the object references of the EJB home object in the global CORBA Naming Service are not cleared from the cache area even if the J2EE server and J2EE application are stopped. If a search request (lookup) for the EJB home objects is received, the un-cleared object references on the cache return to the requester. If the J2EE application is restarted, the cached object references can be used as are. If the J2EE application is not restarted and if you execute a method such as the `create` method for the returned object references, a CORBA exception (`org.omg.CORBA.NO_IMPLEMENT`) occurs.
Note that if a CORBA exception occurs, all the cache information is deleted. If the J2EE application is restarted, valid information is obtained by connecting to the global CORBA Naming Service in the next search request (lookup).
- If the J2EE server is restarted by setting the `ejbserver.jndi.cache` property to "on" while a business interface is being used, a `javax.ejb.EJBException` might occur when a business method is executed. Note that if a `javax.ejb.EJBException` occurs, valid information will be acquired by connecting to a CORBA naming service the next time a lookup request is made.

2.9 Detecting errors in a naming service

You use the error detection of naming services as an option of the caching functionality.

If you use the naming service error detection functionality, the EJB client can detect errors faster when an error occurs in the naming service.

This section provides an overview of the naming service error detection functionality and describes the functionality recommended for concurrent use with the naming service error detection functionality.

The following table describes the organization of this section.

Table 2-36: Organization of this section (Naming management error detection)

Category	Title	Reference location
Explanation	What is the naming management error detection functionality	2.9.1
	Concurrent use of the round-robin search functionality	2.9.2
	Behavior of the naming service error detection functionality	2.9.3
Settings	Execution environment settings (when the Error Detection functionality is used)	2.9.4
Notes	Notes on the naming service error detection functionality	2.9.5

Note: The function-specific explanation is not available for "Implementation" and "Operations".

2.9.1 What is the naming service error detection functionality

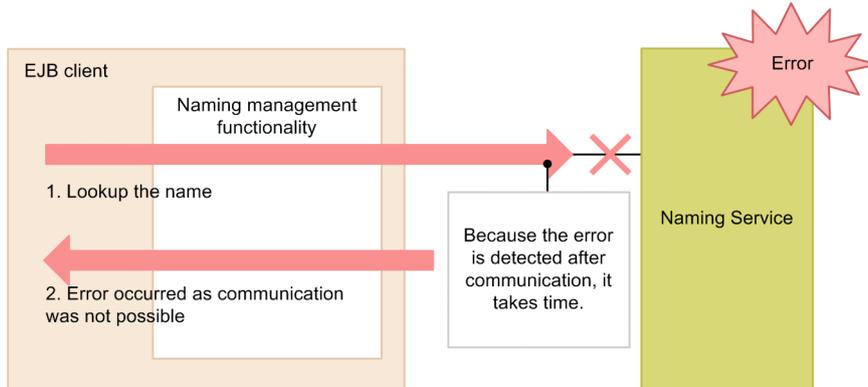
If you use the naming service error detection functionality, the J2EE server detects the error when the naming service stops or if machine error or network error occurs on Application Server.

In the Naming Service Error Detection functionality, the J2EE server monitors the status of the Naming Service functionality and can control the use of the Naming Service functionality that can no longer communicate. Therefore, the EJB client need not perform unnecessary communication.

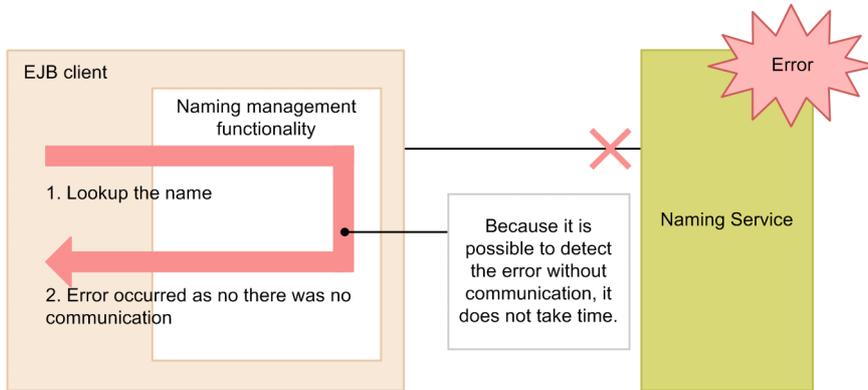
The following figure shows the flow of processing when you use the naming service error detection functionality.

Figure 2-13: Flow of processing when the naming service error detection functionality is used

●When the error detection functionality of Naming Service is not used



●When the error detection functionality of Naming Service is used



Legend:  : Flow of processing

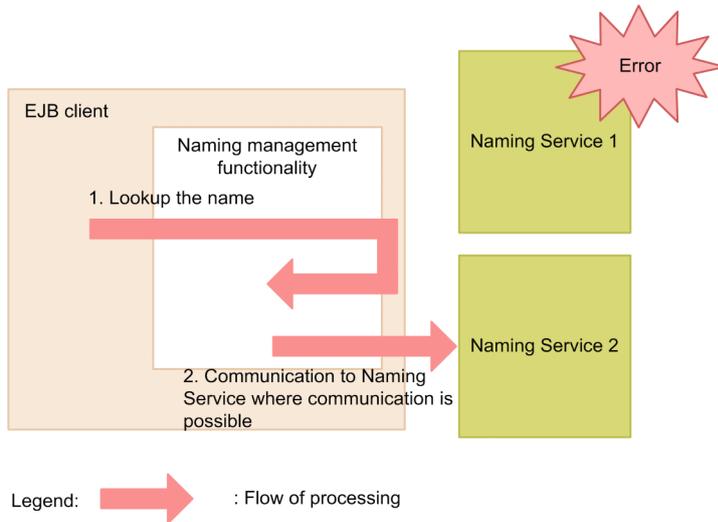
If the naming service error detection functionality is not used, the communication is necessarily performed with the naming service when a name is looked up. Therefore, a lot of time might be taken until an error occurs due to inability to communicate with the naming service. On the other hand, if you use the naming service error detection functionality, communication does not take place with the naming service where the error occurs and the error can be detected. Therefore, this does not take a long time.

2.9.2 Concurrent use of the round-robin search functionality

If you use the round-robin search functionality and the naming service error detection functionality concurrently, when an error occurs on one node, the failed node can be easily separated.

The following figure shows the communication when the round-robin search functionality and the naming service error detection functionality are used concurrently.

Figure 2-14: Communication when the round-robin search functionality and the naming service error detection functionality are used concurrently



If an error is detected in a specific naming service when a name is looked up, the round-robin search functionality switches the communication to a naming service with which communication is possible.

2.9.3 Behavior of the naming service error detection functionality

This section describes the behavior of the naming service error detection functionality.

(1) Lock timing

During naming service error detection, the status of the naming service is checked at the following times and if there is no response, the naming service is locked:

1. When there is no response until the lapse of the sweep interval
2. When there is no response from the naming service when the following operations are first executed after the entire cache area is cleared with the RMI/IIOP communication error as the trigger, and if the cache does not exist:
 - When `InitialContext` is generated
 - When lookup is executed

The following table describes the behavior of the operations for the naming service after the entire cache is cleared.

Table 2-37: Behavior of the operations for the naming service after the entire cache is cleared

Operations in the naming management functionality	When a round-robin search is used	When a round-robin search is not used
When <code>InitialContext</code> is generated	When <code>InitialContext</code> is generated for the first time after the EJB client process starts, if there is no response to the confirmation of the naming service operations, the naming service is locked. The naming service is not locked in cases other than the first time.	The viability is checked and if the response from the naming service cannot be detected, the naming service is locked.
When lookup is executed	Status is checked only for the naming service that attempts to communicate when lookup is issued. If a response cannot be detected, the naming service is locked. For example, if a round-robin search is being executed in	The viability is checked and if the response from the naming service cannot be detected, the naming service is locked.

Operations in the naming management functionality	When a round-robin search is used	When a round-robin search is not used
When lookup is executed	three naming services, the failed naming service is locked at the third lookup at the most.	The viability is checked and if the response from the naming service cannot be detected, the naming service is locked.

In the naming service error detection functionality, when the naming service is locked, the KDJE47111-I message is output to the log at the same time. After the message is output, all the communication with the running naming service functionality is controlled and `javax.naming.NamingException` is thrown.

(2) Behavior when the naming service is locked

This point describes the behavior of the naming service when the EJB client generates `InitialContext` or performs lookup for a naming service locked by the naming service error detection functionality.

The following table describes the operations for the locked naming service separately for the cases when the round-robin search functionality is used and when the functionality is not used.

Table 2-38: Operations for a locked naming service

Operations from the EJB client	When a round-robin search is used	When a round-robin search is not used
When <code>InitialContext</code> is generated	<code>InitialContext</code> for the round robin is returned.	Communication is always controlled. <code>javax.naming.NamingException</code> is returned to the EJB client.
When lookup is executed	The object is searched from another naming service registered in the round robin group and returned.	Communication is always controlled. <code>javax.naming.NamingException</code> is returned to the EJB client.

(3) Timing for unlocking the naming service

This point describes the unlock timing. The status of the naming service is checked at the following time and when a response can be detected from the naming service, the lock is released.

- When the sweep interval lapses

When the naming service is unlocked, the KDJE47110-I message is output to the log at the same time. After the message is output, the communication for the running naming service is not controlled.

2.9.4 Execution environment settings (When the Error Detection functionality is used)

The naming service error detection functionality is an option of the caching functionality. Therefore, the caching functionality is presumed to be set. For details on the settings for the caching functionality of the naming service, see *2.8.3 Settings for using the caching functionality*.

To use the naming service error detection functionality, specify the values listed in the following table.

Table 2-39: Settings for using the naming service error detection functionality

Specified parameters	Value
<code>ejbserver.jndi.cache</code>	on
<code>ejbserver.jndi.cache.interval</code>	1 to 2,147,483,647
<code>ejbserver.jndi.cache.interval.clear.option</code>	check

The following is an example of the system properties settings for an EJB client. In this example, the settings are specified in `usrconf.properties`. Furthermore, in this example, the round-robin search functionality is also set concurrently.

Example of the system property settings for an EJB client

```
...
# Cache settings
ejbserver.jndi.cache=on
ejbserver.jndi.cache.interval=60
ejbserver.jndi.cache.interval.clear.option=check

# Define the logical naming service configuration
ejbserver.jndi.namingservice.group.list=g1;g2;g3
ejbserver.jndi.namingservice.group.g1.providerurls= corbaname::hostA:900;corbaname::hostB:
900;corbaname::hostC:900
ejbserver.jndi.namingservice.group.g2.providerurls=
corbaname::hostD:700;corbaname::hostE:700
ejbserver.jndi.namingservice.group.g3.providerurls=
corbaname::hostF:800;corbaname::hostG:800;corbaname::hostH:800
...
```

2.9.5 Notes on the naming service error detection functionality

This subsection describes the notes on the naming service error detection functionality.

(1) Unlock timing

When the naming service error detection functionality is not used, the failed naming service is successfully searched from the client application immediately after the naming service and J2EE server are restarted.

However, when the naming service error detection functionality is used, the lock can only be released at the sweep interval. In the naming management functionality, if the sweep interval does not lapse after the naming service stops, the connection is not established to the actual naming service. In other words, maximum sweep interval time is necessary after the recovery of the naming service functionality until the search is successful. When the naming service error detection functionality is used, we recommend that you specify a short time (recommended value 60) in the sweep interval setup time (value of the `ejbserver.jndi.cache.interval` property).

(2) Differences in behavior until 07-60

In Application Server versions until 07-60 and in version 08-00 and later, the lock timing is different. In version 08-00 and later, in addition to the sweep interval lapse timing, the operating status is checked when the naming service is searched from the client application.

(3) Checking the viability of the naming management functionality

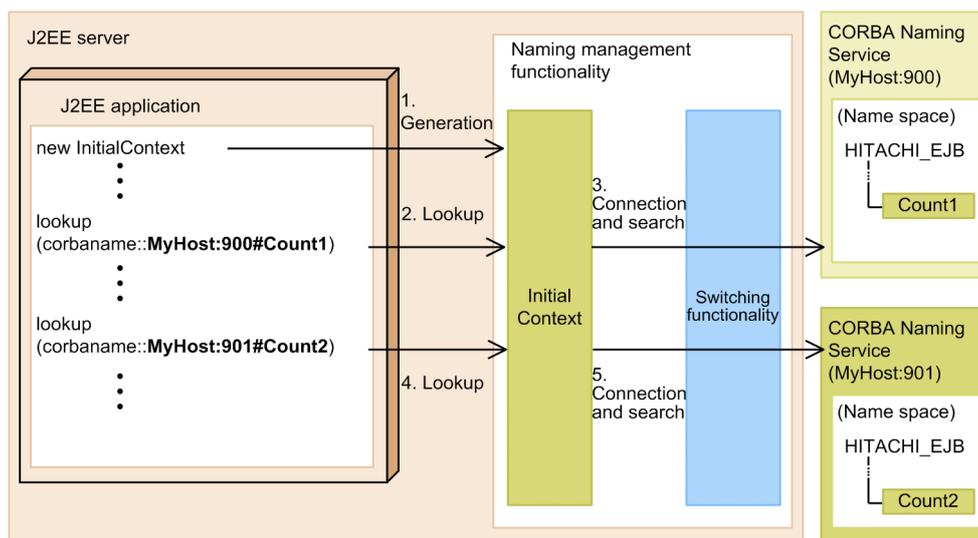
When the naming service error detection functionality is used to check the operating status, the naming service might be considered to have stopped when there is a temporary network error and there is no response due to a high load on the server and during a full garbage collection.

2.10 Switching the CORBA Naming Services

With the naming management functionality, you can switch the CORBA Naming Services connected by JNDI when you search distributed objects through JNDI. You can switch the CORBA Naming Services by passing the name with the prefix `corbaname:` to the argument of the lookup method for the instance of the `InitialContext` class.

The following figure shows the procedure of switching the CORBA Naming Services.

Figure 2-15: Procedure of switching the CORBA Naming Services



The description of the switching procedure of the naming service is as follows. In this procedure, a lookup specifying the prefix `corbaname:` is executed twice from the J2EE application. The connected CORBA Naming Service is switched according to the host name specified after the prefix.

1. The J2EE application generates an instance of the JNDI `javax.naming.InitialContext` class.
2. The J2EE application requests an object search (lookup) for the instance of the `javax.naming.InitialContext` class. At this time, the name `'corbaname::MyHost:900#Count1'` with the prefix `'corbaname:'` is specified.
3. A connection is executed from the naming management functionality that receives the request to the CORBA Naming Service `'MyHost:900'`. After the connection, the EJB home object reference `'Count1'` is searched with the name specified in the lookup.
4. The J2EE application requests an object search (lookup) for the instance of the `InitialContext` class. At this time, the name `'corbaname::MyHost:901#Count2'` with the prefix `'corbaname:'` is specified.
5. A connection is executed from the naming management functionality that receives the request to the CORBA Naming Service `'MyHost:901'`. After the connection, the EJB home object reference `'Count2'` is searched with the name specified in the lookup.

The CORBA Naming Services are switched by using the server management commands to resolve the Enterprise Bean references. For details on the operations, see 9. *Setting up the J2EE Application Properties* in the *uCosminexus Application Server Application Setup Guide*.

Note that when the CORBA Naming Service is running on the local host, specify the computer name or the IP address instead of the character string `"localhost"` in the settings related to the naming service host name.

Customize the J2EE server properties to specify the host name settings for the naming services. For details on how to specify the settings, see 2.3.5 *Execution environment settings*.

2.11 Re-using the EJB home object references (Functionality for re-connecting to the EJB home objects)

The functionality for reconnecting to the EJB home objects enables you to re-use the EJB home objects obtained by the EJB client application when the J2EE server and J2EE applications are restarted after an error in the J2EE server. After the J2EE server and J2EE applications are restarted, the EJB home objects obtained by the EJB client application can be used as are without re-executing a lookup.

This functionality can be used for the EJB home object references of the Session Bean (Stateless Session Bean or Stateful Session Bean).

The following table describes the organization of this section.

Table 2-40: Organization of this section (Functionality for reconnecting to the EJB home objects)

Category	Title	Reference location
Settings	Execution environment settings (J2EE server settings)	2.11.1
Notes	Notes on re-using the EJB home object references	2.11.2

Note: The function-specific explanation is not available for "Explanation", "Implementation" and "Operations".

2.11.1 Execution environment settings (J2EE server settings)

To use the functionality for reconnecting to the EJB home objects, you must specify the J2EE server settings. Specify the J2EE server settings in the `<configuration>` tag of the logical J2EE server (`j2ee-server`) in the Easy Setup definition file.

The following table describes the definition of the functionality for reconnecting to the EJB home objects in the Easy Setup definition file.

Table 2-41: Definition of the functionality for reconnecting to the EJB home objects in the Easy Setup definition file

Specified parameters	Settings
<code>ejbserver.container.ejbhome.sessionbean.reconnect.enabled</code>	Specify whether to enable the functionality for reconnecting to the EJB home objects.

For details on the Easy Setup definition file and the specified parameters, see 4.6 *Easy Setup definition file* in the *uCosminexus Application Server Definition Reference Guide*.

2.11.2 Notes on re-using the EJB home object references

The notes on using the functionality for reconnecting to the EJB home objects are as follows:

- When the J2EE server and J2EE applications are restarted, do not change the J2EE applications. If you change the J2EE applications, you cannot re-use the EJB home object references.
- Fix the communication port of the J2EE server. Also, when you restart the J2EE server, do not change the fixed value.

For details on how to define the value, see 2.14.1 *Fixing the communication port* and 2.14.2 *Fixing the IP address* in the *uCosminexus Application Server EJB Container Functionality Guide*.

- In the user property file for the Java applications, specify `VB_TRANSPARENT` as the value of the EJB client operations for when a remote interface communication error (`ejbserver.container.rebindpolicy` key) occurs.

For details on how to define the value, see 2.13 *Invoking the EJB remote interface* in the *uCosminexus Application Server EJB Container Functionality Guide*.

- When a communication error occurs during EJB invocation, the connection is re-established and the request is sent again if you have specified `VB_TRANSPARENT` as the value of the EJB client operations for when a communication error occurs (`ejbserver.container.rebindpolicy` key).

Therefore, we recommend that you use the functionality for reconnecting to the EJB home objects in the reference node system.

3

Resource Connections and Transaction Management

This chapter describes the resources to which Application Server is able to connect and the connections to the resources. This chapter also describes the management of transactions in the resource connections.

3.1 Organization of this chapter

The following table describes the resource connection and transaction management functionality and reference locations.

Table 3-1: Resource connection and transaction management functionality and reference locations

Functionality	Reference location
Overview of resource connections and transaction management	3.2
Resource connections	3.3
Managing transactions	3.4
Resource sign-on method	3.5
Connecting to a database	3.6
Connecting to a database queue	3.7
Outbound connection with OpenTP1 (SPP or TP1/Message Queue)	3.8
Inbound connection with OpenTP1	3.9
Connection with Cosminexus JMS Provider	3.10
Connecting to the SMTP server	3.11
Using the JavaBeans resources	3.12
Connection with other resources	3.13
Functionality for performance tuning	3.14
Functionality for fault tolerance	3.15
Other resource adapter functionality (For the resource adapters conforming to the Connector 1.5 specifications)	3.16
Connection pool clustering functionality	3.17
Connection test for resources	3.18
Functionality for operations in the firewall environment	3.19
Notes on starting transactions with EJB client applications	3.20

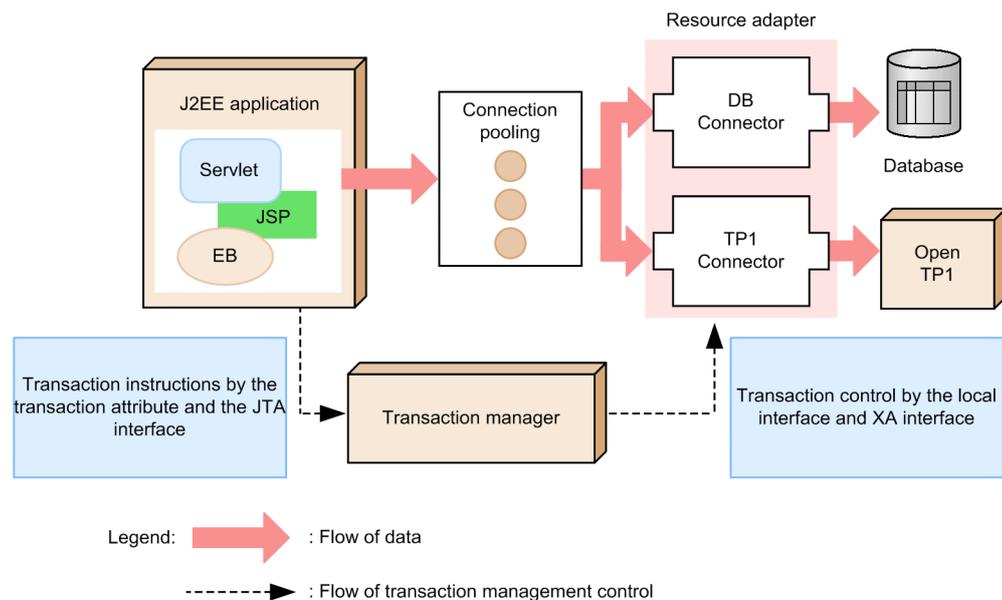
3.2 Overview of resource connections and transaction management

The J2EE components, such as EJBs, servlets, and JSPs, included in the J2EE applications on the J2EE server, can connect to the resources, such as databases and OpenTP1. The resource adapter is used to connect to the resources, such as databases and OpenTP1. With Application Server, you can use the resource adapters conforming to the Connector 1.0 specifications or Connector 1.5 specifications. Application Server also provides the functionality for connecting to the resources, such as the SMTP server and JavaBeans resources, without using a resource adapter.

Application Server also provides the connection pooling and transaction management functionality to access the resources with highly efficient and reliable methods. If you use connection pooling, the connections are pooled for resources, and the connections can be used efficiently. Also, the connections with errors are removed from the connection pool appropriately. If you use the transaction management functionality, the transaction manager can properly control the resource access transactions based on the instructions from the transaction attributes and the JTA interface (`UserTransaction`) specified for each EJB method. A global transaction is used to manage the transactions for multiple resources. If you use a global transaction, you ensure the consistency of updates between resources because the transactions are managed using the two-phase commit protocol.

The following figure shows an example of connecting to the resources using the connection pooling and transaction management functionality.

Figure 3-1: Example of connecting to resources using the connection pooling and transaction management functionality



3.3 Resource connections

This section describes the connection methods for each resource type, and the types of resource adapters used for connecting to resources.

This section also describes the usage methods, functionality, and notes as an explanation for using resource adapters.

The following table describes the organization of this section.

Table 3-2: Organization of this section (Resource connections)

Category	Title	Reference location
Explanation	How to connect to resources	3.3.1
	Types of resource adapters	3.3.2
	How to use resource adapters	3.3.3
	Resource adapter functionality	3.3.4
	Functionality other than the resource adapter functionality	3.3.5
Implementation	Implementation for connecting to the resources	3.3.6
Settings	How to set up resource adapters	3.3.7
	Procedure for resource adapter settings (To deploy and use the resource adapter as a J2EE resource adapter)	3.3.8
	Procedure for resource adapter settings (To include and use resource adapters in J2EE applications)	3.3.9
	Procedure for resource adapter settings (To use resource adapters with Inbound)	3.3.10
	Procedure for resource adapter settings (To use connection pool clustering)	3.3.11
	Connection settings for using components other than resource adapters	3.3.12
Notes	Notes on resource adapters	3.3.13

Note: The functionality-specific explanation is not available for "Operations".

3.3.1 How to connect to resources

Based on the types of resources, Application Server includes the resources that use resource adapters for connection and the resources that do not use resource adapters for connection. This subsection gives an overview of how to connect to each resource for each resource type.

(1) Resources that use resource adapters for connection

The resources that use resource adapters for connection are as follows:

Database

You can connect Application Server to a database. To connect to the database, use DB Connector as the resource adapter.

You can connect to the following databases with DB Connector:

- HiRDB
- Oracle
- SQL Server[#]
- XDM/RD E2

[#]: You can only connect to SQL Server in the case of Windows.

For details on connecting to a database, see *3.6.1 Overview of DB Connector-based connections*.

Database queue

You can connect Application Server to the database queues used with Cosminexus RM. To connect to the database queue, use DB Connector for Cosminexus RM and Cosminexus RM as the resource adapter.

You can connect to the following databases with DB Connector for Cosminexus RM and Cosminexus RM:

- HiRDB
- Oracle

For details on connecting to a database queue, see *3.7.1 Overview of connections using DB Connector for Cosminexus RM and Cosminexus RM*.

OpenTP1

You can connect Application Server to the OpenTP1 products - SPP, TP1/Message Queue, and Outbound. To connect to OpenTP1 SPP and Outbound, use uCosminexus TP1 Connector as the resource adapter. To connect to TP1/Message Queue, use TP1/Message Queue - Access as the resource adapter.

Also, you can use Inbound to connect to Application Server from OpenTP1 SUP. To use Inbound to connect from OpenTP1 SUP, use the TP1 inbound adapter as the resource adapter.

This subsection describes the details on OpenTP1 connections for each resource adapter used for the connections. The following table describes reference locations.

Table 3-3: Reference locations for the OpenTP1 connection details

Connection method	Reference location
Connections using uCosminexus TP1 Connector	<i>3.8.1 Connections using uCosminexus TP1 Connector</i>
Connections using TP1/Message Queue -Access	<i>3.8.2 Connections using TP1/Message Queue - Access</i>
Connections using the TP1 inbound adapter	<i>4. Invoking Application Server from OpenTP1 (TP1 Inbound Integrated Function)</i>

CJMSP Broker

When you use the Cosminexus JMS Provider functionality, you can connect Application Server with CJMSP Broker. To connect to CJMSP Broker, use the CJMSP resource adapter as the resource adapter.

For details on connecting to CJMSP Broker using the CJMSP resource adapter, see *7. Cosminexus JMS Provider*.

Other resources

Regardless of the resource type, you can connect Application Server to the resources connectable with the resource adapters conforming to the Connector 1.0 specifications or Connector 1.5 specifications.

For details on the available resource adapters, see *3.13 Connection with other resources*. For details on the functionality available for Application Server when you use the resource adapters conforming to the Connector 1.5 specifications, see *3.16 Other resource adapter functionality (For the resource adapters conforming to the Connector 1.5 specifications)*.

(2) Resources that do not use resource adapters for connection

The resources that do not use resource adapters for connection are as follows:

SMTP server

You can connect Application Server to the SMTP server. For details on connecting to the SMTP server, see *3.11 Connecting to the SMTP server*.

JavaBeans

You can use JavaBeans resources as resources. For details on using JavaBeans resources, see *3.12 Using JavaBeans resources*.

3.3.2 Types of resource adapters

You can use the resource adapters conforming to the Connector 1.0 specifications or Connector 1.5 specifications with Application Server.

This subsection describes the resource adapters conforming to the respective specifications. This subsection also describes the differences in the DD schema of resource adapters.

(1) Resource adapters conforming to the Connector 1.0 specifications

You can use the following resource adapters with Application Server:

- DB Connector
- DB Connector for Cosminexus RM and Cosminexus RM
- uCosminexus TP1 Connector
- TP1/Message Queue - Access

You can use the functionality supported by the Connector 1.0 specifications with these resource adapters.

Note that with DB Connector and DB Connector for Cosminexus RM, you can use the functionality added with Application Server in addition to the functionality supported by the Connector 1.0 specifications. For details on the available functionality, see *3.3.4 Resource adapter functionality*.

Tip

With Application Server, apart from these resource adapters, you can use the resource adapters conforming to the Connector 1.0 specifications of the standard specifications. However, when you use the resource adapters conforming to the Connector 1.0 specifications of the standard specifications, the settings in the following tags in the DD (`ra.xml`) are ignored.

- `<security-permission>`
Use the `server.policy` file for the security settings.
 - `<authentication-mechanism>`
Regardless of the settings, `BasicPassword` is applied.
-

(2) Resource adapters conforming to the Connector 1.5 specifications

You can use the resource adapters conforming to the Connector 1.5 specifications with Application Server. From the contracts in the Connector 1.5 specifications, the functionality corresponding to the following contracts can be used with Application Server:

- Lifecycle Management
- Work Management
- Message inflow
- Transaction inflow

Application Server provides the following resource adapters:

- TP1 inbound adapter
- CJMSP resource adapter

For details on the available functionality, see *3.3.4 Resource adapter functionality*.

Also, for resource adapters conforming to the Connector 1.5 specifications, see *3.13.1 Resource adapters used for connection with other resources*.

(3) Differences in the schema of the resource adapters conforming to the Connector 1.0 specifications and the Connector 1.5 specifications

This section describes the differences in the DD schema of the resource adapters conforming to the Connector 1.0 specifications and resource adapters conforming to the Connector 1.5 specifications. The DD for a resource adapter is `ra.xml`.

The following table describes the main changes from the schema defined in the Connector 1.0 specifications to the schema defined in the Connector 1.5 specifications. For the changes other than those described in this table, see the Connector 1.5 specifications.

Table 3-4: Main changes from the schema defined in the Connector 1.0 specifications to the schema defined in the Connector 1.5 specifications

Content changed in the Connector 1.5 specifications	DD content in the Connector 1.5 specifications
Specification of the implementation class of the <code>javax.resource.spi.ResourceAdapter</code> interface	<code><connector>-<resourceadapter>-<resourceadapter-class> tag</code> <code><connector>-<resourceadapter>-<config-property> tag</code>
Specification of the Outbound resource adapter	<code><connector>-<resourceadapter>-<outbound-resourceadapter> tag</code>
Specification of the Inbound resource adapter	<code><connector>-<resourceadapter>-<inbound-resourceadapter> tag</code>
Specification of <code>adminobject</code>	<code><connector>-<resourceadapter>-<adminobject> tag</code>

An overview of the content changed in the Connector 1.5 specifications is as follows:

Specification of the implementation class of the `javax.resource.spi.ResourceAdapter` interface

The implementation class of the `javax.resource.spi.ResourceAdapter` interface and the elements specifying their configuration properties were added. For details on the functionality that can be implemented with the addition of the `javax.resource.spi.ResourceAdapter` interface, see *3.16.1 Managing the resource adapter lifecycle* and *3.16.2 Managing the resource adapter work*.

Specification of the Outbound resource adapter

The elements used for explicitly defining the Outbound resource adapter were added.

Note that you can define multiple connections in one DD with the Outbound resource adapter. For details on specifying multiple connection definitions, see *3.16.6 Specifying multiple connection definitions*.

Specification of the Inbound resource adapter

The elements used for explicitly defining the Inbound resource adapter were added.

Specification of `adminobject`

The elements specifying the information related to the Administered objects were added.

(4) Types of RAR files for each resource adapter

The resource adapters that define properties differ depending on the information such as the types of resources to be connected to and the transactions to be used. This section describes the resource adapters used in the following cases:

- To use DB Connectors
- To use DB Connector for Cosminexus RM and Cosminexus RM
- To use other resource adapters

(a) To use DB Connectors

The files for DB Connectors differ depending on the database type to be connected to and the transaction type to be used. The following table describes the DB Connector types.

Table 3-5: DB Connector types

RAR file name	Explanation
<code>DBConnector_HiRDB_Type4_CP.rar</code>	Select this file when you use HiRDB Type4 JDBC Driver to connect to HiRDB or XDM/RD E2 without using a local transaction or transaction management.
<code>DBConnector_HiRDB_Type4_XA.rar</code>	Select this file when you use HiRDB Type4 JDBC Driver to connect to HiRDB using a global transaction.
<code>DBConnector_Oracle_CP.rar</code>	Select this file when you use Oracle JDBC Thin Driver to connect to Oracle without using a local transaction or transaction management.

RAR file name	Explanation
DBConnector_Oracle_XA.rar	Select this file when you use Oracle JDBC Thin Driver to connect to Oracle using a global transaction.
DBConnector_SQLServer_CP.rar	Select this file when you use SQL Server JDBC Driver to connect to SQL Server (only in Windows) without using a local transaction or transaction management.
DBConnector_CP_ClusterPool_Root.rar	This is the root resource adapter of the connection pool clustering functionality. Select this file when a member resource adapter belonging to the root resource adapter connects to Oracle without using a local transaction or transaction management.
DBConnector_Oracle_CP_ClusterPool_Member.rar	This is the member resource adapter of the connection pool clustering functionality. Select this file when you use Oracle JDBC Thin Driver to connect to Oracle without using a local transaction or transaction management. Note that you cannot specify this file in the resource references of J2EE applications.

Note: When you prepare a new DB Connector RAR file, you can use the template files of the HITACHI Connector Property files provided with Application Server to define properties. The template files of the HITACHI Connector Property files are provided for all the DB Connector RAR files. For details on the provided template files, see *4.1.14 Template files of the HITACHI Connector Property files* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

(b) To use DB Connector for Cosminexus RM and Cosminexus RM

When you connect to a database by integrating the system with **Cosminexus RM**, you must import both, resource adapter for the Cosminexus RM integration (**DB Connector for Cosminexus RM**) and the resource adapter provided with Cosminexus RM. For details on the resource adapter provided with Cosminexus RM, see the manual *uCosminexus Application Server Reliable Messaging*.

The files for DB Connector for Cosminexus RM differ depending on the transaction type to be used and the database type to be connected to. The following table describes the types of DB Connector for Cosminexus RM.

Table 3-6: DB Connector for Cosminexus RM types

RAR file name	Explanation
DBConnector_HiRDB_Type4_CP_Cosminexus_RM.rar	Select this file when you use HiRDB Type4 JDBC Driver to connect to HiRDB without using a local transaction or transaction management.
DBConnector_HiRDB_Type4_XA_Cosminexus_RM.rar	Select this file when you use HiRDB Type4 JDBC Driver to connect to HiRDB using a global transaction.
DBConnector_Oracle_CP_Cosminexus_RM.rar	Select this file when you use Oracle JDBC Thin Driver to connect to Oracle without using a local transaction or transaction management.
DBConnector_Oracle_XA_Cosminexus_RM.rar	Select this file when you use Oracle JDBC Thin Driver to connect to Oracle using a global transaction.

Note: When you prepare a new RAR file for DB Connector for Cosminexus RM, you can use the template files of the HITACHI Connector Property files provided with Application Server to define properties. The template files of the HITACHI Connector Property files are provided for all the DB Connector RAR files. For details on the provided template files, see *4.1.14 Template files of the HITACHI Connector Property files* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

Tip

When you use DB Connector for Cosminexus RM and Cosminexus RM for accessing the JDBC and JMS in the same transaction, you can use a local transaction and conclude the single-phase commit operations for a global transaction by sharing the same physical connection. The conditions for concluding the single-phase commit operations are as follows:

- The database system used for DB Connector for Cosminexus RM and Cosminexus RM is the same, and the sign-on method and the security information (user name and password) is identical.
- The value `Shareable` is specified in the `<res-sharing-scope>` tag in the property files of the J2EE applications using the resource adapters (such as the Session Bean property file and Entity Bean property file).

(c) To use other resource adapters

To connect to **OpenTP1 SPP** and Outbound, you use the resource adapters uCosminexus TP1 Connector and TP1/Client/J. For details, see the uCosminexus TP1 Connector documentation and the *OpenTP1 Version 7 TP1/Client User's Guide TP1/Client/J*.

To connect from **OpenTP1 SUP** using Inbound, you use the TP1 inbound adapter. For details, see *4. Invoking Application Server from OpenTP1 (TP1 Inbound Integrated Function)*.

Also, to connect to **CJMSP Broker** when you use Cosminexus JMS Provider, you use the CJMSP resource adapter. For details, see *7. Cosminexus JMS Provider*.

Note that to use a new RAR file for the TP1 inbound adapter or the CJMSP resource adapter, you can use the template file provided with Application Server to define properties. For details, see *4.1.14 Template files of the HITACHI Connector Property files* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

To connect to **TP1/Message Queue**, use the resource adapter TP1/Message Queue - Access. For details, see the *OpenTP1 Version 7 TP1/Message Queue - Access User Guide*.

With Application Server, you can also connect to any resource by using the resource adapters conforming to the Connector 1.0 specifications or Connector 1.5 specifications. To use these resource adapters, see the documentation for resource adapters.

3.3.3 How to use the resource adapters

This subsection describes how to use the resource adapters. There are two types of methods of using the resource adapters to connect to resources:

- Method of deploying and using the resource adapter as a J2EE resource adapter
- Method of including and using the resource adapter in a J2EE application

The following is a description of the methods:

(1) Deploying and using the resource adapter as a J2EE resource adapter

Deploy the resource adapter imported into the J2EE server as a shared standalone module. The resource adapter is now available to all the J2EE applications running on that J2EE server. The resource adapters allocated to the J2EE server are called the **J2EE resource adapters**.

(2) Including and using the resource adapter in a J2EE application

Include the resource adapter in a J2EE application to use the resource adapter. The EJBs and WARs included in the same J2EE application can use the resource adapter. For details on how to include and use the resource adapter in a J2EE application, see *3.3.9 Procedure for resource adapter settings (To include and use the resource adapter in the J2EE application)*.

Note that the resource adapters that can be included in a J2EE application are limited. For details, see *(3) Resource adapters available in each usage method*.

(3) Resource adapters available in each usage method

The resource adapters must be compatible with the Application Server versions. Also, you can deploy and use the resource adapters conforming to the Connector 1.0 specifications and Connector 1.5 specifications with Application Server. However, when you include and use the resource adapters in J2EE applications, you cannot use the following resource adapters:

- Resource adapters specified in `XATransaction`
- Resource adapters containing the native library
- Resource adapters for which the starting order must be controlled
- Resource adapters using the Application Server-specific functionality

The following table describes the usage of the resource adapters provided by Application Server.

Table 3-7: Usage of the resource adapters provided by Application Server

Resource adapter		Usage		
		Deploying and using the resource adapter as a J2EE resource adapter	Including and using the resource adapter in a J2EE application	
DB Connector	Specified in <code>NoTransaction</code> or <code>LocalTransaction</code>		Y	Y
	Specified in <code>XATransaction</code>		Y	--
	Cluster configuration	Root resource adapter	Y	--
		Member resource adapter	Y	--
uCosminexus TP1 Connector	Specified in <code>NoTransaction</code> or <code>LocalTransaction</code>		Y	Y
	Specified in <code>XATransaction</code>		Y	--
TP1/Message Queue - Access		Y	--	
Cosminexus RM	Cosminexus RM (without database)		Y	--
	Cosminexus RM (with database)		Y	--
	DB Connector for Cosminexus RM		Y	--
TP1 inbound adapter		Y	--	
CJMSP resource adapter		Y	--	

Legend:

Y: Available

--: Not applicable

If you execute the following operations for the unavailable resource adapters, an error message is output and the operation fails:

- Including and importing the resource adapter in a J2EE application
- Including the resource adapter in a J2EE application and performing a connection test or starting the resource adapter

3.3.4 Resource adapter functionality

This subsection describes the following resource adapter functionality:

- DB Connector
- DB Connector used for connection pool clustering (root resource adapter and member resource adapter)
- DB Connector for Cosminexus RM
- TP1 inbound adapter
- CJMSP resource adapter
- Other resource adapters conforming to the Connector 1.5 specifications

For details on any other resource adapter functionality, see the documentation for the resource adapter you want to use.

The following table describes the functionality available for each type of resource adapter. For details on the functionality, see the description in the reference location.

Table 3-8: Functionality available for each resource adapter type

Functionality	Types of resource adapters							Reference location
	DB Connector	Root resource adapter	Member resource adapter	DB Connector for Cosminexus RM	TP1 inbound adapter	CJMSP resource adapter	Other resource adapters conforming to the Connector 1.5 specifications ^{#1}	
Connection pooling	Y	N	M	Y	--	Y	Y	3.14.1
Connection pool warming up	Y	N	Y	Y	--	Y	Y	3.14.2
Connection count adjustment functionality	Y	N	Y	Y	--	Y	Y	
Connection sharing or association	Y	N	Y	Y	--	N	Y	3.14.3
Statement pooling	Y	N	Y	Y	--	N	Y	3.14.4
Caching DataSource objects	Y	Y	N	Y	--	N	N	3.14.7
Optimizing the container-managed sign-on for DB Connector	Y	N	Y	Y	--	N	N	3.14.8
Detecting connection errors	Y	N	M	Y	--	N	Y	3.15.1
Waiting for a connection when connections deplete	Y	N	M	Y	--	Y	Y	3.15.2
Retrying to obtain a connection	Y	N	N	Y	--	Y	Y	3.15.3
Displaying the connection pool information	Y	N	Y	Y	--	Y	N	3.15.4
Clearing the connection pool	Y	N	Y	Y	--	Y	Y	3.15.5
Automatically closing connections	Y	N	Y	Y	--	N	Y	3.15.6
Connection sweeper	Y	N	Y	Y	--	Y	Y	3.15.7
Statement cancellation	Y	N	Y	Y	--	N	Y	3.15.8
Output of the SQL statement for troubleshooting	M	N	M	M	--	N	N	3.15.10
Automatically closing objects	Y	N	Y	Y	--	N	N	3.15.11
Managing the resource adapter lifecycle	N	N	N	N	Y	Y	Y	3.16.1
Managing the resource adapter work	N	N	N	N	Y	Y	Y	3.16.2
Message inflow	N	N	N	N	Y	Y	Y	3.16.3

Functionality	Types of resource adapters							Reference location
	DB Connector	Root resource adapter	Member resource adapter	DB Connector for Cosminexus RM	TP1 inbound adapter	CJMSP resource adapter	Other resource adapters conforming to the Connector or 1.5 specifications ^{#1}	
Transaction inflow	N	N	N	N	Y	N	N	3.16.4
Looking up the Administered objects	N	N	N	N	--	Y	Y	3.16.5
Specifying multiple connection definitions	N	N	N	N	--	Y	Y	3.16.6
Suspending a connection pool	N	N	Y	N	--	N	N	3.17.4
Restarting a connection pool	N	N	Y	N	--	N	N	
Displaying the connection pool status	Y	N	Y	Y	--	N	Y	
Connection test for resources	Y	Y	Y	Y	--	Y	Y	3.18
Assigning optional names to J2EE resources	Y	Y	N	Y	--	Y	Y	2.6
PRF trace of the connection ID	Y	Y	Y	Y	--	--	--	#2

Legend:

M: Always enabled

Y: Available

N: Not available

--: Not applicable

#1: This table describes whether the functionality provided by Application Server is available. For details on the functionality provided by resource adapters, see the documentation for the resource adapter you want to use.

#2: See 4.6 Trace based performance analysis in the *uCosminexus Application Server Maintenance and Migration Guide*.

3.3.5 Functionality other than the resource adapter functionality

This subsection describes the functionality implemented apart from the resource adapter functionality. The functionality described in this subsection can be used regardless of the types of resource adapters.

The following table describes the functionality implemented apart from the resource adapter functionality. For details on the functionality, see the description in the reference location.

Table 3–9: Functionality other than the resource adapter functionality

Functionality	Reference location
Light transaction	3.14.5
In-process transaction service	3.14.6
Transaction timeout and statement cancellation	3.15.8
Transaction recovery	3.15.9

3.3.6 Implementation for connecting to the resources

To connect from the application to a resource, the resource references must be obtained for Enterprise Beans and servlets. The methods of obtaining the resource references include the method of using lookup and the method of using DI (Dependency Injection).

Note that when using EJB 3.0 or later, use DI to obtain the resource references.

(1) Method of obtaining the resource references by using lookup

When you want to use the lookup method, connect the application to the resource as follows:

1. Use the JNDI to look up the factory class that is used for obtaining the connection to the resource.
Specify the name to be looked up using the DD of Enterprise Beans and servlets. The applicable tag is the `<resource-ref-name>` tag in the `<resource-ref>` tag.
2. Use the connection factory class to obtain the connection.
3. Use the obtained connection to connect to the resource.
4. Close the used connection.

If you use connection pooling, a pooled connection is obtained in step 2 and the connection is returned to the pool in step 4. The user program does not require a connection pooling-aware coding.

(2) Method of obtaining the resource references by using the DI

To use the DI for obtaining the resource references, the DD definitions are unnecessary. For an overview of the DI and the notes on DI usage, see *12.4 Using the DI*.

(3) Notes on connecting to the resources

If you obtain a resource connection through a user program, make sure you close the connection after use. Specifically, close the connection using the `finally` clause so that the connection is definitely closed even when an exception occurs.

Note that the time when the `finalize` method is invoked depends on the timing of JavaVM garbage collection, so do not specify the design of using the `finalize` method to close the connection. If a connection is not closed properly through the user program, the maximum number of connections that can be obtained is reached and you might not be able to obtain more connections.

3.3.7 How to set up resource adapters

The method of setting up resource adapters includes the following two methods:

- **Method of deploying and setting up the resource adapter as a J2EE resource adapter**
In this method, you directly deploy and set up the resource adapter on the J2EE server.
- **Method of including and setting the resource adapter in a J2EE application**
This method is specified for the resource adapters that are included and used in a J2EE application.

Reference note

If the resource adapter is DB Connector, the TP1 inbound adapter, or the CJMSP resource adapter, you can use the template files of the HITACHI Connector Property files provided by Application Server. If you use the template file, you can edit the HITACHI Connector Property file before you import the resource adapter. Therefore, the HITACHI Connector Property file to be edited need not be obtained with the server management commands (`cjgetrarprop` command or `cjgetresprop` command).

For details on the storage destination of the template files of the HITACHI Connector Property files and the notes on the usage of the template files, see *4.1.14 Template files of the HITACHI Connector Property files* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

! Important note

To use the resource adapters that were used with the old Application Server versions, the resource adapters must be migrated. For details on how to migrate the resources, see *10.9 Migrating the resource adapters* in the *uCosminexus Application Server Maintenance and Migration Guide*.

3.3.8 Procedure for resource adapter settings (To deploy and use the resource adapter as a J2EE resource adapter)

This subsection describes the following procedure for deploying and using the resource adapter as a J2EE resource adapter:

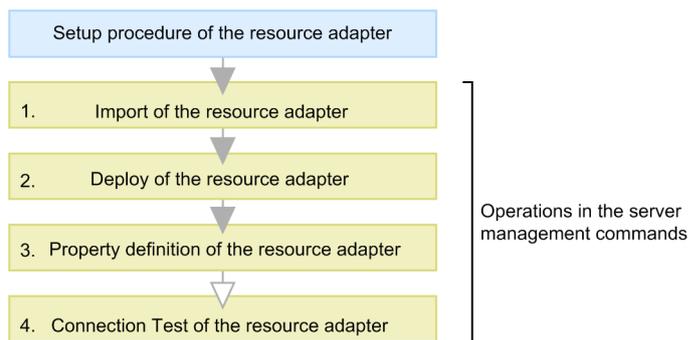
- Procedure for setting up a new resource adapter
- Procedure for changing the resource adapter settings
- Procedure for replacing resource adapters

Use the server management commands to set up resource adapters.

(1) Procedure for setting up a new resource adapter

The following figure shows the procedure for setting up a new resource adapter when Application Server is connected to a database or another resource.

Figure 3-2: Procedure for setting up a new resource adapter



Legend: ▼ : Required work ▽ : Optional work

A description of points 1 to 4 in the figure is as follows:

1. Use the server management commands to import the resource adapter.
Use the `cjimportres` command to import the resource adapter.
You import a different RAR file when DB Connector is used to connect to a database and when other resource adapters are used to connect to various resources such as OpenTPI. For details on the resource adapters to be imported, see *3.3.2 Types of resource adapters*.
2. Use the server management commands to deploy the resource adapter.
Use the `cjdeployrar` command to deploy the resource adapter.
If you deploy a resource adapter, you can use the resource adapter as a J2EE resource adapter. A *J2EE resource adapter* is a resource adapter that is deployed as a shared standalone module for the J2EE server. If you deploy a resource adapter that was imported with the server management commands, the resource adapter becomes available to all the J2EE applications running on the J2EE server.
3. Use the server management commands to define resource adapter properties.
Use the `cjgetrarprop` command to obtain the HITACHI Connector Property file, edit the file, and then use the `cjsetrarprop` command to apply the edited content.
For details on the resource adapter properties specified for the functionality to be used, see the following locations:

- Settings for using the resource connection and transaction management functionality
3.4.12 Settings in the execution environment
 - Functionality for performance tuning
3.14.10 Settings in the execution environment
 - Functionality for fault tolerance
3.15.13 Settings in the execution environment
 - Setting the optional names for J2EE resources
2.6.6 Setting the optional names for the J2EE resources
4. Use the server management commands to implement the resource adapter connection test.
Use the `cjtestres` command to implement the resource adapter connection test. For details on the content to be verified in the connection test for each resource, see *3.18 Connection test for resources*.

! Important note

The connection test for connecting to a database by using DB Connector for Cosminexus RM and Cosminexus RM involves the following order:

1. Start DB Connector for Cosminexus RM.
2. Implement the connection test for Cosminexus RM.
3. Start Cosminexus RM.
4. Implement the connection test for DB Connector for Cosminexus RM.

For details on the connection test of the J2EE resource adapters used when DB Connector for Cosminexus RM is used, see *2.7 Functionality for DB Connector for Cosminexus RM* in the manual *uCosminexus Application Server Reliable Messaging*.

For details on the operations with the server management commands, see *3. Basic Operations of the Server Management Commands* in the *uCosminexus Application Server Application Setup Guide*. Also, for details on the commands, see *2.4 Resource operation commands used with the J2EE server* in the *uCosminexus Application Server Command Reference Guide*. For details on the property files, see *4. Property Files Used for Setting Up the Resources* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

! Important note

To use a resource adapter, you must resolve the references from the J2EE application to the resource adapter. When you define the properties for a J2EE application that uses resource adapters, resolve the references from the J2EE application to the resource adapter.

Reference note

In the cases such as those described below, you can set up the resource adapter efficiently by exporting or importing the resource adapter:

- When you export the resource adapter set up in the development environment, and then import the resource adapter into the operating environment
- When you export the resource adapter that is already running in the operating environment, and then import the resource adapter into an extended J2EE server

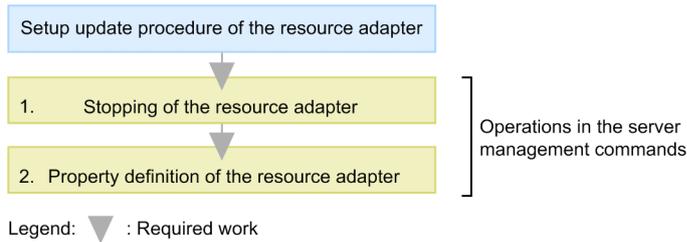
Execute the export and import operations using `cjexportrar` and `cjimportres`.

Note that you cannot export or import resource adapters between hosts with different Application Server versions and platforms. To set up a resource adapter on the host that exports the resource adapter and the host with a different Application Server version and platform, set up a new resource adapter.

(2) Procedure for changing the resource adapter settings

This section describes the procedure for changing the settings for a deployed resource adapter. The following figure shows the procedure for changing the settings.

Figure 3-3: Procedure for changing the resource adapter settings



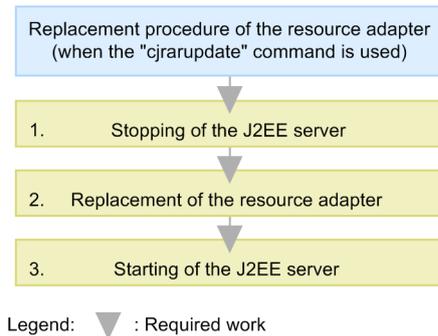
A description of points 1 and 2 in the figure is as follows:

1. Use the server management commands to stop the resource adapter.
Use the `cjstoprar` command to stop the resource adapter. Note that before you stop the resource adapter, make sure you stop all the J2EE applications that are using the resource adapter.
2. Use the server management commands to define the resource adapter properties.
The resource adapter is already deployed; therefore, use the `cjgetrarprop` command to obtain the property file, edit the file, and then use the `cjsetrarprop` command to apply the edited content.

(3) Procedure for replacing the resource adapters

This section describes the procedure for replacing the resource adapters. The following figure shows the procedure for replacing the resource adapters.

Figure 3-4: Procedure for replacing the resource adapters

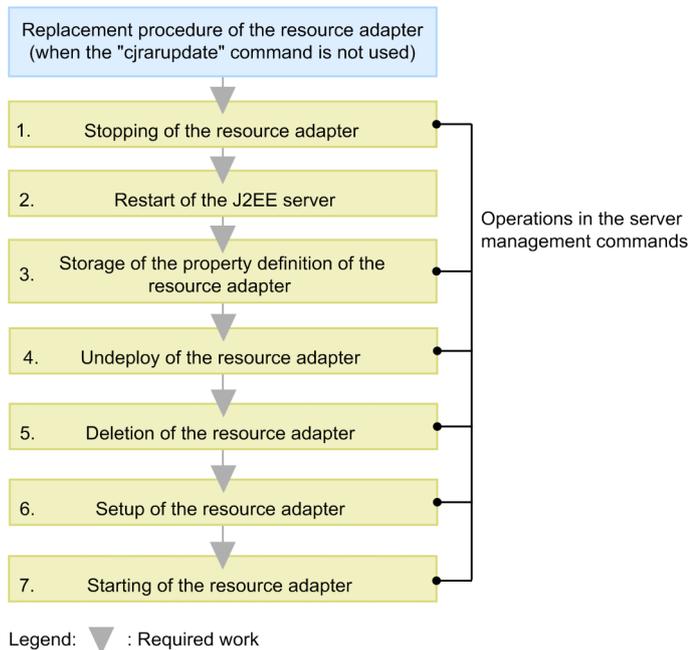


A description of points 1 to 3 in the figure is as follows:

1. Stop the J2EE server.
Use the `cjstopsv` command to stop the J2EE server.
2. Replace the resource adapter.
Use the `cjrarupdate` command to replace the resource adapter.
3. Start the J2EE server.
Use the `cjstartsv` command to start the J2EE server.

You can also replace a resource adapter without using the `cjrarupdate` command. The following figure shows the procedure for replacing the resource adapters without using the `cjrarupdate` command.

Figure 3-5: Procedure for replacing the resource adapters (when the `cjrarupdate` command is not used)



A description of points 1 to 7 in the figure is as follows:

1. Use the server management commands to stop the resource adapter.
Use the `cjstoprar` command to stop the resource adapter you want to replace. Note that before you stop the resource adapter, make sure you stop all the J2EE applications that are using the resource adapter.
2. Restart the J2EE server.
Use the `cjstopsv` command to stop the J2EE server, and then use the `cjstartsv` command to start the J2EE server.
3. Use the server management commands to save the resource adapter property definitions.
To inherit the resource adapter property definitions, use the `cjgetrarprop` command or `cjgetresprop` command to obtain the HITACHI Connector Property file of the resource adapter.
4. Use the server management commands to undeploy the resource adapter.
Use the `cjundeployrar` command to undeploy the resource adapter you want to replace.
5. Use the server management commands to delete the resource adapter.
Use the `cjdeleteres` command to delete the resource adapter you want to replace.
6. Use the server management commands to set up the resource adapter.
Set up the resource adapter as described in (1) *Procedure for setting up a new resource adapter*. To inherit the resource adapter property definitions, use the HITACHI Connector Property file obtained in point 3.
7. Use the server management commands to start the resource adapter.
Use the `cjstartrar` command to start the resource adapter. Note that after you start the resource adapter, start the J2EE applications that use the resource adapter.

3.3.9 Procedure for resource adapter settings (To include and use the resource adapter in the J2EE application)

Use the server management commands to set up the resource adapter. This subsection describes the procedure for resource adapter settings to include and use the resource adapter in the J2EE application.

Note that the method of setting up a resource adapter when the resource adapter is included and used in an application includes the following two methods:

- Method of importing the J2EE application containing the resource adapter into the J2EE server
- Method of adding the resource adapter in a J2EE application already imported into the J2EE server

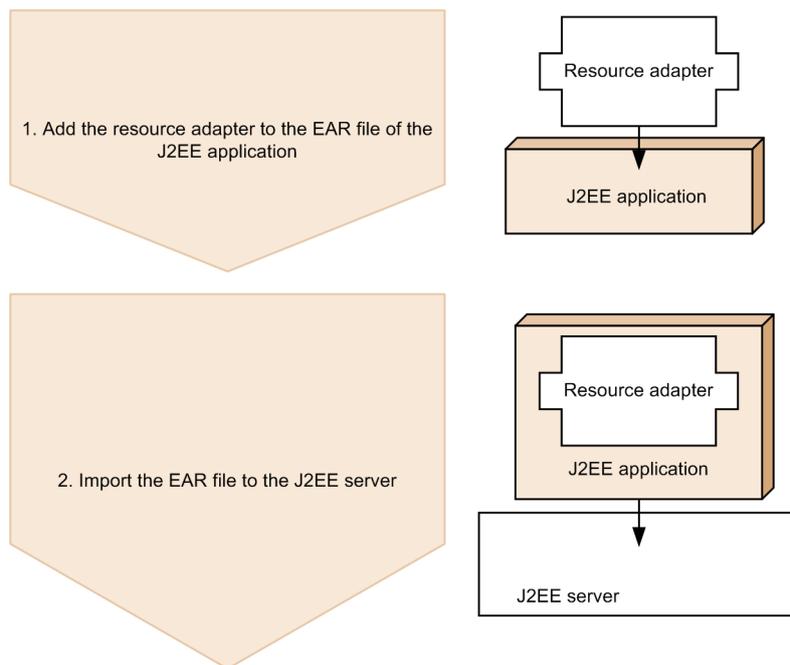
This subsection describes each of the methods.

Note that the resource adapters included in a J2EE application are limited. For details on the resource adapters included in a J2EE application, see 3.3.3 *How to use the resource adapters*.

(1) Method of importing the J2EE application containing the resource adapter into the J2EE server

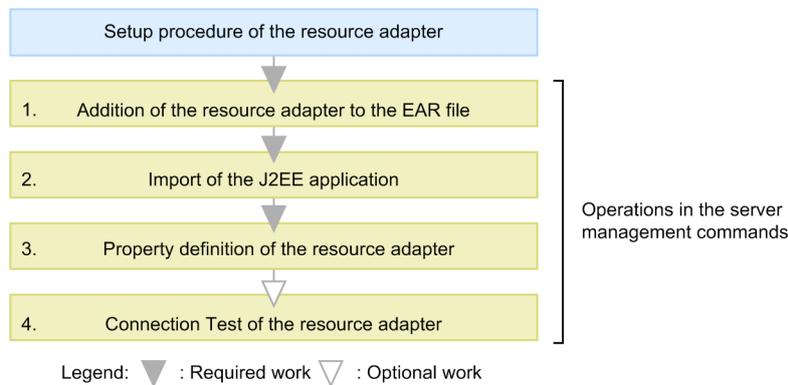
In this method, you create a J2EE application including the RAR file of the resource adapter, and then import that EAR file into the J2EE server. The following figure gives an overview of this method.

Figure 3-6: Overview of the method of importing the J2EE application containing the resource adapter into the J2EE server



The following figure shows the procedure for importing a J2EE application containing a resource adapter into the J2EE server.

Figure 3-7: Procedure for importing a J2EE application containing a resource adapter into the J2EE server



A description of points 1 to 4 in the figure is as follows:

1. Use the server management commands to add the RAR file of the resource adapter into the EAR file of the J2EE application.

Use the `cjaddapp` command to add the RAR file of the resource adapter into the EAR file of the J2EE application.

The resource adapters included in the EAR file are limited. For details on the resource adapters included in the EAR file, see *3.3.3 How to use the resource adapters*.

2. Use the server management commands to import the J2EE application.

Use the `cjimportapp` command to import the J2EE application.

3. Use the server management commands to define the resource adapter properties.

Use the `cjgetappprop` command to obtain the HITACHI Connector Property file, edit the file, and then use the `cjsetappprop` command to apply the edited content.

For details on the resource adapter properties specified for the functionality to be used, see the following locations:

- Settings for using the resource connection and the transaction management functionality
3.4.12 Settings in the execution environment
- Functionality for performance tuning
3.14.10 Settings in the execution environment
- Functionality for fault tolerance
3.15.13 Settings in the execution environment
- Setting the optional names for J2EE resources
2.6.6 Setting the optional names for the J2EE resources

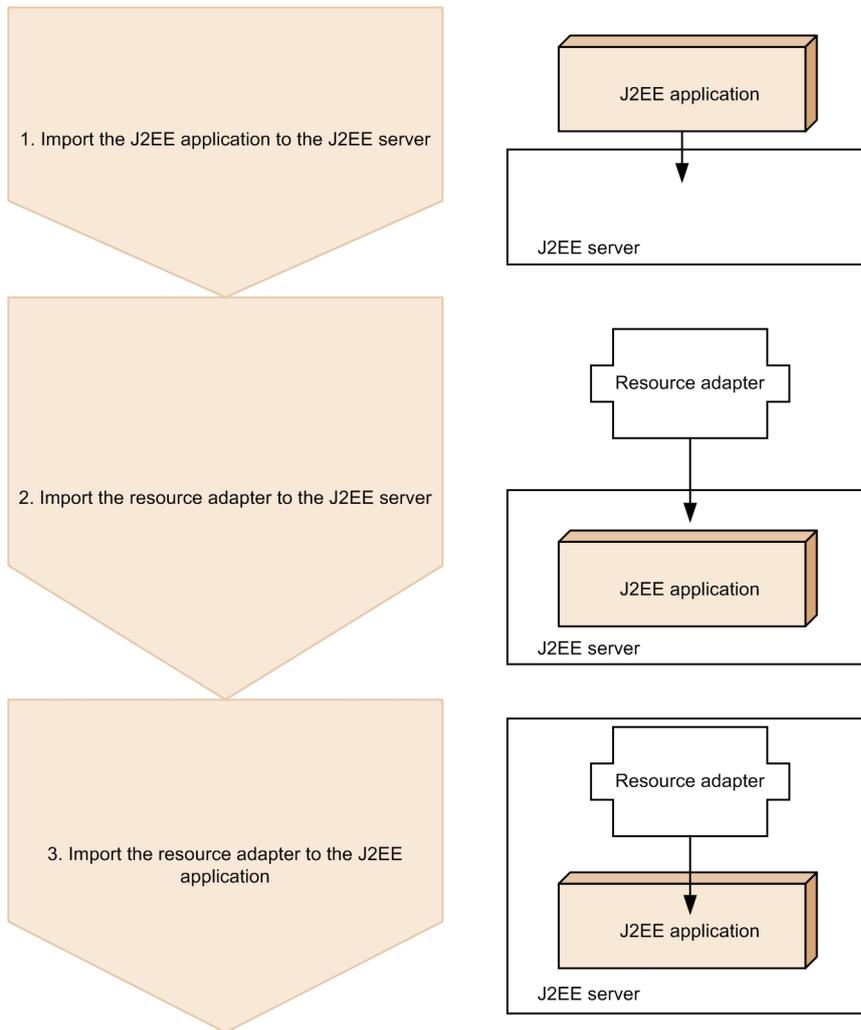
4. Use the server management commands to implement the resource adapter connection test.

Use the `cjtestres` command to implement the resource adapter connection test. For details on the content to be verified in the connection test for each resource, see *3.18 Connection test for resources*.

(2) Method of adding the resource adapter in an imported J2EE application

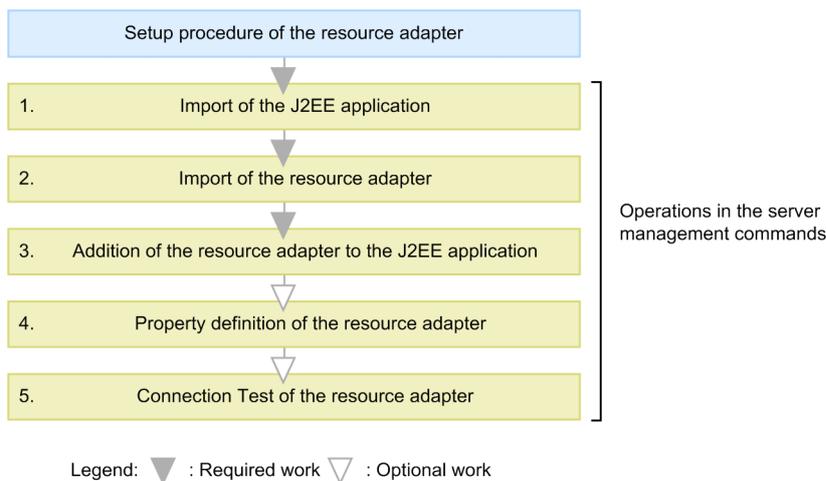
In this method, you import the resource adapter (RAR file) into the J2EE server and then add the resource adapter in a J2EE application that is already imported into the same J2EE server. The following figure gives an overview of this method.

Figure 3-8: Overview of the method of adding the resource adapter in an imported J2EE application



The following figure shows the procedure for adding the resource adapter into an imported J2EE application.

Figure 3-9: Procedure for adding the resource adapter into an imported J2EE application



A description of points 1 to 5 in the figure is as follows:

1. Use the server management commands to import the J2EE application.
Use the `cjimportapp` command to import the J2EE application.
2. Use the server management commands to import the resource adapter.
Use the `cjimportres` command to import the resource adapter.
3. Use the server management commands to add the resource adapter into the J2EE application.
Use the `cjaddapp` command to add the resource adapter into the J2EE application.
4. Use the server management commands to define the resource adapter properties.
Use the `cjgetappprop` command to obtain the HITACHI Connector Property file, edit the file, and then use the `cjsetappprop` command to apply the edited content.
For details on the resource adapter properties specified for the functionality to be used, see the following locations:
 - Settings for using the resource connection and the transaction management functionality
3.4.12 Settings in the execution environment
 - Functionality for performance tuning
3.14.10 Settings in the execution environment
 - Functionality for fault tolerance
3.15.13 Settings in the execution environment
 - Setting the optional names for J2EE resources
2.6.6 Setting the optional names for the J2EE resources
5. Use the server management commands to implement the resource adapter connection test.
Use the `cjtestres` command to implement the resource adapter connection test. For details on the content to be verified in the connection test for each resource, see
3.18 Connection test for resources.

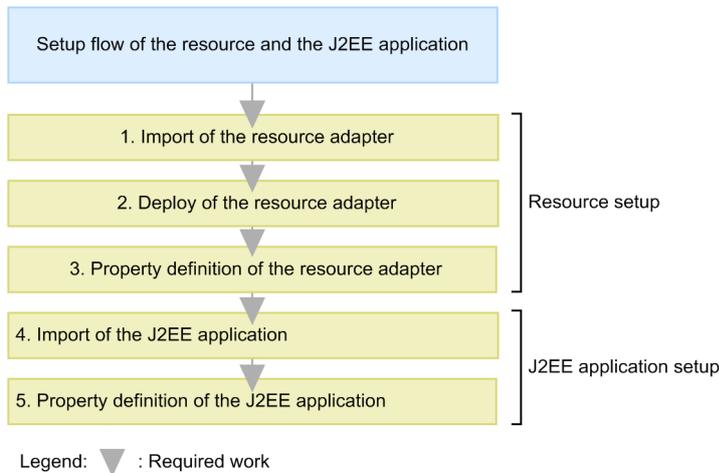
3.3.10 Procedure for resource adapter settings (To use the resource adapter with Inbound)

This subsection describes the procedure for the resource adapter and J2EE application settings for executing message inflow. The procedure for the settings varies depending on whether you directly deploy the resource adapter on the J2EE server, or whether you include the resource adapter in a J2EE application.

(1) Directly deploying the resource adapter on the J2EE server

The following figure shows the procedure for specifying the settings for directly deploying the resource adapter on the J2EE server.

Figure 3-10: Procedure for specifying the settings for directly deploying the resource adapter on the J2EE server



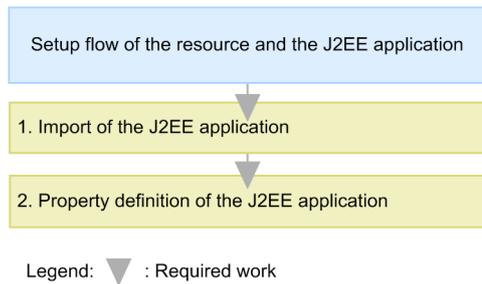
A description of points 1 to 5 in the figure is as follows. The server management command-based operations are described here.

1. Import the resource adapters conforming to the Connector 1.5 specifications.
Execute the `cjimportres` command by specifying `-type rar`.
2. Deploy the resource adapter.
Execute the `cjdeployrar` command.
3. Define the resource adapter properties.
Use the `cjgetrarprop` command to obtain the HITACHI Connector Property file. Edit the file, and then use the `cjsetrarprop` command to apply the edited content.
The Administered objects will be set up here. For details on the settings, see *3.16.8(2) Setting up the Administered objects*.
4. Import the J2EE application containing the Message-driven Bean.
Use the `cjimportapp` command.
5. Define the J2EE application properties.
Execute the `cjgetappprop` command by specifying `-type all` to obtain HITACHI Application Integrated Property File. Edit the file, and then execute the `cjsetappprop` command by specifying `-type all` to apply the edited content.
The following items will be set up here:
 - Mapping the Message-driven Beans and resource adapters
For the settings, see *3.16.8(3) Settings for mapping the Message-driven Beans and resource adapters*.
 - Interfaces used by the Message-driven Beans
For the settings, see *3.16.8(4) Setting up the interfaces used by the Message-driven Beans*.
 - `ActivationSpec` settings
For the settings, see *3.16.8(5) ActivationSpec settings*.

(2) To include and use the resource adapter in the J2EE application

The following figure shows the procedure for specifying the settings to include and use the resource adapter in the J2EE application.

Figure 3-11: Procedure for specifying the settings for including and using the resource adapter in the J2EE application



A description of points 1 and 2 in the figure is as follows. Note that the server management command-based operations are described here.

1. Import the J2EE application containing the Message-driven Beans.
Use the `cjimportapp` command.
2. Define the J2EE application properties.
Execute the `cjgetappprop` command by specifying `-type all` to obtain HITACHI Application Integrated Property File. Edit the file, and then execute the `cjsetappprop` command by specifying `-type all` to apply the edited content.

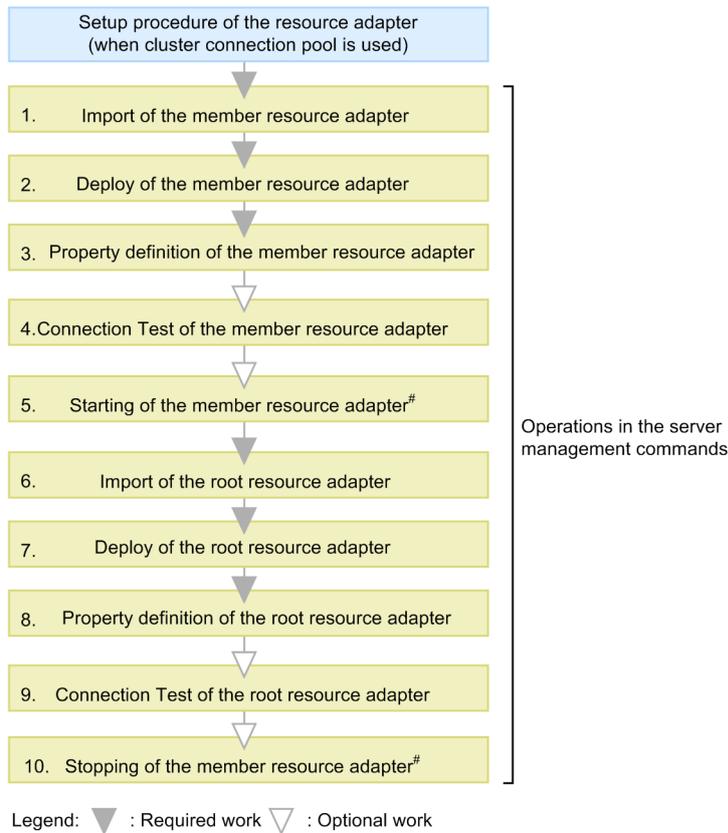
The following items will be set up here:

- Information on the Administered objects
For the settings, see *3.16.8(2) Setting up the Administered objects*.
- Mapping the Message-driven Beans and resource adapters
For the settings, see *3.16.8(3) Settings for mapping the Message-driven Beans and resource adapters*.
- Interfaces used by the Message-driven Beans
For the settings, see *3.16.8(4) Setting up the interfaces used by the Message-driven Beans*.
- ActivationSpec settings
For the settings, see *3.16.8(5) ActivationSpec settings*.

3.3.11 Procedure for resource adapter settings (To use connection pool clustering)

The following figure shows the procedure for resource adapter settings for connecting to a database when the connection pool is clustered.

Figure 3-12: Procedure for resource adapter settings in connection pool clustering



#: This is the work necessary when executing Connection Test of the root resource adapter.

A description of points 1 to 10 in the figure is as follows:

1. Use the server management commands to import the member resource adapters.
Use the `cjimportres` command to import the member resource adapters.
For details on the resource adapters to be imported, see 3.3.2 *Types of resource adapters*.
2. Use the server management commands to deploy the member resource adapters.
Use the `cjdeployrar` command to deploy the member resource adapters.
3. Use the server management commands to define the member resource adapter properties.
Use the `cjgetrarprop` command to obtain the HITACHI Connector Property file, edit the file, and then use the `cjsetrarprop` command to apply the edited content.
The items that you can set up by defining the member resource adapter and root resource adapter properties differ.
For details on the items defined in the properties that can be set for the member resource adapters and root resource adapter, see 3.17 *Functionality for connection pool clustering*.

For details on the resource adapter properties specified for the functionality to be used, see the following locations:

- Settings for using the resource connection and the transaction management functionality
3.4.12 Settings in the execution environment
- Functionality for performance tuning
3.14.10 Settings in the execution environment
- Functionality for fault tolerance
3.15.13 Settings in the execution environment
- Setting the optional names for J2EE resources
2.6.6 Setting the optional names for the J2EE resources

4. Use the server management commands to implement the connection test for the member resource adapters.
Use the `cjtestres` command to implement the connection test for the member resource adapters.
For details on the content to be verified in the connection test for the member resource adapters, see 3.18 *Connection test for resources*.
The steps from 1 to 4 are repeated for the number of member resource adapters only.
5. Use the server management commands to start the member resource adapters.
To implement the connection test of the root resource adapter, start the member resource adapters first. Use the `cjstartrar` command to start the member resource adapters.
6. Use the server management commands to import the root resource adapter.
Use the `cjimportres` command to import the root resource adapter.
For details on the resource adapters to be imported, see 3.3.2 *Types of resource adapters*.
7. Use the server management commands to deploy the root resource adapter.
Use the `cjdeployrar` command to deploy the root resource adapter.
8. Use the server management commands to define the root resource adapter properties.
Use the `cjgetrarprop` command to obtain the HITACHI Connector Property file, edit the file, and then use the `cjsetrarprop` command to apply the edited content.
9. Use the server management commands to implement the connection test for the root resource adapter.
Use the `cjtestres` command to implement the connection test for the root resource adapter.
For details on the content to be verified in the connection test for the root resource adapter, see 3.18 *Connection test for resources*.
10. Use the server management commands to stop the member resource adapters.
After you implement the connection test for the root resource adapter, stop the member resource adapter. Use the `cjstoprar` command to stop the member resource adapters.

For details on the operations with the server management commands, see 3. *Basic Operations of the Server Management Commands* in the *uCosminexus Application Server Application Setup Guide*. Also, for details on the commands, see 2.4 *Resource operation commands used with the J2EE server* in the *uCosminexus Application Server Command Reference Guide*. For details on the property files, see 4. *Property Files Used for Setting Up the Resources* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

Important note

To cluster a connection pool, you must resolve the references from the J2EE application to the root resource adapter. When you use the root resource adapter to define the J2EE application properties, resolve the references from the J2EE application to the root resource adapter.

3.3.12 Connection settings for using components other than the resource adapters

To specify the settings for using the JavaBeans resources and JavaMail, apart from the resource adapters, see the following locations:

- To use the JavaBeans resources
See 3.12.4 *Setting up the JavaBeans resources*.
- To use JavaMail
The mail configuration is only used for the SMTP server connection.
Use the POP3 server to receive mails. The IMAP server cannot be used.
For details, see 3.11 *Connecting to the SMTP server*.

3.3.13 Notes on the resource adapters

This subsection describes the notes on the resource adapters.

Functionality that is unavailable based on the deploy method

When you include and use the resource adapter in the J2EE application, the following functionality is not enabled even if the resources included in the J2EE application are updated:

- Detecting the J2EE application updates
- Reloading the J2EE applications

For details, see *13.8.13 Notes and restrictions related to reloading*.

Notes on the display names of the resource adapters

The EJBs and WARs in the J2EE application can concurrently use the resource adapters that are deployed as J2EE resource adapters and the resource adapters that are included in the J2EE application. However, two or more resource adapters with the same display name cannot be used on one J2EE server. If you try to use resource adapters with the same display name on one J2EE server, an error message is displayed and the resource adapter fails to start. The examples of procedures when the resource adapter fails to start are as follows:

Example 1.

1. A resource adapter with the display name 'Rar1' is deployed on the J2EE server.
2. A J2EE application containing a resource adapter with the display name 'Rar1' is imported.

Example 2.

1. A resource adapter with the display name 'Rar2' is deployed on the J2EE server.
2. A resource adapter with the display name 'Rar2' is added to the imported J2EE application.

Example 3.

1. A J2EE application containing a resource adapter with the display name 'Rar3' is imported.
2. A resource adapter with the display name 'Rar3' is deployed on the J2EE server.

Notes on the optional names for resource adapters

If multiple resource adapters are deployed with the same optional name, an error message is displayed and the resource adapter fails to start.

Notes on the start processing of the resource adapters

If the start processing of a resource adapter fails while a J2EE server is being started, the resource adapter status changes from Start to Stop. In this case, the resource adapter does not start even when you start the J2EE server the next time. The application that is using the applicable resource adapter also fails to start.

In such cases, see the error information, eliminate the cause of the error, and then take the following action:

If the transaction support level of the resource adapter is `LocalTransaction` or `NoTransaction`

1. Start the resource adapter that changed to Stop status.
2. Start the application that is using the resource adapter.

If the transaction support level of the resource adapter is `XATransaction`

1. Start the resource adapter that changed to Stop status.
2. Restart the J2EE server to recover the uncompleted transactions.
3. Start the application that is using the resource adapter.

Other notes

The functionality for referencing the URL connection `Resource Factory` is not supported.

3.4 Managing transactions

This section gives an overview of transaction management.

The transaction management methods used for resource connections include the Application Server-managed method and user-managed method.

When the transactions are managed with Application Server, you can use the transaction manager of Application Server to manage the transactions.

The following table describes the organization of this section.

Table 3-10: Organization of this section (Managing transactions)

Category	Title	Reference location
Explanation	Transaction management methods for the resource connections	3.4.1
	Local transaction and global transaction	3.4.2
	Transaction types available for each resource	3.4.3
	Functionality provided with the transaction services	3.4.4
	Transaction operations during system exceptions	3.4.5
	Obtaining the transaction manager	3.4.6
Implementation [#]	Overview of processing and the points to remember when using the container-managed transactions (CMT)	3.4.7
	Overview of processing and the points to remember when using the <code>UserTransaction</code> interface	3.4.8
	Overview of processing and the points to remember when using the resource adapter-specific transaction management interface	3.4.9
	Overview of processing and the points to remember when transactions are not used	3.4.10
	Notes on the JTA-based transaction implementation	3.4.11
Settings	Settings in the execution environment	3.4.12

Note: The functionality-specific explanation is not available for "Operations".

[#]: For details on the implementation of transactions with the EJB client, see 3.5 *Implementing transactions with the EJB client applications* in the *uCosminexus Application Server EJB Container Functionality Guide*.

Tip

From among the resources, the transactions are not managed for the SMTP server and JavaBeans resources.

Reference note

You can also start a transaction with the EJB client application. For the notes on starting transactions with the EJB client applications, see 3.20 *Notes on starting transactions with the EJB client applications*.

3.4.1 Transaction management methods for the resource connections

The transaction management methods for the resource connections include the Application Server-managed method and non-Application Server-managed method (user-managed method). This subsection describes each transaction management method.

(1) Application Server-managed transactions

In this method, transactions are managed through the Application Server transaction manager. The user manages the transaction either by operating the APIs of the `javax.transaction.UserTransaction` interface or by specifying the CMT attributes of the EJB method.

- **Management by the UserTransaction interface**

You can manage the transactions by issuing the APIs of the `javax.transaction.UserTransaction` interface from the servlets, JSPs, or EJBs (BMT). For details on the BMT, see 2.7.2 *BMT* in the *uCosminexus Application Server EJB Container Functionality Guide*.

- **Management by the CMT attributes of the EJB**

You can manage the transactions using the transaction attributes specified for each EJB (CMT) method. For details on the CMT, see 2.7.3 *CMT* in the *uCosminexus Application Server EJB Container Functionality Guide*.

For Application Server-managed transactions, you can choose a local transaction or global transaction as the transaction type. For the types of Application Server-managed transactions, see 3.4.2 *Local transaction and global transaction*.

(2) User-managed transactions (Transactions not managed by Application Server)

In this method, the user directly manages the transactions using the resource-specific APIs. For example, when the JDBC interface is used to connect to the database, the user directly operates the APIs such as `setAutoCommit()`, `commit()`, and `rollback()` of the `java.sql.Connection` interface.

3.4.2 Local transaction and global transaction

When you use the Application Server-managed transactions, you integrate the Application Server transaction manager and the resource manager (such as DBMS) that manages the resources to manage the transactions. In this case, you choose either a local transaction or a global transaction as the transaction type.

This subsection describes the local transactions and global transactions.

(1) Local transactions

You use a local transaction when only one resource manages the transactions. When you use a local transaction, the resource manager concludes the transaction.

(2) Global transactions

You use a global transaction when multiple resources manage the transactions. When you use a global transaction, the transaction manager adjusts the multiple resource transactions and concludes the transactions in such a way that the consistency is not lost. The two-phase commit protocol is used to conclude the transactions. Note that when you use a global transaction, the in-process transaction service is used. For details on the in-process transaction service, see 3.14.6 *In-process transaction service*.

The cost of processing a global transaction is comparatively higher, so we recommend that you use a local transaction when the transactions are managed by a single resource only.

Note that the light transaction functionality[#] is enabled by default, so you cannot use a global transaction. To use a global transaction, you must disable the light transaction functionality.

#

For details on the light transactions, see 3.14.5 *Light transactions*.

(3) Relationship between the light transaction functionality and transaction management methods

The **light transaction functionality** provides an optimal environment for the local transactions.

The following table describes the mapping between the transaction management methods and light transactions.

Table 3-11: Mapping the transaction management methods and light transactions

Transaction management method		Light transaction enabled	Light transaction disabled
Application Server-managed transactions	Local transaction	Y	Y
	Global transaction	N	Y
Non-Application Server-managed transactions		Y	Y

Legend:

Y: Available

N: Not available

For details on the light transaction functionality, see *3.14.5 Light transactions*.

3.4.3 Transaction types available for each resource

This subsection describes the transaction types available for the following resources:

- Database (Connection method: DB Connector)
- Database queue (Connection method: DB Connector for Cosminexus RM and Cosminexus RM)
- OpenTP1 (Outbound connection) (Connection method: uCosminexus TP1 Connector or TP1/Message Queue - Access)
- OpenTP1 (Inbound connection) (Connection method: TP1 inbound adapter)
- CJMSP Broker (Connection method: CJMSP resource adapter)
- Other resources (Connection method: Resource adapters conforming to the Connector 1.5 specifications)

The transaction types available for each resource are determined by the settings in the following items:

Transaction support levels specified for each resource adapter

Different transaction types are available for each of the following three transaction support levels:

- `NoTransaction`
The resource transactions are not managed.
- `LocalTransaction`
The resource transactions are managed using a local transaction.
- `XATransaction`
The resource transactions are managed using a global transaction.

Note that the settings for the transaction support levels are specified as the resource adapter properties. For details on the resource adapter settings, see *3.4.12 Settings in the execution environment*.

Enabling and disabling of the light transaction functionality

The available transaction types differ based on whether the light transaction functionality is enabled or disabled.

(1) For database connections

The following table describes the transaction types determined by the mapping between the connection methods and transaction support levels.

Table 3-12: Available transaction types (for database connections)

Connection method	Transaction support level	Light transaction functionality	
		Enabled	Disabled
DB Connector (DBConnector_HiRDB_Type4_CP.rar) (DBConnector_Oracle_CP.rar)	NoTransaction	No	No
	LocalTransaction	Local	Local

3. Resource Connections and Transaction Management

Connection method	Transaction support level	Light transaction functionality	
		Enabled	Disabled
(DBConnector_SQLServer_CP.rar) (DBConnector_CP_ClusterPool_Root.rar) (DBConnector_Oracle_CP_ClusterPool_Member.rar)	LocalTransaction	Local	Local
DB Connector (DBConnector_HiRDB_Type4_XA.rar) (DBConnector_Oracle_XA.rar)	XATransaction	--	Global

Legend:

- Global: Global transaction
- Local: Local transaction
- No: Transactions are not managed
- : Cannot be specified

(2) For connecting to a database queue

The following table describes the transaction types determined by the mapping between the connection methods and transaction support levels.

Table 3-13: Available transaction types (for connecting to a database queue)

Connection method	Transaction support level	Light transaction functionality	
		Enabled	Disabled
DB Connector for Cosminexus RM and Cosminexus RM (DBConnector_HiRDB_Type4_CP_Cosminexus_RM.rar) (DBConnector_Oracle_CP_Cosminexus_RM.rar)	NoTransaction LocalTransaction	No Local	No Local
DB Connector for Cosminexus RM and Cosminexus RM (DBConnector_HiRDB_Type4_XA_Cosminexus_RM.rar) (DBConnector_Oracle_XA_Cosminexus_RM.rar)	XATransaction	--	Global

Legend:

- Global: Global transaction
- Local: Local transaction
- No: Transactions are not managed
- : Cannot be specified

(3) For OpenTP1 connections (Outbound connection)

The following table describes the transaction types determined by the mapping between the connection methods and transaction support levels.

Table 3-14: Available transaction types (For OpenTP1 connections (Outbound connection))

Connection method	Transaction support level	Light transaction functionality	
		Enabled	Disabled
uCosminexus TPI Connector	NoTransaction	No	No
	LocalTransaction	Local	Local
	XATransaction	--	Global
TPI/Message Queue - Access	NoTransaction	No	No
	LocalTransaction	Local	Local
	XATransaction	--	Global

Legend:

Global: Global transaction

Local: Local transaction

No: Transactions are not managed

--: Cannot be specified

(4) For OpenTP1 connections (Inbound connection)

For details on the transaction types available for the TPI inbound adapter, see 3.13.2 *Functionality available for other resource connections*.

(5) For CJMSP Broker connections

The following table describes the transaction types determined by the mapping between the connection methods and transaction support levels.

Table 3-15: Available transaction types (For CJMSP Broker connections)

Connection method	Transaction support level	Light transaction functionality	
		Enabled	Disabled
CJMSP resource adapter	NoTransaction	No	No
	LocalTransaction	Local	Local
	XATransaction	--	--

Legend:

Local: Local transaction

No: Transactions are not managed

--: Cannot be specified

(6) For the other resources

For details on the transaction types available for the resource adapters conforming to the Connector 1.5 specifications, see 3.13.2 *Functionality available for other resource connections*.

3.4.4 Functionality provided with the transaction services

The transaction service provides the following functionality. Note that with Application Server, a transaction service is started as an in-process of the J2EE server.

- **Controlling the start, commit conclusion, and rollback conclusion of global transactions**

This functionality controls the transaction processing using the two-phase commit protocol. The two-phase commit protocol is a method that separates the processing at the synchronization points into two stages, the prepare processing (resource update preparation) and the commit processing (resource update processing). With the two-phase commit protocol, you can synchronize, commit, or roll back multiple resource objects such as DBMS. Also, even if an error occurs during the transaction processing, you can consistently and automatically roll back all the resource objects with the two-phase commit protocol.

- **Propagating the transaction context (Invoking Enterprise Beans using the RMI-IIOP)**

This functionality manages the transaction context that indicates the state of the global transaction. For example, when a client, such as a servlet/ JSP, invokes the remote Enterprise Bean method, this functionality transmits the client-side transaction context to the server-side Enterprise Bean.

- **System recovery by managing the transaction information using the status files and restarting the J2EE server after an error occurs**

When the J2EE server stops due to a system error, this functionality recovers the transaction processing of the application programs that were running, and rolls back or commits the transaction processing. Whether the transaction is rolled back or committed is determined based on the extent to which the transaction processing has progressed. From the two-phase commit, if the transaction processing has progressed until the pre-completion of the first phase, the global transaction is rolled back. If the first phase from the two-phase commit is complete, the global transaction is rolled back or committed according to the decision for the root transaction branch.

- **Duplicating the status file**

When you use the status file duplication functionality, if an error occurs on the disk where one status file is allocated, the transaction is recovered using the other status file #. However, if you use this functionality, the disk is accessed twice, so the online processing response time is delayed.

#

The online processing cannot be continued.

- **Reporting errors during heuristic conclusion in the resource manager**

When a heuristic conclusion is detected in the resource manager for resources such as databases, this functionality reports the error through messages.

3.4.5 Transaction operations during system exceptions

During an EJB invocation, the behavior of the invocation source transaction when a system exception occurs at the invocation destination varies as follows depending on the system definition.

(1) When the invocation destination inherits the invocation source transaction (when the transaction attributes of the invocation destination are Required, Supports, and Mandatory of the CMT)

If a system exception is returned at the invocation destination, the transaction is rolled back by the container. This operation is defined in the EJB specifications.

(2) When the invocation destination does not inherit the invocation source transaction (when the transaction attributes of the invocation destination are BMT, or NotSupported, RequiresNew, and Never of the CMT)

The invocation source and invocation destination transactions respectively operate as follows:

Invocation source transaction

- During remote invocation in the EJB remote interface
 - When the light transaction is disabled:
 - The OTS marks the transaction for roll back.
 - When the light transaction is enabled:
 - The transaction is not marked for roll back.
- During local call optimization in the EJB remote interface

The operation differs based on the value of the `ejbserver.distributedtx.rollbackClientTxOnSystemException` key in `usrconf.properties`.

If `true`:

The transaction is marked for roll back.

If `false`:

The transaction is not marked for roll back.

- During the invocation of the EJB local interface
The transaction is marked for roll back.

Invocation destination transaction

The transaction is rolled back by the container. This operation is defined in the EJB specifications.

For details on the local call optimization, see *2.13.1 Local call optimization in the EJB remote interface* in the *uCosminexus Application Server EJB Container Functionality Guide*.

3.4.6 Obtaining the transaction manager

The transaction manager (`javax.transaction.TransactionManager` or `javax.transaction.Transaction`) provides APIs for managing transactions. To use the framework that uses the transaction manager APIs, you can use the JNDI to obtain the transaction manager. To obtain the transaction manager, look up with the name `java:comp/TransactionManager`.

This subsection describes the transaction manager APIs supported by Application Server, and the notes on using Synchronization.

(1) Supported APIs

The following table describes the transaction manager APIs supported by Application Server.

Table 3-16: Transaction manager APIs supported by Application Server

Interface	Method	Availability
<code>javax.transaction.TransactionManager</code>	<code>begin</code>	Y
	<code>commit</code>	Y
	<code>getStatus</code>	Y
	<code>getTransaction</code>	Y
	<code>resume</code>	Y
	<code>rollback</code>	Y
	<code>setRollbackOnly</code>	Y
	<code>setTransactionTimeout</code>	Y
	<code>suspend</code>	Y
<code>javax.transaction.Transaction</code>	<code>commit</code>	Y
	<code>delistResource</code>	--
	<code>enlistResource</code>	--
	<code>getStatus</code>	Y
	<code>registerSynchronization</code>	Y
	<code>rollback</code>	Y

Interface	Method	Availability
<code>javax.transaction.Transaction</code>	<code>setRollbackOnly</code>	Y

Legend:

Y: Available

--: Not available

Note: If you try to use an unavailable method, `javax.transaction.SystemException` is thrown.

(2) Notes on using the transaction manager

- To use the transaction manager obtained through lookup, implement the processing included in transaction management (such as obtaining, using, and releasing the database connections) in the following range—after the transaction starts and before the transaction concludes, or after the transaction starts and before the transaction is suspended, and after the transaction is restarted and before the transaction concludes. The processing that is implemented before the transaction starts, after the transaction concludes, and while the transaction is suspended, is not included in the transaction management.
- Do not use `UserTransaction` with the components that use the transaction manager to control transactions.
- A transaction started with the `begin` method of the transaction manager (`javax.transaction.TransactionManager`) times out at a specified time if the timeout value is specified with the `setTransactionTimeout` method before the `begin` method is invoked. If the `setTransactionTimeout` method is not invoked, the transaction times out at the value (default 180 seconds) specified in the `ejbserver.jta.TransactionManager.defaultTimeOut` property of the logical J2EE server in the Easy Setup Definition file.

(3) Notes on using Synchronization

You cannot use the services provided by the J2EE server with the `beforeCompletion` method and `afterCompletion` method of `Synchronization` (`javax.transaction.Synchronization`) registered using the `registerSynchronization` method of the transaction (`javax.transaction.Transaction`). The examples of services that are not available are as follows:

- Resource access
- Transaction operations using `UserTransaction` or CMT
- EJB access
- JNDI access

From among these services, when resource access is performed, some of the transactions managed by the transaction manager are not managed when the resources are accessed, so a mismatch might occur. To access the resources, specify settings in any framework so that the framework controls the direct transactions for the resources.

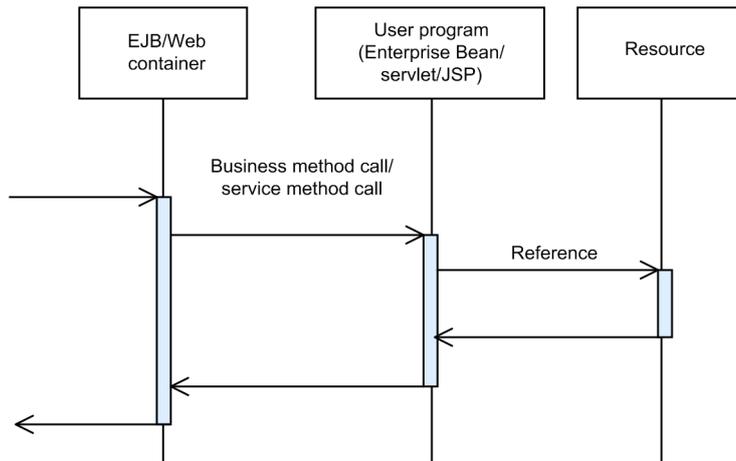
Because the above precautions need to be taken, we do not recommend the use of `Synchronization` in the user programs. If you want to use the transaction conclusion timing from the user program, use `javax.ejb.SessionSynchronization` of the EJB.

3.4.7 Overview of processing and the points to remember when using the container-managed transactions (CMT)

If you use a container-managed transaction, you can automatically start the transaction just before the Enterprise Bean business method is invoked and automatically commit the transaction immediately after the processing of the business method ends. The transaction management processing need not be coded as a user program coding at all, and you can easily manage the transactions for accessing the resources.

The following figure shows the sequence for using the container-managed transactions.

Figure 3-13: Sequence for using the container-managed transactions



Remember the following points when you use the container-managed transactions:

- When you use the container-managed transactions, you can specify the transaction attributes for each Enterprise Bean method. You can specify either `Required`, `RequiresNew`, `Mandatory`, `Supports`, `NotSupported`, or `Never`. To use the transaction, specify `Container` in the `<transaction-type>` tag of the DD, and specify the transaction attributes for each method in the `<trans-attribute>` tag. You can also specify the definition in an annotation without using the DD. For details on the transaction attributes, see 2.7.3 CMT in the *uCosminexus Application Server EJB Container Functionality Guide*. For details on the annotations, see 2. Annotations and Dependency Injection supported by Application Server in the *uCosminexus Application Server API Reference Guide*.
- When you access the resources in the business method used after the transaction is started, the resource access transactions are automatically managed.
- To access multiple resources after the transaction starts, you must use a resource adapter that supports the global transactions, and set the transaction support level of the resource adapter to `XATransaction`.
- When you use the container-managed transactions, you need not code the processing for transaction management as the user program coding.
- You can use the container-managed transactions with Enterprise Beans. The container-managed transactions cannot be used with servlets and JSPs.

3.4.8 Overview of processing and the points to remember when using the `UserTransaction` interface

If you use the `UserTransaction` interface, you can send the transaction start and conclusion instructions from the user program to the transaction manager. Use this method when you want to minutely control the transaction with the user program.

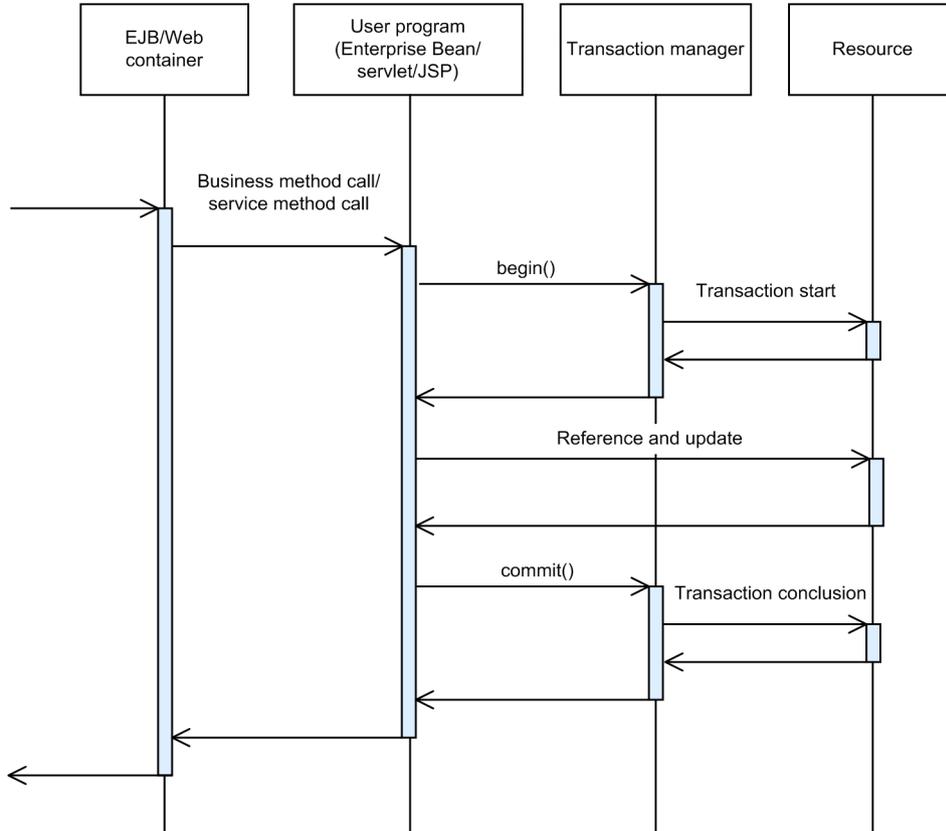
To send the transaction start and conclusion instructions from the user program to the transaction manager:

1. Obtain the `UserTransaction` object.
You obtain the `UserTransaction` object using one of the following methods:
 - Method of using the JNDI from the Naming Service and looking up "java:comp/UserTransaction"
 - Method of obtaining the `UserTransaction` object by invoking the `getUserTransaction` method of the `EJBContext` interface
 - Method of obtaining the `UserTransaction` object by using the DI
2. Invoke the `begin` method of the `UserTransaction` object to start the transaction.
3. Access the resources.

- Invoke the `commit` method or `rollback` method of the `UserTransaction` object to conclude the transaction.

The following figure shows the sequence for using the `UserTransaction` interface.

Figure 3-14: Sequence for using the `UserTransaction` interface



Remember the following points when you use the `UserTransaction` interface:

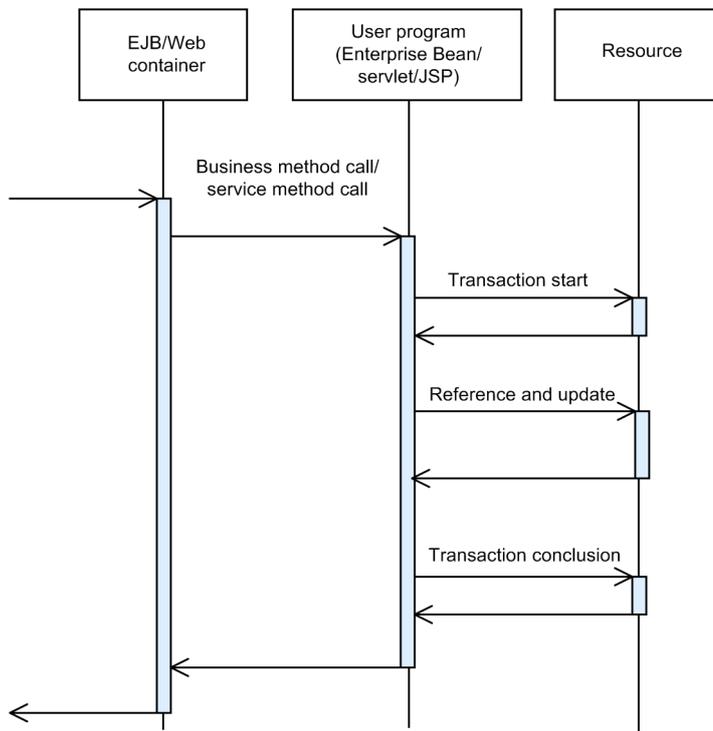
- To use the `UserTransaction` interface, specify `Bean` in the `<transaction-type>` tag of the DD. You can also specify the definition in an annotation without using the DD. For details on annotations, see 2. *Annotations and Dependency Injection supported by Application Server in the uCosminexus Application Server API Reference Guide.*
- When you access the resources after the transaction is started, the resource access transactions are automatically managed.
- To access multiple resources after the transaction starts, you must use a resource adapter that supports the global transactions, and set the transaction support level of the resource adapter to `XATransaction`.
- The `UserTransaction` interface can be used with Enterprise Beans, servlets, and JSPs.
- A transaction that is started with the user program by using the `UserTransaction` interface must be concluded by issuing `commit` or `rollback` with the user program even if an exception occurs. If the transaction is not concluded, problems, such as the resource lock is not released, or the subsequent transactions cannot be started, might occur.

3.4.9 Overview of processing and the points to remember when using the resource adapter-specific transaction management interface

The user program can directly control the resource transactions by using the resource adapter-specific interfaces. For example, if DB Connector exists, the user program can directly control the resource transactions by using the `setAutoCommit` method, `commit` method, and `rollback` method of the `Connection` interface.

The following figure shows the sequence for using the resource adapter-specific transaction management interface.

Figure 3-15: Sequence for using the resource adapter-specific transaction management interface



Remember the following points when you use the resource adapter-specific transaction management interface:

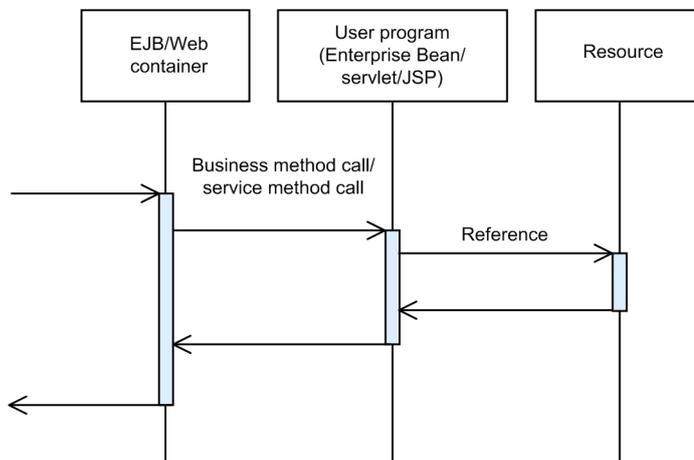
- The functionality provided by transaction manager, such as transaction timeout, cannot be used.
- The transactions for the access of multiple resources cannot be managed.

3.4.10 Overview of processing and the points to remember when transactions are not used

You can also choose to not manage the resource access transactions. You can use this method to reduce the cost of transaction management when the resources are only referenced.

The following figure shows the sequence when transactions are not used.

Figure 3-16: Sequence when transactions are not used



Remember the following points when you do not use transactions:

- Transactions are not used when you specify `Container` in the `<transaction-type>` tag and specify either `NotSupported` or `Never` in the `<trans-attribute>` tag of the Enterprise Bean DD. You can also specify the definition in an annotation without using the DD. For details on the transaction attributes, see 2.7.3 *CMT* in the *uCosminexus Application Server EJB Container Functionality Guide*. For details on annotations, see 2. *Annotations and Dependency Injection supported by Application Server* in the *uCosminexus Application Server API Reference Guide*.
- Transactions are not used when you specify `Bean` in the `<transaction-type>` tag of the Enterprise Bean DD, and invoke the `begin` method of the `UserTransaction` object. You can also specify the definition in an annotation without using the DD. For details on annotations, see 2. *Annotations and Dependency Injection supported by Application Server* in the *uCosminexus Application Server API Reference Guide*.
- Transactions are not used when the `begin` method of the `UserTransaction` object is invoked by servlets and JSPs.
- You can also choose to manage the access to a specific resource adapter without using transactions. To realize this, set the transaction support level of the resource adapter that will not be managed using transactions to `NoTransaction`. The transactions are not managed for a resource adapter with the transaction support level `NoTransaction` even if the resource is accessed after the transaction starts.

3.4.11 Notes on the JTA-based transaction implementation

The following table describes the processing content and operations of the programs that use the JTA to implement transactions.

Table 3–17: JTA operations

Processing content	Operations
<ul style="list-style-type: none"> • When an exception inheriting <code>java.rmi.RemoteException</code> or <code>java.rmi.RemoteException</code> occurs during EJB invocation • When a CORBA system exception occurs 	Whether the client-side transactions will be marked for rollback varies according to the property (<code>ejbserver.distributedtx.rollbackClientTxOnSystemException</code>) settings. For details on the values set for the properties and the operations, see 3.4.5 <i>Transaction operations during system exceptions</i> .
When the <code>javax.transaction.UserTransaction.commit</code> method is invoked after a transaction timeout occurs	The <code>javax.transaction.RollbackException</code> exception occurs. However, if you invoke the <code>UserTransaction.commit</code> method two or more times after a transaction timeout occurs, the <code>java.lang.IllegalStateException</code> exception occurs from the second time onwards.
When the <code>javax.transaction.UserTransaction.rollback</code> method is invoked after a transaction timeout occurs	An exception does not occur.
Return value of the <code>javax.transaction.UserTransaction.getStatus</code> method after the transaction timeout	<code>javax.transaction.Status.STATUS_ROLLEDBACK</code> is returned. However, after the <code>UserTransaction.commit</code> method, or <code>UserTransaction.rollback</code> method is invoked, <code>javax.transaction.Status.STATUS_NO_TRANSACTION</code> is returned.
EJB invocation from the <code>beforeCompletion</code> method or <code>afterCompletion</code> method of the EJB for which <code>javax.ejb.SessionSynchronization</code> is implemented	An exception does not occur and the EJB can be invoked.

3.4.12 Settings in the execution environment

To use the resource connection and transaction management functionality, you must set up the J2EE server and the resource adapters.

(1) J2EE server settings

Specify the J2EE server settings with the Easy Setup definition file. Define the resource connection and the transaction management functionality in the `<configuration>` tag of the logical J2EE server (`j2ee-server`) in the Easy Setup definition file. The following table describes the settings in the Easy Setup definition file. Note that with transaction management you can also specify a transaction timeout.

For details on the Easy Setup Definition file and parameters, see *4.6 Easy Setup Definition file* in the *uCosminexus Application Server Definition Reference Guide*.

For details on the transaction timeout, see *3.15.8 Transaction timeout and statement cancellation*.

Table 3-18: Definition in the Easy Setup definition file for using the resource connection and transaction management functionality

Items	Specified parameters	Settings
Transaction types	<code>ejbserver.distributedtx.XATransaction.enabled</code>	Specifies whether to use the light transaction or the global transaction. By default, the light transaction is enabled.
Client transaction operations when a system exception occurs	<code>ejbserver.distributedtx.rollbackClientTxOnSystemException</code>	Specifies whether to mark the client transaction for rollback when a system exception occurs.
Directory storing the status file	<ul style="list-style-type: none"> <code>ejbserver.distributedtx.ots.status.directory1</code> <code>ejbserver.distributedtx.ots.status.directory2</code> 	Specified as a directory for storing the status file of the in-process transaction service and the status file backup.
Timeout in error detection	<code>ejbserver.connectionpool.validation.timeout</code>	Specifies the timeout value for the connection error detection functionality and the timeout value for the deletion of connections using the connection count adjustment functionality.

! Important note

Specifying the directory for storing the status file of the in-process transaction service and the status file backup

To guarantee transaction consistency, the in-process transaction service imports the host name or IP address as the identity information of the J2EE server into the status file. Therefore, to change the value set in the `vbroker.se.iiop_tp.host` parameter in the J2EE server configuration definition, or to change the IP address of the computer on which the J2EE server is running, without specifying the `vbroker.se.iiop_tp.host` parameter:

1. Stop the J2EE server when there are no transactions on the J2EE server.
2. Change the IP address, or the settings in the `vbroker.se.iiop_tp.host` parameter.
3. Delete the directory specified in the `ejbserver.distributedtx.ots.status.directory1` parameter.
4. Start the J2EE server.

(2) Resource adapter settings

Use the server management commands and property files to specify the resource adapter settings in the execution environment. Define the functionality for managing transactions in the `<resourceadapter>` tag of the HITACHI Connector Property file. The following table describes the settings.

For details on the HITACHI Connector Property file, see *4.1 HITACHI Connector Property file* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

Table 3-19: Definition in the HITACHI Connector Property file for the transaction management functionality

Items	Specified tags	Settings
Transaction support level	<outbound-resourceadapter>-<transaction-support> tag	Specifies the transaction support levels. Specify no transaction management (NoTransaction), local transaction (LocalTransaction), or global transaction (XATransaction).

3.5 Resource sign-on method

You can choose one of the following methods as a resource sign-on method:

- **Container-managed sign-on**

In this method, Application Server is used to automatically sign-on to the resource. In this method, if you set up the user name and password for each resource adapter, Application Server automatically transmits the user name and password to the resource when the connection is obtained.

To use the container-managed sign-on, specify `Container` in the `<res-auth>` tag within the `<resource-ref>` tag of the DD for Enterprise Beans and servlets.

- **Component-managed sign-on**

In this method, the user program is used to sign-on to the resource. To use this method, the user name and password used when the connection is obtained is specified in the user program.

Example: For DB Connector

When you invoke `getConnection` of the `DataSource` class, specify the user name and password in the argument.

To use the component-managed sign-on, specify `Application` in the `<res-auth>` tag within the `<resource-ref>` tag of the DD for Enterprise Beans and servlets.

If connection pooling is used, we recommend that you use the container-managed sign-on that can efficiently re-use connections.

Note that when you use Setup Wizard to set up the system, application authentication cannot be used because `true` is specified in `ejbserver.connectionpool.applicationAuthentication.disabled`.

3.6 Connecting to a database

This section describes the functionality of using DB Connector to connect to a database.

The following table describes the organization of this section.

Table 3-20: Organization of this section (Connecting to a database)

Category	Title	Reference location
Explanation	Overview of connections using DB Connector	3.6.1
	Available J2EE components and functionality	3.6.2
	Connectable databases	3.6.3
	Types of DB Connectors (RAR file)	3.6.4
	Preconditions and notes on connecting to HiRDB	3.6.5
	Preconditions and notes on connecting to Oracle	3.6.6
	Preconditions and notes on connecting to SQL Server	3.6.7
	Preconditions and notes on connecting to XDM/RD E2	3.6.8
Settings	Settings in the execution environment (resource adapter settings)	3.6.9

Note: The functionality-specific explanation is not available for "Implementation" and "Operations".

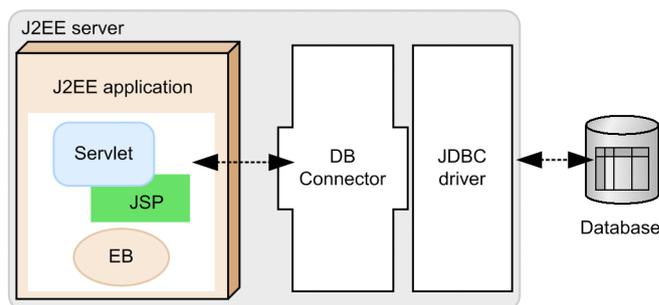
The databases that you can connect to by using DB Connector include HiRDB, Oracle, SQL Server, and XDM/RD E2. When you connect to a database, the information such as the database connection methods and available JDBC drivers differ according to the type of database you want to connect to. This section describes the preconditions for database connections and the available functionality.

3.6.1 Overview of DB Connector-based connections

To connect to a database, you can use DB Connector as the resource adapter. DB Connector is a resource adapter for accessing the database using the JDBC. To use DB Connector, you look up the connection factory (`javax.sql.DataSource` interface) from the J2EE application.

The following figure gives an overview of connections using DB Connector.

Figure 3-17: Overview of connections using DB Connector



Legend: -----> : Flow of data

With the database connections using DB Connector, you use HiRDB Type4 JDBC Driver, Oracle JDBC Thin Driver, or SQL Server as the JDBC drivers.

For details on the DB Connector settings, see 3.6.9 *Settings in the execution environment (resource adapter settings)*, and 4.2 *Settings for connecting to a database* in the *uCosminexus Application Server Application Setup Guide*.

For details on the connection methods for connecting to a database queue, see 3.7 *Connecting to a database queue*.

! Important note

To use a resource adapter, you must resolve the references from the J2EE application to the resource adapter. When you customize a J2EE application that uses the resource adapters, resolve the references from the J2EE application to the resource adapter.

3.6.2 Available J2EE components and functionality

This subsection describes the functionality available for the database connections.

The following table provides a database-wise description of the J2EE components and functionality available for the database connections.

Table 3-21: Available J2EE components and functionality

	Items	HiRDB	Oracle11g	SQL Server	XDM/RD E2
J2EE components	Servlet/JSP	Y	Y	Y	Y
	Stateless Session Bean	Y	Y	Y	Y
	Stateful Session Bean	Y	Y	Y	Y
	Singleton Session Bean	Y	Y	Y	Y
	Entity Bean (BMP)	Y	Y	Y	N
	Entity Bean (CMP 1.1)	Y	N	N	N
	Entity Bean (CMP 2.0)	Y ^{#1}	N	N	N
	Message-driven Bean (database access from the onMessage method)	Y	Y	Y	Y
Available functionality	Connection pooling	Y	Y	Y	Y
	Connection pool warming up	Y	Y	Y	Y
	Connection sharing association	Y	Y	Y	Y
	Caching the DataSource object	Y	R ^{#5}	Y	Y
	Optimizing the container-managed sign-on for DB Connector	Y	Y	Y	Y
	Connection count adjustment functionality	Y	Y	Y	Y
	Connection sweeper functionality	Y	Y	Y	Y
	Displaying the connection pool information (cjlistpool command)	Y	Y	Y	Y
	Clearing the connection pool (cjclearpool command)	Y	Y	Y	Y
	Connection test for resources	Y	Y	Y	Y
	Detecting the connection errors	Y	Y	Y	Y

Items		HiRDB	Oracle11g	SQL Server	XDM/RD E2
Available functionality	Timeout in detecting connection errors	Y	Y	Y	Y
	Waiting to obtain connections during connection depletion	Y	Y	Y	Y
	Retrying a connection	Y	Y	Y	Y
	Auto-closing a connection	Y	Y	Y	Y
	Suspending a connection pool	Y	R ^{#6}	N	Y
	Restarting a connection pool	Y	R ^{#6}	N	Y
	Statement pooling	Y ^{#2}	Y	Y	Y ^{#2}
	Statement cancellation	Y	Y ^{#3}	Y	Y
	Statement <code>setQueryTimeout</code> method	Y	Y ^{#3}	Y	N
	Optional name functionality for J2EE resources	Y	Y	Y	Y
	Functionality for reporting exceptions occurring in Client APIs	Y	Y	Y	Y
	PRF trace output of the connection ID	Y	Y	N	Y
	Output of the SQL statement for troubleshooting	Y	Y	Y	Y
Connection pool clustering	N	Y ^{#4}	N	N	

Legend:

- Y: Available
- N: Not available
- R: Partially restricted

#1: The CMR functionality, which is the CMP 2.0 functionality, is not available.

#2: The HiRDB auto-reconnect functionality and the statement pooling functionality cannot be used together.

#3 Precautions need to be taken when you connect to Oracle. For details, see *3.6.6 Preconditions and notes on connecting to Oracle*.

#4: This functionality is available when the RAC functionality is used and when Oracle JDBC Thin Driver is used for connection.

#5: This functionality is not available when you use a member resource adapter to connect to Oracle11g.

#6: This functionality is available when you use a member resource adapter to connect to Oracle11g.

3.6.3 Connectable databases

This subsection describes the databases that can be connected to by using DB Connector.

(1) Types of connectable databases

The databases that you can connect to by using DB Connector include HiRDB, Oracle, SQL Server, and XDM/RD E2. Note that you cannot use global transactions with SQL Server and XDM/RD E2.

(2) Mapping the databases and JDBC drivers

To connect to a database by using DB Connector, you require a JDBC driver compatible with the database. The following table describes the mapping between the database to be connected to and the available JDBC drivers.

Table 3-22: Database to be connected to and the available JDBC drivers (for database connections)

Database	JDBC driver		
	HiRDB Type4 JDBC Driver	Oracle JDBC Thin Driver	JDBC driver for SQL Server
HiRDB	M	N	N
Oracle	N	M	N
SQL Server	N	N	Y
XDM/RD E2	Y	N	N

Legend:

M: Available and usage is recommended.

Y: Available.

N: Not available.

(3) JDBC specifications supported by DB Connector

The following table describes the JDBC drivers used for connections and the JDBC specifications supported by DB Connector. However, if the JDBC drivers used for connections do not support the functionality provided in the JDBC specifications, that functionality cannot be used with DB Connector.

Table 3-23: JDBC drivers used for connections and the JDBC specifications supported by DB Connector

JDBC drivers used for connections	JDBC specifications supported by DB Connector
HiRDB Type4 JDBC Driver	JDBC 4.0
Oracle JDBC Thin Driver	JDBC 4.0
SQL Server JDBC Driver 3.0	JDBC 3.0
JDBC Driver 4.0 for SQL Server	JDBC 4.0

Tip

Oracle JDBC Thin Driver supports functionality in the range defined in the JDBC 4.0 specifications. The Oracle extension functionality using the classes and interfaces in the Oracle package is not available.

Also, when Application Server connects to a database by using Oracle JDBC Thin Driver, the extension functionality of Oracle JDBC Thin Driver, statement cache and connection cache functionality, cannot be used. Use the statement pooling or connection pooling provided with Application Server. For details on statement pooling or connection pooling, see *3.14 Functionality for performance tuning*.

Note that the `java.sql.Wrapper` interface is not supported. If a method of the `Wrapper` interface is used, the `UnsupportedOperationException` exception is thrown.

! Important note

From among the methods of the `java.sql.Statement` interface in the JDBC specifications, the following methods are not supported:

- `setCursorName(String name)`
- `setFetchDirection(int direction)`

3.6.4 Types of DB Connectors (RAR file)

When you use DB Connector to connect to a database, you use the RAR file supported by the JDBC driver to be used. Use the server management commands to operate the RAR file. For details on how to operate the RAR file by using the server management commands, see the *uCosminexus Application Server Application Setup Guide*.

The following table describes the types of JDBC drivers and the corresponding RAR files.

Table 3-24: Mapping the JDBC drivers and RAR files

JDBC driver	RAR file	Explanation
HiRDB Type4 JDBC Driver	DBConnector_HiRDB_Type4_CP.rar	Use this file when transactions are not managed, or when a local transaction is used.
	DBConnector_HiRDB_Type4_XA.rar [#]	Use this file when a global transaction is used.
Oracle JDBC Thin Driver	DBConnector_Oracle_CP.rar	Use this file when transactions are not managed, or when a local transaction is used.
	DBConnector_Oracle_XA.rar [#]	Use this file when a global transaction is used.
	DBConnector_CP_ClusterPool_Root.rar	Use this file when the root resource adapter is used with the connection pool clustering functionality.
	DBConnector_Oracle_CP_ClusterPool_Member.rar	Use this file when the member resource adapter is used with the connection pool clustering functionality.
JDBC driver for SQL Server	DBConnector_SQLServer_CP.rar	This RAR file is used to connect to SQL Server. Use this file when transactions are not managed, or when a local transaction is used.

[#]: The light transaction functionality cannot be enabled.

3.6.5 Preconditions and notes on connecting to HiRDB

This subsection describes the preconditions and notes on connecting to HiRDB. The available JDBC drivers differ according to the HiRDB version.

(1) Preconditions for HiRDB Version 8

The preconditions for HiRDB Version 8 are as follows:

- **Available JDBC driver**
The available JDBC driver is HiRDB Type4 JDBC Driver.
- **Connection method**
You use one of the following RAR files:
 - DBConnector_HiRDB_Type4_CP.rar
 - DBConnector_HiRDB_Type4_XA.rar

The specifiable transaction support level varies according to the RAR file type. The availability of the light transaction also differs.

The following table describes the transaction support levels that can be specified for each RAR file.

Table 3-25: Transaction support levels available for each RAR file (HiRDB Version 8)

Used DB Connector (RAR file)	Transaction support level	Light transaction	
		Enabled	Disabled
DBConnector_HiRDB_Type4_CP.rar	NoTransaction LocalTransaction	Y	Y
DBConnector_HiRDB_Type4_XA.rar	XATransaction	--	Y

Legend:

Y: Available

--: Not applicable

(2) Preconditions for HiRDB Version 9

The preconditions for HiRDB Version 9 are as follows:

- **Available JDBC driver**

The available JDBC driver is HiRDB Type4 JDBC Driver.

- **Connection method**

You use one of the following RAR files:

- DBConnector_HiRDB_Type4_CP.rar
- DBConnector_HiRDB_Type4_XA.rar

The specifiable transaction support level varies according to the RAR file type. The availability of the light transaction also differs.

The following table describes the transaction support levels that can be specified for each RAR file.

Table 3-26: Transaction support levels available for each RAR file (HiRDB Version 9)

Used DB Connector (RAR file)	transaction support level	Light transaction	
		Enabled	Disabled
DBConnector_HiRDB_Type4_CP.rar	NoTransaction LocalTransaction	Y	Y
DBConnector_HiRDB_Type4_XA.rar	XATransaction	--	Y

Legend:

Y: Available

--: Not applicable

(3) Notes on connecting to HiRDB

Note the following when you connect to HiRDB:

- If you use the HiRDB auto-reconnect functionality, the connection is automatically re-established when the HiRDB server is disconnected due to an error such as the database network error. However, if the connection is lost during a transaction, the `SQLException` exception occurs. When the J2EE application receives the `SQLException` exception, do not continue the processing. If you continue to access DBMS, problems such as data inconsistency might occur.
- If you enable the connection pooling functionality and statement pooling functionality and use `Statement.cancel()`, the `SQLException` exception might occur. In this case, we recommend that you disable either the connection pooling functionality or statement pooling functionality.
- If the J2EE applications satisfy all the following conditions, two connections are used, which is double the number of users using the HiRDB connections concurrently. Specify a value that is twice the number of users using the HiRDB connections concurrently in the `pd_max_users` operand of the HiRDB common system definitions. For details on the `pd_max_users` operand, see the manual *HiRDB Version 9 System Definition*.
 1. DB Connector is used with the transaction support level `XATransaction`.
 2. The database is accessed by using the connection[#] in the Application Server-managed transactions.
 3. Before the transaction in point 2 concludes, the database is accessed by using the connection[#] outside the transaction.

#: This connection is the same connection as the one obtained from DB Connector in point 1.
- An error occurs when the J2EE applications satisfy all the following conditions:
 1. DB Connector is used with the transaction support level `XATransaction`.

2. The database is accessed by using the connection# in the Application Server-managed transactions.
 3. Before the transaction in point 2 concludes, the database is accessed by using the connection# outside the transaction.
 - #: This connection is the same connection as the one obtained from DB Connector in point 1.
- If all the following conditions are satisfied, the same connection cannot participate in multiple different global transactions concurrently. Use a different connection for each transaction.
 1. DB Connector is used with the transaction support level `XATransaction`.
 2. The database is accessed by using the connection# in the Application Server-managed transactions.
 3. Before the transaction in point 2 concludes, the database is accessed by using the connection# outside the transaction.
 - #: This connection is the same connection as the one obtained from DB Connector in point 1.
 - If you enable the HiRDB auto-reconnect functionality and statement pooling functionality, the `SQLException` exception containing message `KFPA11901-E` might occur during the execution of the SQL statement after the auto-reconnect functionality re-establishes the connection. When you connect to HiRDB using DB Connector with the transaction support level `LocalTransaction` or `NoTransaction` and enable the statement pooling functionality, do not use the HiRDB auto-reconnect functionality. When an error occurs, execute the connection error detection functionality or the `cjclearpool` command.

3.6.6 Preconditions and notes on connecting to Oracle

This subsection describes the preconditions and notes on connecting to Oracle.

(1) Preconditions for Oracle11g

The preconditions for Oracle11g are as follows:

- **Available JDBC driver**

The available JDBC driver is Oracle JDBC Thin Driver.

- **Connection method**

You use one of the following RAR files:

- `DBConnector_Oracle_CP.rar`
- `DBConnector_Oracle_XA.rar`
- `DBConnector_CP_ClusterPool_Root.rar`
- `DBConnector_Oracle_CP_ClusterPool_Member.rar`

The specifiable transaction support level varies according to the RAR file type. The availability of the light transaction also differs.

The following table describes the transaction support levels that can be specified for each RAR file.

Table 3-27: Transaction support levels available for each RAR file (Oracle11g)

Used DB Connector (RAR file)	Transaction support level	Light transaction	
		Enabled	Disabled
<code>DBConnector_Oracle_CP.rar</code>	<code>NoTransaction</code> <code>LocalTransaction</code>	Y	Y
<code>DBConnector_Oracle_XA.rar</code>	<code>XATransaction</code>	--	Y
<code>DBConnector_CP_ClusterPool_Root.rar</code> <code>DBConnector_Oracle_CP_ClusterPool_Member.rar</code>	<code>NoTransaction</code> <code>LocalTransaction</code>	Y	Y

Legend:
Y: Available

--: Not applicable

(2) Notes on connecting to Oracle

Note the following when you connect to Oracle:

(a) Differences in the available J2EE components

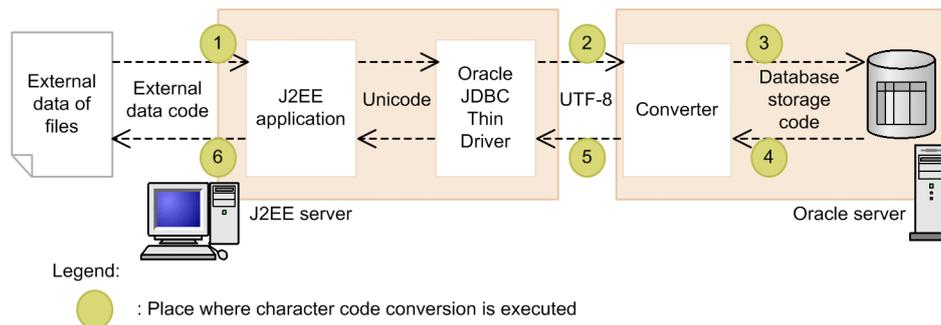
The J2EE components available when you use Oracle JDBC Thin Driver to connect to Oracle are as follows:

- Servlets /JSPs
- Stateless Session Bean
- Stateful Session Bean
- Entity Bean (BMP)
- Entity Bean (CMP2.0)
- Message-driven Bean

(b) Settings for preventing the differences in Japanese character code conversion and garbled characters in Oracle JDBC Thin Driver

When the data is stored in the database and when the data is extracted from the database, the JDBC driver converts the character code between Unicode and the database storage code appropriately. The following figure shows the locations where the character code conversion is implemented when you use Oracle JDBC Thin Driver.

Figure 3-18: Locations where the character code conversion is implemented when Oracle JDBC Thin Driver is used



The following table describes the character code conversion executed at each location in the figure.

Table 3-28: Locations where the character code conversion is implemented when Oracle JDBC Thin Driver is used

Location	Implemented content
1	When the external data, such as a network or file, is read by the J2EE server, the external data code is converted to Unicode.
2	When the data read by the J2EE server is stored in the Oracle server, Unicode is converted to UTF-8. This code conversion is implemented by the JDBC driver part on the J2EE server.
3	The Oracle server converter converts UTF-8 to the database storage code. This code conversion is implemented on the Oracle server.
4	The Oracle server converter converts the database storage code to UTF-8.
5	When the data stored by the J2EE server is obtained from the Oracle server, UTF-8 is converted to Unicode.
6	When the data obtained with the J2EE server is written to a network or a file, Unicode is converted to the external data code.

When you use Oracle JDBC Thin Driver, the characters might get garbled due to the differences in the mapping rules supported by the converters for JavaVM and the Oracle server. The garbled characters occur due to the combination of the external data character code and the database storage code, and the combination of converters that perform the character code conversion.

To avoid garbled characters, use the following combinations of the external data character code and the database storage code:

- **When the external data character code is Shift JIS (CP932)**
Specify "AL32UTF8" or "JA16SJISTILDE" in the database storage code. Note that if you specify "JA16SJIS" in the database storage code, the "~" (swung dash) character gets garbled.
- **When the external data character code is SJIS**
Specify "AL32UTF8" in the database storage code. Note that if you specify "JA16SJIS" in the database storage code, the "[", "]", "/", and "-" characters get garbled.
- **When the external data character code is EUC**
Specify "AL32UTF8" in the database storage code. Note that if you specify "JA16EUC" in the database storage code, the "[", and "]" characters get garbled.

(c) Dedicated server connection

The following functionality is not available for dedicated server connections. To use the following functionality, use a shared server connection. For details, inquire with the Oracle Support Service.

- Statement cancellation
- Query timeout

(d) Return value of the select statement during the conclusion of a JTA transaction

When using Oracle with an XA transaction, if you issue the Oracle `select` statements for multiple databases that were updated in a transaction during the conclusion of a JTA transaction, the return value of each select statement might be different. This is due to the restrictions in the distributed read consistency for Oracle.

(3) Method of connecting to Oracle using Oracle RAC

The method of connecting to Oracle using Oracle RAC varies according to the Oracle version or the functionality used for load balancing. The following table describes the mapping between the Oracle version, the functionality used for load balancing, and the RAR file used.

Table 3-29: Mapping between the Oracle version, functionality used, and the RAR file used

Oracle version	Functionality used for load balancing	DB Connector RAR file name
<ul style="list-style-type: none"> • Oracle10g • Oracle11g 	Application Server functionality (Connection pool clustering functionality)	DBConnector_CP_ClusterPool_Root.rar DBConnector_Oracle_CP_ClusterPool_Member.rar
	Oracle functionality	DBConnector_Oracle_CP.rar DBConnector_Oracle_CP_Cosminexus_RM.rar

3.6.7 Preconditions and notes on connecting to SQL Server

This subsection describes the preconditions and notes on connecting to SQL Server.

(1) Preconditions for SQL Server

The preconditions for SQL Server are as follows:

- **Available JDBC driver**

The available JDBC driver is JDBC driver for SQL Server.

• **Connection method**

Use `DBConnector_SQLServer_CP.rar`. The RAR file to be used varies according to the SQL Server to be connected to.

The following table describes the transaction support levels that can be specified for each RAR file.

Table 3-30: Transaction support levels available for each RAR file (SQL Server)

Used DB Connector (RAR file)	Transaction support level	Light transaction	
		Enabled	Disabled
<code>DBConnector_SQLServer_CP.rar</code>	<code>NoTransaction</code> <code>LocalTransaction</code>	Y	Y

Legend:

Y: Available

Note that you can only connect to SQL Server in Windows.

(2) Notes on connecting to SQL Server

This subsection describes the notes related to the character code conversion in the system and the notes on the DB Connector settings when you connect to SQL Server.

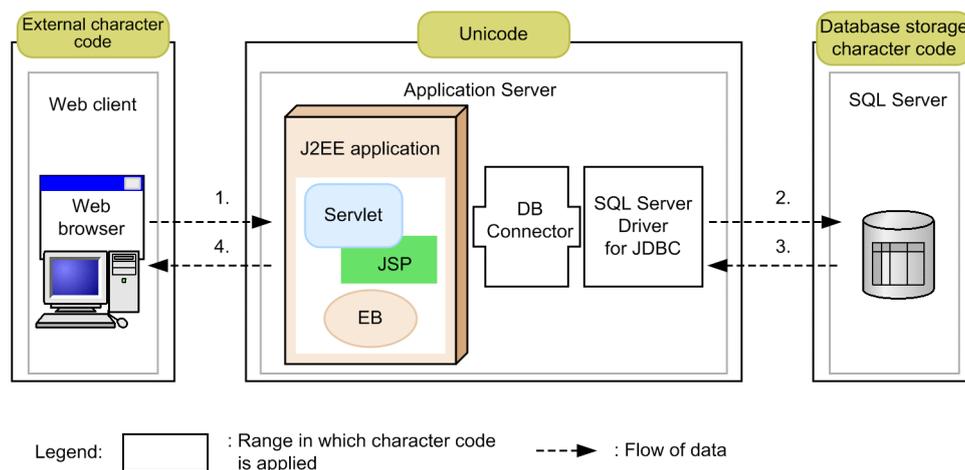
(a) Notes on character code conversion in a system

When you are connected to SQL Server and want to store data containing a Japanese character code in the database, you must take into account the character code conversion in the system. This section gives an overview of the character code conversion in a system, and describes the notes on specifying the settings to avoid garbled characters.

With Java, the Japanese character code is expressed using Unicode. When you use SQL Server, the character code conversion is implemented during the processing between the Web client and Application Server and between Application Server and SQL Server.

The following figure gives an overview of the character code conversion when SQL Server is used.

Figure 3-19: Overview of the character code conversion when SQL Server is used



A description of points 1 to 4 in the figure is as follows:

1. When Application Server receives data from the Web client, the external character code is converted to Unicode.
2. When Application Server stores the data in SQL Server, Unicode is converted to the database storage character code.
3. When the data stored in SQL Server is obtained by Application Server, the database storage character code is converted to Unicode.

4. When Application Server sends the data to the Web client, Unicode is converted to the external character code.

When you use SQL Server, problems such as garbled characters might occur depending on the combination of the external character code and database storage character code, and the types of converter used for implementing the character code conversion. To avoid garbling, you must note the character code settings.

SQL Server supports the character data types described in the following table. When you use SQL Server, you can prevent the occurrence of garbling during the character code conversion by using the Unicode data types.

Table 3-31: Character data types supported by SQL Server

Category	Character data type
Unicode data types	nchar, nvarchar, ntext
Non-Unicode character data types	char, varchar, text

The following paragraphs describe the character code conversion when you use the Unicode data types and when you use the non-Unicode data types as the database storage character code in SQL Server:

When you use the Unicode data types

If `true` (default value) is set in the `sendStringParametersAsUnicode` key of the DB Connector property, garbling does not occur. If `false` is set, and if you use Shift_JIS, EUC-JP, ISO-2022-JP, or UTF-8 in the external character code, garbling might occur.

When you use the non-Unicode character data types

Garbling might occur depending on the external character code settings.

- When Windows-31J is used in the external character code
Garbling does not occur.
- When Shift_JIS, EUC-JP, or ISO-2022-JP is used in the external character code
The characters such as those shown below might be garbled.

「φ」 「ε」 「¬」 「||」 「-」 「~」

- When UTF-8 is used in the external character code
The characters such as those shown below might be garbled.

「φ」 「ε」 「¬」 「𐀀」 「¡」 「~」 「-」 「~」

(b) Notes on specifying the `selectMethod` property of DB Connector

The following table describes the notes on specifying `direct` in the value of the `selectMethod` property (item name of <config-property-name>) of DB Connector.

Table 3-32: Notes on specifying the `selectMethod` property of DB Connector

Condition	Notes
The connection error detection functionality is enabled	A connection error that occurs might be misdiagnosed as normal. As a result, the connection with an error might be returned to the user application program, so do not use the connection error detection functionality. When an error is detected, execute the <code>cjclearpool</code> command.
Multiple Statement, PreparedStatement and CallableStatement are generated concurrently	The JDBC driver for SQL Server generates a connection to SQL Server for each concurrently generated statement. Also, note that when the statement pooling functionality is used, connections are generated for each pooled statement, and a large amount of memory is consumed.

(3) Notes on connecting to SQL Server 2005

Make sure you use the connection pooling functionality to connect to SQL Server 2005.

The following events might occur depending on the behavior of SQL Server JDBC Driver 3.0:

(a) The database sessions are not disconnected and the unused sessions are left behind.

The database sessions are not disconnected and the un-used sessions are left behind until garbage collection occurs. To avoid this event, execute the `javagc` command.

Note that this event occurs in the following conditions:

When the following operations are performed:

- Connection test of DB Connector
- Stopping the DB Connector
- Execution of the `cjclearpool` command

When the following functionality is executed:

- Method cancellation functionality
Message KDJE31016-W is output.
- Transaction timeout functionality
Message KDJE31002-W is output.
- Connection count adjustment functionality
Message KDJE49532-I is output.
- Connection sweeper functionality
Message KDJE50010-I is output when the managed connections are destroyed, including cases when the connection sweeper functionality is operated by setting the log or trace level output by DB Connector to WARNING or INFORMATION.

(b) An attempt to obtain a connection fails.

When the number of database sessions reaches the maximum concurrent user connections for the database because the above-mentioned (a) database sessions are not disconnected, you can no longer generate new sessions. When you execute the `getConnection` method of the `javax.sql.DataSource` interface from the user program, `java.sql.SQLException` might occur. To avoid this event, specify 0 (unlimited) as the maximum concurrent user connections for the database.

Note that this event occurs when 1 or more is specified as the maximum concurrent user connections for the database.

(4) Notes on connecting to SQL Server 2008 or SQL Server 2012

The notes on connecting to SQL Server 2008 or SQL Server 2012 are as follows. Note that none of the precautions on using SQL Server 2005 are applicable.

- Set the authentication mode of SQL Server 2008 or SQL Server 2012 to the mixed mode (Windows authentication and SQL Server authentication). For details on the authentication mode, see the documentation for SQL Server 2008 or SQL Server 2012.
- You cannot specify both JDBC driver JAR files for SQL Server JDBC Driver 3.0 and JDBC driver JAR files for JDBC Driver 4.0 for SQL Server in the user class path of the J2EE server.
- You cannot specify the following settings with `DBConnector_SQLServer_CP.rar`:

Settings added in SQL Server JDBC Driver 2.0

```
responseBuffering
encrypt
hostNameCertificate
trustServerCertificate
trustStore
trustStorePassword
```

Settings added in SQL Server JDBC Driver 3.0

```
sendTimeAsDatetime
```

Settings added in JDBC Driver 4.0 for SQL Server

```
authenticationScheme
```

Therefore, to operate the default value of `responseBuffering` as `adaptive`, the adaptive buffering functionality is always enabled. For details, see the documentation for SQL Server 2008 or SQL Server 2012.

3.6.8 Preconditions and notes on connecting to XDM/RD E2

This subsection describes the preconditions and notes on connecting to XDM/RD E2.

- **Available JDBC driver**

The available JDBC driver is HiRDB Type4 JDBC Driver.

- **Connection method**

Use `DBConnector_HiRDB_Type4_CP.rar`.

The following table describes the transaction support levels that can be specified for each RAR file.

Table 3-33: Transaction support levels available for each RAR file (XDM/RD E2)

Used DB Connector (RAR file)	Transaction support level	Light transaction	
		Enabled	Disabled
<code>DBConnector_HiRDB_Type4_CP.rar</code>	<code>NoTransaction</code> <code>LocalTransaction</code>	Y	Y

Legend:

Y: Available

3.6.9 Settings in the execution environment (resource adapter settings)

You must set up the property files to use the functionality for connecting to a database.

Use server management commands and property files to specify the resource adapter settings in the execution environment. Use the HITACHI Connector Property file to define the functionality used for connecting to a database.

This subsection describes the common DB Connector settings for connecting to a database regardless of the functionality used.

- **Waiting time until the database connection is established**

Specify the time for which a J2EE application waits until a database connection is established with `loginTimeout` in the `<config-property>` tag.

For details on the database connection settings, see *4.2 Settings for connecting to a database* in the *uCosminexus Application Server Application Setup Guide*.

3.7 Connecting to a database queue

This section describes the functionality of using DB Connector for Cosminexus RM and Cosminexus RM to connect to a database queue.

The following table describes the organization of this section.

Table 3-34: Organization of this section (Connecting to a database queue)

Category	Title	Reference location
Explanation	Overview of connections using DB Connector for Cosminexus RM and Cosminexus RM	3.7.1
	Features of connections using DB Connector for Cosminexus RM and Cosminexus RM	3.7.2
	Available functionality	3.7.3
	Connectable databases	3.7.4
	Types of DB Connector for Cosminexus RM (RAR file)	3.7.5
	Preconditions for connecting to a HiRDB queue	3.7.6
	Preconditions for connecting to an Oracle queue	3.7.7
Settings	Settings for connecting to a database queue	3.7.8

Note: The functionality-specific explanation is not available for "Implementation" and "Operations".

To connect to a database queue, you use DB Connector for Cosminexus RM and Cosminexus RM as the resource adapter.

For details on Cosminexus RM, see the manual *uCosminexus Application Server Reliable Messaging*. Also, for details on the connectable databases, see *3.6 Connecting to a database*.

3.7.1 Overview of connections using DB Connector for Cosminexus RM and Cosminexus RM

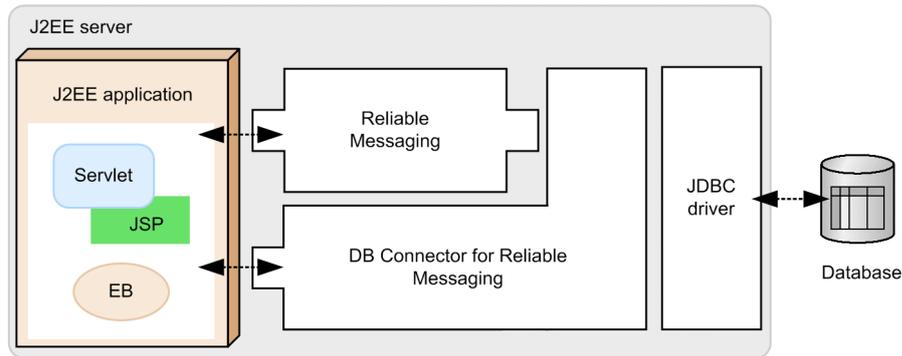
DB Connector for Cosminexus RM is a resource adapter that is integrated with Cosminexus RM and uses the JMS interface to connect to a database.

By integrating with Cosminexus RM, you can use the JMS interface to access the database queue from the servlets, JSPs, and Enterprise Beans (Session Beans, Entity Beans, and Message-driven Beans). You can also use the JDBC interface to access the database tables.

When you use the JMS interface and JDBC interface to access the same database, you can process the global transaction using a single-phase commit. Due to this, the processing performance improves. Also, the JMS interface and JDBC interface can share the physical connection used for the database connection, so the resources can be utilized effectively.

The following figure gives an overview of connections using DB Connector for Cosminexus RM and Cosminexus RM.

Figure 3-20: Overview of connections using DB Connector for Cosminexus RM and Cosminexus RM



Legend: -----> : Flow of data

For database connections through DB Connector for Cosminexus RM and Cosminexus RM, you use HiRDB Type4 JDBC Driver or Oracle JDBC Thin Driver as the JDBC driver.

For details on the settings for DB Connector for Cosminexus RM, see *2.7 Functionality for DB Connector for Cosminexus RM* in the manual *uCosminexus Application Server Reliable Messaging*.

3.7.2 Features of connections using DB Connector for Cosminexus RM and Cosminexus RM

This subsection describes the features of connections using DB Connector for Cosminexus RM and Cosminexus RM.

(1) Overview of processing by DB Connector for Cosminexus RM and Cosminexus RM

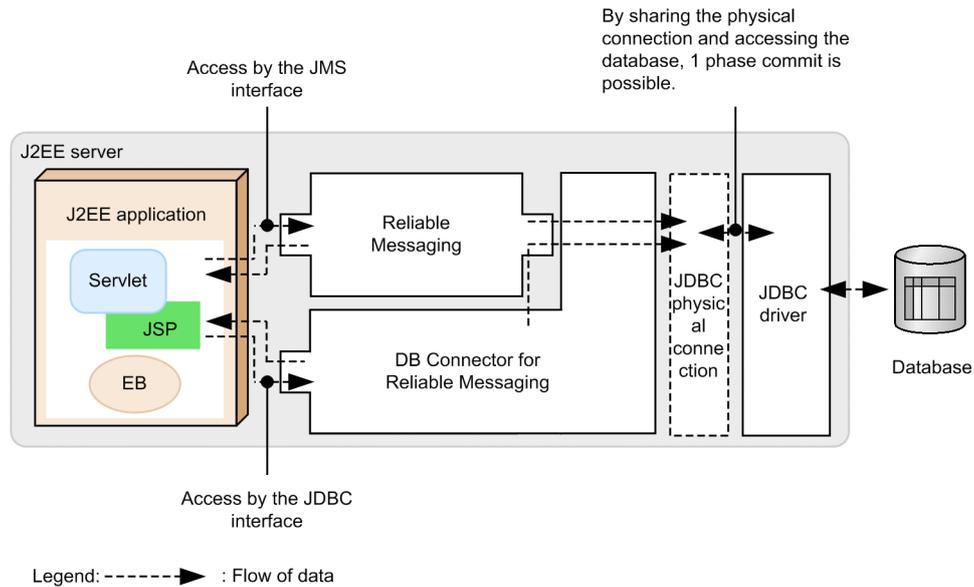
By using DB Connector for Cosminexus RM to integrate with Cosminexus RM, you can send and receive messages using the JMS interface for the database queue, and invoke the Message-driven Beans started by the messages.

When you connect to a database using DB Connector for Cosminexus RM and Cosminexus RM, the following can be realized because the JMS interface and JDBC interface share the physical connection used for access and the processing performance is improved:

- Application of the local transactions
- Single-phase commit for the global transactions

The following figure gives an overview of DB Connector for Cosminexus RM and Cosminexus RM processing when the JMS interface and JDBC interface are used to access the database from servlets, JSPs, or Enterprise Beans.

Figure 3-21: Overview of the processing for DB Connector for Cosminexus RM and Cosminexus RM



The database is accessed via Cosminexus RM using the JMS interface, and the database is accessed via DB Connector for Cosminexus RM using the JDBC interface. Note that to set up a queue on the database, the JDBC interface-based database access is used internally with Cosminexus RM. Therefore, the database is accessed via the JDBC driver.

Furthermore, with DB Connector for Cosminexus RM, the database is accessed via Cosminexus RM, by sharing the physical connection with Cosminexus RM. These resource adapters use the connection pool for Cosminexus RM to obtain the connections. As a result, the sharing of the physical connections is realized.

(2) Configuring the database connections using DB Connector for Cosminexus RM and Cosminexus RM

This subsection describes the resource configuration pattern when DB Connector for Cosminexus RM and Cosminexus RM are used for connection. The resource configurations for the following cases will be described here:

- When the JMS interface is used alone
- When the same database is accessed with the JMS interface and JDBC interface
- When different databases are accessed with the JMS interface and JDBC interface
- When the Message-driven Beans are used

Note that the connections can be shared when the same database is accessed with the JMS interface and JDBC interface and when the Message-driven Beans are used. However, to share a connection, the following preconditions must be satisfied:

Preconditions for connection sharing

- A database queue is accessed using the JMS interface, and a table is accessed using the JDBC interface in the same transaction through the user program.
- The database on which the queue is set by Cosminexus RM and the database that accesses the table are the same.
- The same security information (user name and password) and sign-on method are used for accessing the database queue through the JMS interface and for accessing the table on the database through the JDBC interface.
- The value `Shareable` is set in the `<res-sharing-scope>` tag of the DD for the J2EE application that specifies the reference to DB Connector for Cosminexus RM and Cosminexus RM.

! Important note

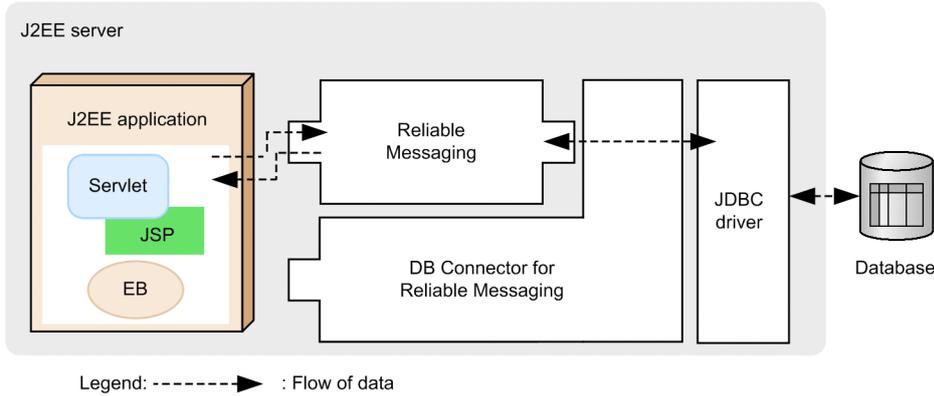
- When you use only the JDBC interface to access the database, use DB Connector as the resource adapter.

- If the preconditions for connection sharing are not satisfied, do not access the database using both JMS interface and JDBC interface through DB Connector for Cosminexus RM and Cosminexus RM. In this case, set up the configuration described in (c) *When different databases are accessed with the JMS interface and JDBC interface*, and use DB Connector to access the tables.

(a) When the JMS interface is used alone

When the database queue alone is accessed with the user program, use the configuration shown in the following figure.

Figure 3-22: Configuration when the JMS interface is used alone

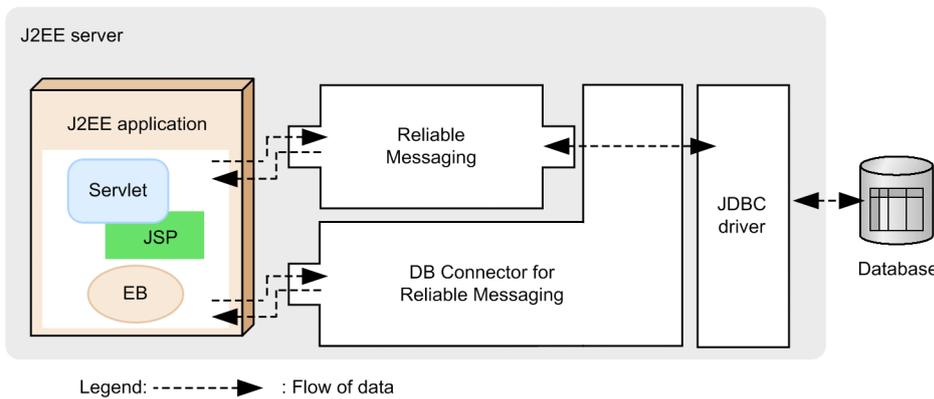


(b) When the same database is accessed with the JMS interface and JDBC interface

When the JMS interface and JDBC interface are used to access the database with the user program, the same database can be accessed by two types of interfaces if the preconditions for connection sharing are satisfied. At this time, use the configuration shown in the following figure.

By using this configuration, you can improve the processing performance of the transactions through connection sharing and utilize the resources effectively.

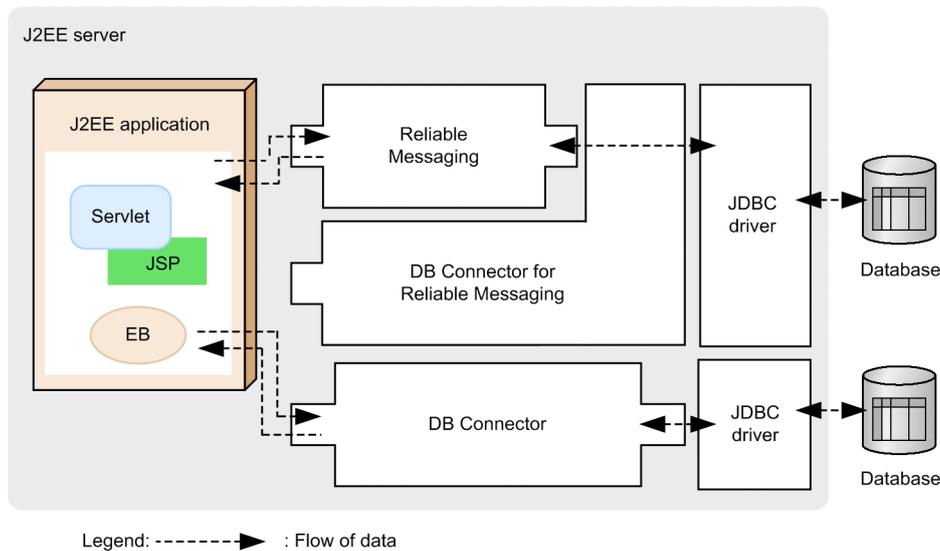
Figure 3-23: Configuration when the same database is accessed with the JMS interface and JDBC interface



(c) When different databases are accessed with the JMS interface and JDBC interface

When the JMS interface and JDBC interface are used to access the database with the user program, you must use a configuration in which different databases will be accessed for each interface if the preconditions for connection sharing are not satisfied. At this time, use the configuration shown in the following figure.

Figure 3-24: Configuration when different databases are accessed with the JMS interface and JDBC interface



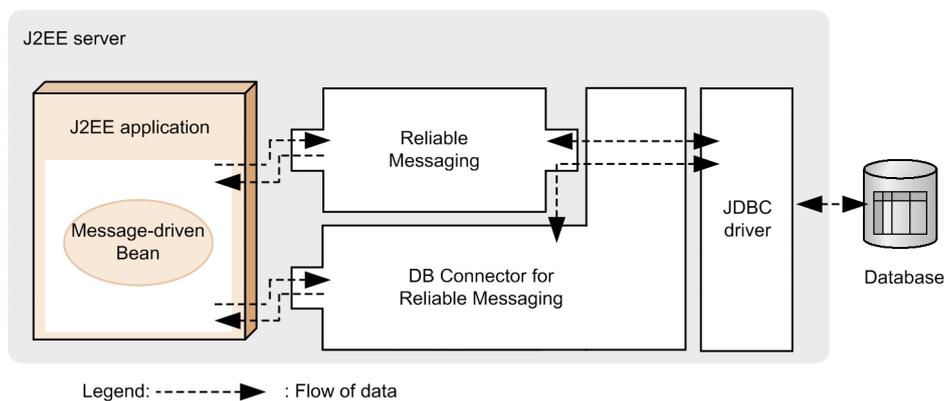
Use DB Connector for Cosminexus RM and Cosminexus RM to access the database queue. Also, to access the database tables, you must prepare and use DB Connector separately.

(d) When the Message-driven Beans are used

When the Message-driven Beans are used, use the configuration show in the following figure. If the connection sharing conditions are satisfied, you can use the JDBC interface of DB Connector for Cosminexus RM and share the connection between the Message-driven Beans and the JDBC interface.

Note that when the Message-driven Beans are used, you can use the global transactions and local transactions when the associated resource adapter is Cosminexus RM 01-01 or later. When you use the Cosminexus RM 01-00 resource adapter, you must use a global transaction. At this time, you cannot use a local transaction, but if the preconditions for connection sharing are satisfied, a global transaction is concluded through a single phase.

Figure 3-25: Configuration when the Message-driven Beans are used



3.7.3 Available functionality

For details on the functionality available for connections using DB Connector for Cosminexus RM and Cosminexus RM, see 2.7 *Functionality for DB Connector for Cosminexus RM* in the manual *uCosminexus Application Server Reliable Messaging*.

Note that when you use the JDBC interface to connect to a database, you can use the same functionality as when you use DB Connector for connection. For details on the available functionality, see 3.3.4 *Resource adapter functionality*.

3.7.4 Connectable databases

This subsection describes the databases that can be connected to by using DB Connector for Cosminexus RM and Cosminexus RM.

(1) Types of connectable databases

The databases that can be connected by using DB Connector for Cosminexus RM and Cosminexus RM include HiRDB and Oracle. Note that you cannot connect to SQL Server and XDM/RD E2.

(2) Mapping the databases and JDBC drivers

To use DB Connector for Cosminexus RM and Cosminexus RM to connect to a database, you require a JDBC driver compatible with the database.

The following table describes the database to be connected to and the available JDBC drivers.

Table 3-35: Database to be connected to and the available JDBC drivers (for connecting to a database queue)

Database	JDBC driver	
	HiRDB Type4 JDBC Driver	Oracle JDBC Thin Driver
HiRDB Version 8	Y	N
HiRDB Version 9	Y	N
Oracle	N	Y

Legend:

Y: Available

N: Not available

(3) Notes

Note the following when you use DB Connector for Cosminexus RM and Cosminexus RM to connect to a database:

- A JDBC-specific transaction control cannot be executed with the JDBC connection (`java.sql.Connection`, `Connection`) provided by DB Connector for Cosminexus RM. If a JDBC-specific transaction control is executed, an exception occurs when the `setAutoCommit(boolean)` method is invoked with the argument `false`, and when the `releaseSavepoint(Savepoint)` method, `rollback(Savepoint)` method, `setSavepoint()` method, and `setSavepoint(String)` method are invoked. Use DB Connector when you want to execute a JDBC-specific transaction control.
- The collection points and event ID of the trace based performance analysis for DB Connector for Cosminexus RM partially differs from that for DB Connector.
For a JDBC connection (`java.sql.Connection`, `Connection`), the trace based performance analysis is output at the two trace collection points of JDBC connection of DB Connector for Cosminexus RM with one-time access, and the JDBC connection in DB Connector.
Also, the trace based performance analysis is output at the same trace collection points as DB Connector for the JDBC connection products (such as `java.sql.Statement`). The trace based performance analysis is only output at the trace collection points for DB Connector for Cosminexus RM for `java.sql.DataSource`.
- When the operation information is monitored with DB Connector for Cosminexus RM, incorrect values are output for the following items of DB Connector for Cosminexus RM resource adapter. DB Connector for Cosminexus RM obtains a connection via Cosminexus RM; therefore, the monitored information of the integrated resource adapter is accumulated and output to Cosminexus RM.
 - Transaction support level
 - Current value of the pool (total)
 - Number of connections in use
 - Number of unused connections

- Execution count of the `createManagedConnection()` method of `ManagedConnectionFactory`
 - Execution count of the `getConnection()` method of `ManagedConnection`
 - Execution count of the `cleanup()` method of `ManagedConnection`
 - Execution count of the `destroy()` method of `ManagedConnection`
 - Execution time of the `allocateConnection()` method of `ConnectionManager`
 - Execution time of the `createManagedConnection()` method of `ManagedConnectionFactory`
 - Failure count of the `allocateConnection()` method of `ConnectionManager`
 - Number of times the FATAL error occurred in `ManagedConnection`
- DB Connector for Cosminexus RM shares the connection pool of the integrated Cosminexus RM. Therefore, you need not specify the settings for each functionality of the connection pool in DB Connector for Cosminexus RM.
 - To use a resource adapter, you must resolve the references from the J2EE application to the resource adapter. When you customize a J2EE application that uses the resource adapters, resolve the references from the J2EE application to the resource adapter.

3.7.5 Types of DB Connector for Cosminexus RM (RAR file)

When you use DB Connector for Cosminexus RM to connect to a database, use a RAR file according to the JDBC driver to be used. Use the server management commands to operate the RAR files. For details on how to use the server management commands to operate the RAR files, see the *uCosminexus Application Server Application Setup Guide*.

The following table describes the types of JDBC drivers and the corresponding RAR files.

Table 3-36: Mapping the JDBC drivers and RAR files

JDBC driver	RAR file	Explanation
HiRDB Type4 JDBC Driver	<code>DBConnector_HiRDB_Type4_CP_Cosminexus_RM.rar</code>	Use this file when the transactions are not managed, or when a local transaction is used.
	<code>DBConnector_HiRDB_Type4_XA_Cosminexus_RM.rar#</code>	Use this file when a global transaction is used.
Oracle JDBC Thin Driver	<code>DBConnector_Oracle_CP_Cosminexus_RM.rar</code>	Use this file when the transactions are not managed, or when a local transaction is used.
	<code>DBConnector_Oracle_XA_Cosminexus_RM.rar#</code>	Use this file when a global transaction is used.

#: The light transaction functionality cannot be enabled.

3.7.6 Preconditions for connecting to a HiRDB queue

This subsection describes the preconditions for connecting to a HiRDB queue.

(1) Preconditions for HiRDB Version 8

The preconditions for HiRDB Version 8 are as follows:

- **Available JDBC driver**
The available JDBC driver is HiRDB Type4 JDBC Driver.
- **Connection method**
You use either of the following RAR files:
 - `DBConnector_HiRDB_Type4_CP_Cosminexus_RM.rar`

- DBConnector_HiRDB_Type4_XA_Cosminexus_RM.rar

The specifiable transaction support level varies according to the RAR file type. The availability of the light transaction also differs.

The following table describes the transaction support levels that can be specified for each RAR file.

Table 3-37: Transaction support levels available for each RAR file (HiRDB Version 8)

Used DB Connector for Cosminexus RM (RAR file)	Transaction support level	Light transaction	
		Enabled	Disabled
DBConnector_HiRDB_Type4_CP_Cosminexus_RM.rar	NoTransaction LocalTransaction	Y	Y
DBConnector_HiRDB_Type4_XA_Cosminexus_RM.rar	XATransaction	--	Y

Legend:
 Y: Available
 --: Not applicable

(2) Preconditions for HiRDB Version 9

The preconditions for HiRDB Version 9 are as follows:

- **Available JDBC driver**

The available JDBC driver is HiRDB Type4 JDBC Driver.

- **Connection method**

You use either of the following RAR files:

- DBConnector_HiRDB_Type4_CP_Cosminexus_RM.rar
- DBConnector_HiRDB_Type4_XA_Cosminexus_RM.rar

The specifiable transaction support level varies according to the RAR file type. The availability of the light transaction also differs.

The following table describes the transaction support levels that can be specified for each RAR file.

Table 3-38: Transaction support levels available for each RAR file (HiRDB Version 9)

Used DB Connector for Cosminexus RM (RAR file)	Transaction support level	Light transaction	
		Enabled	Disabled
DBConnector_HiRDB_Type4_CP_Cosminexus_RM.rar	NoTransaction LocalTransaction	Y	Y
DBConnector_HiRDB_Type4_XA_Cosminexus_RM.rar	XATransaction	--	Y

Legend:
 Y: Available
 --: Not applicable

3.7.7 Preconditions for connecting to an Oracle queue

This subsection describes the preconditions for connecting to an Oracle queue.

- **Available JDBC driver**

The available JDBC driver is Oracle JDBC Thin Driver.

- **Connection method**

Use `DBConnector_Oracle_CP_Cosminexus_RM.rar` or `DBConnector_Oracle_XA_Cosminexus_RM.rar`.

The specifiable transaction support level varies according to the RAR file type. The availability of the light transaction also differs.

The following table describes the transaction support levels that can be specified for each RAR file.

Table 3–39: Transaction support levels available for each RAR file (Oracle9i)

Used DB Connector for Cosminexus RM (RAR file)	Transaction support level	Light transaction	
		Enabled	Disabled
<code>DBConnector_Oracle_CP_Cosminexus_RM.rar</code>	<code>NoTransaction</code> <code>LocalTransaction</code>	Y	Y
<code>DBConnector_Oracle_XA_Cosminexus_RM.rar</code>	<code>XATransaction</code>	--	Y

Legend:

Y: Available

--: Not applicable

3.7.8 Settings for connecting to a database queue

For details on the settings for connecting to a database queue, see *2.7 Functionality for DB Connector for Cosminexus RM* in the manual *uCosminexus Application Server Reliable Messaging*.

3.8 Outbound connection with OpenTP1 (SPP or TP1/Message Queue)

This section describes the functionality for connecting OpenTP1 (SPP or TP1/Message Queue) and Outbound. The following table describes the organization of this section.

Table 3-40: Organization of this section (Outbound connection with OpenTP1 (SPP or TP1/Message Queue))

Category	Title	Reference location
Explanation	Connections using uCosminexus TP1 Connector	3.8.1
	Connections using TP1/Message Queue - Access	3.8.2
Settings	Connection settings for OpenTP1 and Outbound	3.8.3

Note: The functionality-specific explanation is not available for "Implementation" and "Operations".

Application Server can connect to OpenTP1 SPP or TP1/Message Queue. Use the following resource adapters to connect to OpenTP1:

- Use uCosminexus TP1 Connector to connect to OpenTP1 SPP.
- Use TP1/Message Queue - Access to connect to TP1/Message Queue.

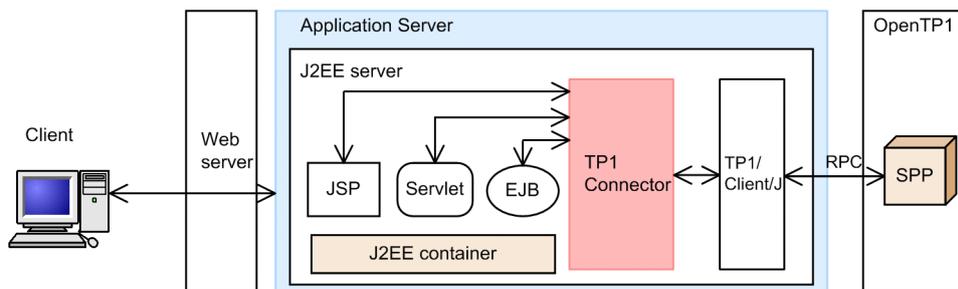
This section describes the connections with OpenTP1.

3.8.1 Connections using uCosminexus TP1 Connector

To connect to a J2EE server and OpenTP1 SPP, you combine and use uCosminexus TP1 Connector and TP1/Client/J as the resource adapters.

The following figure shows OpenTP1 integration.

Figure 3-26: OpenTP1 integration using uCosminexus TP1 Connector



uCosminexus TP1 Connector uses TP1/Client/J RPC to connect to OpenTP1, and accesses OpenTP1 SPP. Due to this, you can integrate OpenTP1 SPP with a J2EE server.

! Important note

To use a resource adapter, you must resolve the references from the J2EE application to the resource adapter. When you customize a J2EE application that uses the resource adapters, resolve the references from the J2EE application to the resource adapter.

For details on the uCosminexus TP1 Connector settings, see the uCosminexus TP1 Connector documentation. For details on the TP1/Client/J settings, see the *OpenTP1 Version 7 TP1/Client User's Guide TP1/Client/J*.

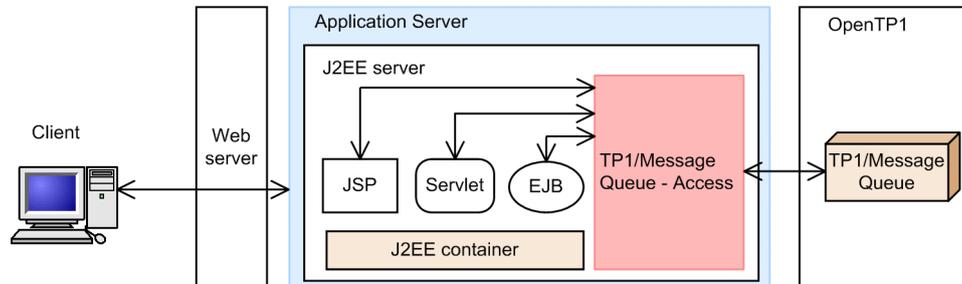
Also, for details on how to set up the resource adapters using the server management commands, see the uCosminexus TP1 Connector documentation, the manual *OpenTP1 Version 7 TP1/Client User's Guide TP1/Client/J*, and *4.4 Settings for connecting to other resources* in the *uCosminexus Application Server Application Setup Guide*.

3.8.2 Connections using TP1/Message Queue - Access

To connect Application Server and TP1/Message Queue, you use TP1/Message Queue - Access as the resource adapter.

The following figure shows the TP1/Message Queue integration using TP1/Message Queue - Access.

Figure 3-27: TP1/Message Queue integration using TP1/Message Queue - Access



TP1/Message Queue - Access provides the JMS interface. The J2EE server uses this JMS interface to connect to TP1/Message Queue.

! Important note

To use a resource adapter, you must resolve the references from the J2EE application to the resource adapter. When you customize a J2EE application that uses the resource adapters, resolve the references from the J2EE application to the resource adapter.

For details on the TP1/Message Queue - Access settings, see the *OpenTP1 Version 7 TP1/Message Queue - Access User Guide*.

Also, for details on how to set up the resource adapters using the server management commands, see *4.4 Settings for connecting to other resources* in the *uCosminexus Application Server Application Setup Guide*.

3.8.3 Connection settings for OpenTP1 and Outbound

For details on the settings for establishing a connection using uCosminexus TP1 Connector, see the uCosminexus TP1 Connector documentation, the *OpenTP1 Version 7 TP1/Client User's Guide TP1/Client/J*, and *4.4 Settings for connecting to other resources* in the *uCosminexus Application Server Application Setup Guide*.

For details on the settings for establishing a connection using TP1/Message Queue - Access, see the *OpenTP1 Version 7 TP1/Message Queue - Access User Guide*, and *4.4 Settings for connecting to other resources* in the *uCosminexus Application Server Application Setup Guide*.

3.9 Inbound connection with OpenTP1

When you invoke a business program running on a J2EE server for Application Server from the SUP on a legacy system using OpenTP1, you use the TP1 inbound adapter as the resource adapter for connecting OpenTP1 and the J2EE server.

The TP1 inbound adapter is a resource adapter conforming to the Connector 1.5 specifications and provides the following functionality:

- Substitution functionality for OpenTP1 schedule services
- Functionality for executing OpenTP1 and RPC communication

By using the above functionality, you can invoke the business programs on the J2EE server using the same procedure as that for invoking OpenTP1 SPP from OpenTP1 SUP.

For details on connecting to OpenTP1 using the TP1 inbound adapter, see *4. Invoking Application Server from OpenTP1 (TP1 Inbound Integrated Function)*.

Important note

The TP1 inbound integrated function can be invoked from the SUP, SPP, and MHP, but the invocation from the SUP will be described as an example in this manual.

3.10 Connection with Cosminexus JMS Provider

When you use the JMS Provider functionality provided by Application Server, you use the CJMSP Broker process to manage the message destinations. Use the CJMSP resource adapter to connect the J2EE server and CJMSP Broker.

By using the CJMSP resource adapter, you can send and receive messages with the PTP messaging model or Pub/Sub messaging model.

For details on connecting to OpenTP1 using the CJMSP resource adapter, see *7. Cosminexus JMS Provider*.

3.11 Connecting to the SMTP server

A J2EE application can send a mail to the SMTP server by using JavaMail.

For details on the settings for connecting to the SMTP server, see *6.3 Mail configuration settings* in the *uCosminexus Application Server Application Setup Guide*.

3.12 Using the JavaBeans resources

This section describes the use of the JavaBeans resources.

The following table describes the organization of this section.

Table 3-41: Organization of this section (Using the JavaBeans resources)

Category	Title	Reference location
Explanation	JavaBeans resource functionality	3.12.1
	Procedure for starting the JavaBeans resources	3.12.2
Implementation	Implementing the JavaBeans resources	3.12.3
Settings	Setting up the JavaBeans resources	3.12.4
	Replacing the JavaBeans resources	3.12.5

Note: The functionality-specific explanation is not available for "Operations".

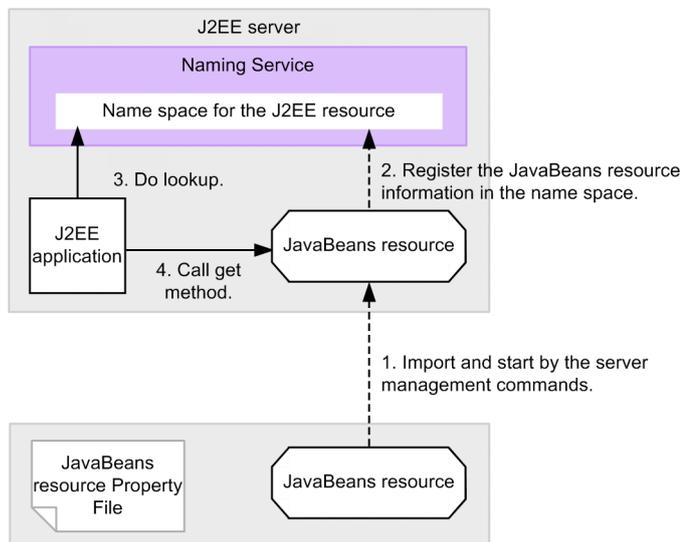
3.12.1 JavaBeans resource functionality

With the JavaBeans resources, you can store and manage the settings for operating the J2EE applications in a batch. You can obtain the settings by looking up the JavaBeans resource from the J2EE application by using JNDI.

3.12.2 Procedure for starting the JavaBeans resources

You can start multiple JavaBeans resources on a J2EE server. The following figure shows the procedure for starting the JavaBeans resources when the server management commands are used.

Figure 3-28: Procedure for starting the JavaBeans resources



1. Import and start the JavaBeans resource.
Use the server management commands to import and start the JavaBeans resource on the J2EE server.
2. Register the JavaBeans resource references to the Naming Service.
When you start the JavaBeans resource, register the JavaBeans resource references in the J2EE resource Namespace of the Naming Service.
3. Look up from the J2EE application to the Naming Service.

Perform a look up from the J2EE application using the JavaBeans resource into the Naming Service. The `set` method of the JavaBeans resource is invoked by extending the lookup processing, and the property values specified in the JavaBeans resource property file are set up.

4. Use the obtained references to invoke the `get` method of the JavaBeans resource.
Use the references obtained by lookup to invoke the `get` method of the JavaBeans resource, and use the `get` method to obtain the values specified with the `set` method.

If a single class configures the implementation class of the JavaBeans resource, the class is available in the class file format, and if multiple classes configure the implementation class, like the classes related to the JavaBeans resource class, the classes are available in the JAR file format.

3.12.3 Implementing the JavaBeans resources

This subsection uses examples to describe the procedure for implementing the JavaBeans resources.

(1) Import settings

This section describes the required settings and notes when you use the server management commands (`cjimportjb` command) to import the JavaBeans resources.

(a) Settings for the JavaBeans resource property file

Create the JavaBeans resource property file keeping the following points in mind:

- Specify the class name and implementation class name of the JavaBeans resource in the `<res-type>` tag and `<class-name>` tag. If you want to set the same value in the `<res-type>` tag and `<class-name>` tag, you can omit the `<res-type>` tag.
- Specify the method names of the `set` method and `get` method in the `<property-name>` tag.
- Specify the argument type of the `set` method in the `<property-type>` tag.
If the actual argument type of the applicable `set` method does not match the value in the `<property-type>` tag, an error occurs during lookup.
- Specify the value passed to the argument of the `set` method in the `<property-value>` tag.

An example of settings for the JavaBeans resource property file is as follows:

```
<!DOCTYPE hitachi-javabeans-resource-property PUBLIC "-//Hitachi, Ltd.//DTD JavaBeans
Resource Property 7.0//EN"
'http://localhost/hitachi-javabeans-resource-property_7_0.dtd'>

<hitachi-javabeans-resource-property>
  <description></description>
  <display-name>JavaBean_resource</display-name>
  <class-name>com.mycompany.mypackage.MyJavaBean</class-name>
  <runtime>
    <property>
      <property-name>UserName</property-name>
      <property-type>java.lang.String</property-type>
      <property-value>Hitachi</property-value>
    </property>
    <property>
      <property-name>UserID</property-name>
      <property-type>java.lang.String</property-type>
      <property-value>01234567</property-value>
    </property>
  </runtime>
</hitachi-javabeans-resource-property>
```

The template file (`jb_template.xml`) of the JavaBeans resource property file is stored in the following directory:

`Cosminexus-installation-directory\CC\admin\templates`

(b) How to use the -d option

By using the `-d` option during an import operation, the JavaBeans resources can be imported with the same directory configuration and without creating an archive. Specify the top directory to be imported as the directory to be specified in `-d`.

An example of specifying the `-d` option when you import a JavaBeans resource is as follows. In this example, the "MyJavaBean" class with package name "com.mycompany.mypackage" is imported.

```
Directory-specified-in--d\
+ com\
+ mycompany
+ mypackage
+ MyJavaBean.class
```

The `-d` option imports all the files existing beneath the specified directory, so do not include unnecessary files in the directory.

Notes on importing multiple JavaBeans resources

You cannot import a JavaBeans resource with the same implementation class name as that of an imported JavaBeans resource. Delete the JavaBeans resource that was imported earlier and then import the target JavaBeans resource, or change and re-create the implementation class name, and then import the JavaBeans resource.

(2) Creating the implementation class of the JavaBeans resources

You declare the method for operating the data (property) managed by JavaBeans. To register the data, specify the `set` method (`set + property-name`). To reference the data, specify the `get` method (`get + property-name`).

An example implementation of the class that registers and references the JavaBeans resource is as follows:

```
package com.mycompany.mypackage;
public class MyJavaBean {
    private String username;
    private String userid;

    public void setUsername(String user_name) {
        this.username = user_name;
    }
    public void setUserID(String user_id) {
        this.userid = user_id;
    }
    public String getUsername() {
        return this.username;
    }
    public int getUserID() {
        return this.userid;
    }
}
```

(3) Application settings

This section describes the implementation and definitions required in an application when you use the JavaBeans resources.

(a) Implementing lookup (JavaBeans resources)

You obtain a JavaBeans resource by using lookup or DI. This section describes how to look up with the `java:comp/env` format.

```
Context initCtx = new InitialContext();
MyJavaBean jib = (MyJavaBean) initCtx.lookup("java:comp/env/bean/myJB");
```

Like other resources, the range of JavaBeans resource lookup is the application in the same J2EE server process.

(b) Contents defined in the DD (JavaBeans resources)

To use lookup to obtain a JavaBeans resource, you define the information on the name to be looked up and the implementation class name in the DD (`ejb-jar.xml` or `web.xml`). The tags to be specified are as follows:

- In the `<resource-env-ref-name>` tag, you specify the value to be specified in the `java:comp/env` format for lookup.
- In the `<resource-env-ref-type>` tag, specify the implementation class name of the JavaBeans resource.

Also, when you deploy a created application on the J2EE server, bind the reference name in lookup and the actual name with `linked-to`. To execute this operation, use the server management commands (`cjsetappprop` command).

- Use `cjgetappprop` to obtain the property file of the relevant application.
- Add the `<linked-to>` tag in the `<resource-env-ref>` tag and specify the display name of the JavaBeans resource to be used.
- Use `cjsetappprop` to set up the property file of the relevant application.

The following is an example of the `<resource-env-ref>` tag in the property file passed by `cjsetappprop`:

```
<resource-env-ref>
  <resource-env-ref-name>bean/myJB</resource-env-ref-name>
  <resource-env-ref-type>com.mycompany.mypackage.MyJavaBean</resource-env-ref-type>
  <linked-to>JavaBean_resource</linked-to>
</resource-env-ref>
```

(4) Starting and terminating an application

You start or terminate an application that uses a JavaBeans resource with the server management commands or Management Server. For details on how to start an application, see *10.2.1 Starting J2EE applications* in the *uCosminexus Application Server Application Setup Guide*. For details on how to terminate an application, see *10.2.2 Stopping J2EE applications* in the *uCosminexus Application Server Application Setup Guide*.

Application Server provides a sample program for JavaBeans resources. For an overview and the execution method of the sample program, see *Appendix M.6 Sample program for JavaBeans resources* in the *uCosminexus Application Server System Setup and Operation Guide*.

3.12.4 Setting up the JavaBeans resources

This subsection describes the settings for the JavaBeans resources.

You use the server management commands to set up the properties for the JavaBeans resources and to import the JavaBeans resources.

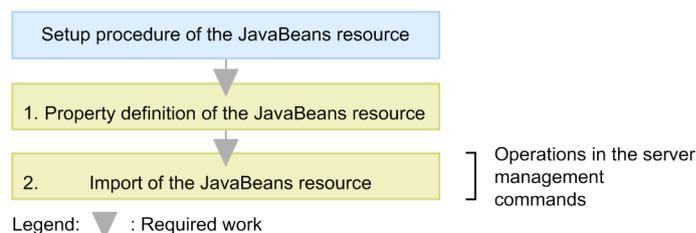
To set up the JavaBeans resources, use the server management commands.

This subsection describes the procedure for setting up a new JavaBeans resource, the procedure for changing the settings, and the procedure for replacing the JavaBeans resource.

(1) Procedure for setting up a new JavaBeans resource

The following figure shows the procedure for setting up a new JavaBeans resource.

Figure 3-29: Procedure for setting up a new JavaBeans resource



A description of points 1 and 2 in the figure is as follows:

1. Create the JavaBeans resource property file and define the JavaBeans resource properties.

Use the template for the JavaBeans resource property file to create the JavaBeans resource property file, and then define the properties for the JavaBeans resource. The template for the JavaBeans resource property file is stored in the following directory:

- In Windows
`Cosminexus-installation-directory\CC\admin\templates\jb_template.xml`
- In UNIX
`/opt/Cosminexus/CC/admin/templates/jb_template.xml`

For details on the content that can be specified in the definition of the JavaBeans resource properties, see (4) *Content that can be specified in the definition of the JavaBeans resource properties*.

2. Use the server management commands to import the JavaBeans resource.

Specify the path of the JavaBeans resource property file set up in point 1 and the JAR file containing the JavaBeans resource, in the argument, use the `cjimportjb` command, and then import the JavaBeans resource.

For details on the operations with server management commands, see 3. *Basic Operations of Server Management Commands* in the *uCosminexus Application Server Application Setup Guide*. Also, for details on the commands, see 2.4 *Resource operation commands used with the J2EE server* in the *uCosminexus Application Server Command Reference Guide*. For details on the property files, see 4.2 *Property files for JavaBeans resources* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

Reference note

- When you import multiple JavaBeans resources, you cannot import a JavaBeans resource with the same implementation class name as an imported JavaBeans resource. Delete the JavaBeans resource that was imported earlier, or change and re-create the implementation class name, and then import the JavaBeans resource.
 Also, if another class file is being used apart from the implementation class of the JavaBeans resource, the other class file is not checked.
- If you use the `-d` option in the `cjimportjb` command during the import operation, the JavaBeans resources can be imported with the same directory configuration and without generating an archive. As directory, specify the top directory to be imported.
 Note that if the specified directory is not a directory configuration, all the files directly beneath the specified directory will be imported.
 To import all the files existing under the specified directory, do not include unnecessary files in the directory.

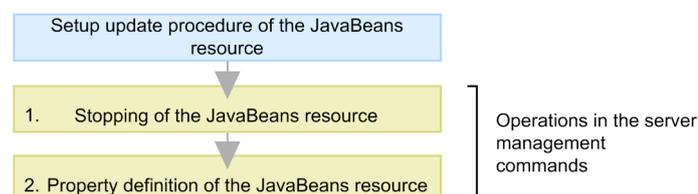
Important note

To use a JavaBeans resource, you must resolve the references from the J2EE application to the JavaBeans resource. When you define the properties of a J2EE application that uses the JavaBeans resource, resolve the references from the J2EE application to the JavaBeans resource.

(2) Procedure for changing the settings of a JavaBeans resource

The following figure shows the procedure for changing the settings of a JavaBeans resource.

Figure 3-30: Procedure for changing the settings of a JavaBeans resource



Legend: ▼ : Required work

A description of points 1 and 2 in the figure is as follows:

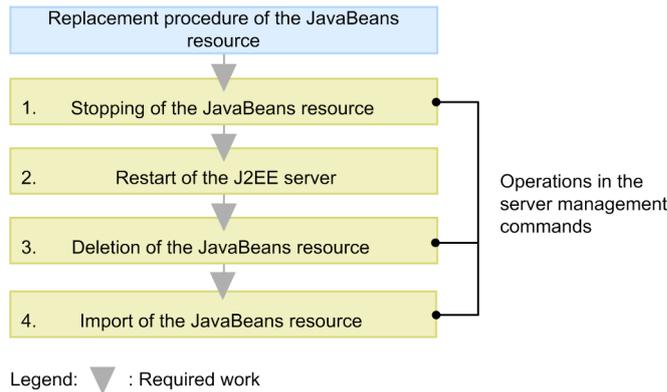
1. Use the server management commands to stop the JavaBeans resource.
 Use the `cjstopjb` command to stop the JavaBeans resource. Note that before you stop the JavaBeans resource, stop all the J2EE applications that are using the JavaBeans resource.

2. Use the server management commands to define the JavaBeans resource properties.
 Use the `cjgetjbprop` command to obtain the JavaBeans resource property file, edit the file, and then use the `cjsetjbprop` command to apply the edited content.
 For details on the content you can specify in the definition of the JavaBeans resource properties, see (4) *Content that can be specified in the definition of the JavaBeans resource properties*.

(3) Procedure for replacing the JavaBeans resource

The following figure shows the procedure for replacing the JavaBeans resource.

Figure 3-31: Procedure for replacing the JavaBeans resource



A description of points 1 to 4 in the figure is as follows:

1. Use the server management commands to stop the JavaBeans resource.
 Use the `cjstopjb` command to stop the JavaBeans resource you want to replace. Note that before you stop the JavaBeans resource, stop all the J2EE applications that are using the JavaBeans resource.
2. Restart the J2EE server.
 Use the `cjstopsv` command to stop the J2EE server, and then use the `cjstartsv` command to restart the J2EE server.
3. Use the server management commands to delete the JavaBeans resource.
 Use the `cjdeletejb` command to delete the JavaBeans resource you want to replace.
4. Use the server management commands to import the JavaBeans resource.
 Use the `cjimportjb` command to import a new JavaBeans resource.

(4) Content that can be specified in the definition of the JavaBeans resource properties

To set up new JavaBeans resource properties, use the template of the JavaBeans resource property file to create the JavaBeans resource property file, and then define the properties. Also, to change the settings, use the server management commands to obtain the JavaBeans resource property file, and then define the properties. Use the `cjgetjbprop` command to obtain the JavaBeans resource property file, edit the file, and then use the `cjsetjbprop` command to apply the edited content.

The following table describes the main content that can be specified in the definition of the JavaBeans resource properties.

Table 3-42: Main content that can be specified in the definition of the JavaBeans resource properties

Category	Item	Settings
Runtime properties	Resource type	Specify the class name of the JavaBeans resource in the <code><res-type></code> tag.
	Property information	Specify the information in the following tags beneath the <code><property></code> tag:

Category	Item	Settings
Runtime properties	Property information	<ul style="list-style-type: none"> Specify the method name of the <code>set</code> method and <code>get</code> method of the JavaBeans resource in the <code><property-name></code> tag. Specify the argument type of the <code>set</code> method of the JavaBeans resource in the <code><property-type></code> tag. Specify the value passed to the argument of the <code>set</code> method of the JavaBeans resource in the <code><property-value></code> tag.
Optional name information	Optional name	<p>Specify the optional name for the JavaBeans resource in the <code><resource-env-external-property></code>-<code><optional-name></code> tag, when you want to assign an optional name to the JavaBeans resource.</p> <p>For the optional name settings, see 2.6.6 <i>Setting the optional names for the J2EE resources</i>.</p>

Reference note

The `<res-auth>` attribute and `<res-sharing-scope>` attribute specified in the `<resource-ref>` tag of the DD (`ra.xml`) cannot be specified in the `<resource-env-external-property>` tag of the JavaBeans resource property file.

3.12.5 Replacing the JavaBeans resources

This subsection describes the procedure for replacing the JavaBeans resources. To replace a JavaBeans resource, stop the J2EE application and JavaBeans resource, restart the J2EE server, and then replace the JavaBeans resource. You use the server management commands to replace the JavaBeans resources.

The procedure is as follows:

1. Stop the J2EE application that is using the JavaBeans resource you want to replace.

To stop the J2EE application, execute the `cjstopapp` command. The format and example of execution of the `cjstopapp` command are as follows:

Format of execution

```
cjstopapp J2EE-server-name -name J2EE-application-name
```

Example of execution

```
cjstopapp MyServer -name Appl
```

2. Stop the JavaBeans resource you want to replace.

To stop the JavaBeans resource, execute the `cjstopjb` command. The format and example of execution of the `cjstopjb` command are as follows:

Format of execution

```
cjstopjb server-name -resname display-name-of-JavaBeans-resource
```

Example of execution

```
cjstopjb MyServer -resname javabeansname
```

3. Restart the J2EE server.

To delete a JavaBeans resource that is started once, you must restart the J2EE server. To stop the J2EE server, execute the `cjstopsv` command. The format and example of execution of the `cjstopsv` command are as follows:

Format of execution

```
cjstopsv J2EE-server-name
```

Example of execution

```
cjstopsv MyServer
```

After you stop the J2EE server, start the server once again. To start the J2EE server, execute the `cjstartsv` command. The format and example of execution of the `cjstartsv` command are as follows:

Format of execution

```
cjstartsv J2EE-server-name
```

Example of execution

```
cjstartsv MyServer
```

4. Delete the JavaBeans resource you want to replace.

To delete the JavaBeans resource, execute the `cjdeletejb` command. The format and example of execution of the `cjdeletejb` command are as follows:

Format of execution

```
cjdeletejb server-name -resname display-name-of-JavaBeans-resource
```

Example of execution

```
cjdeletejb MyServer -resname MyJavaBeans
```

5. Import the replaced JavaBeans resource.

To import the JavaBeans resource, execute the `cjimportjb` command. The format and example of execution of the `cjimportjb` command are as follows:

Format of execution

```
cjimportjb server-name -f JAR-file-path -c path-of-the-JavaBeans-resource-property-file
```

Example of execution

```
cjimportjb MyServer -f Myjavabeans.jar -c Myjavabeansprop.xml
```

6. Start the JavaBeans resource.

To start the JavaBeans resource, execute the `cjstartjb` command. The format and example of execution of the `cjstartjb` command are as follows:

Format of execution

```
cjstartjb server-name -resname display-name-of-JavaBeans-resource
```

Example of execution

```
cjstartjb MyServer -resname javabeansname
```

7. Start the J2EE application.

To start the J2EE application, execute the `cjstartapp` command. The format and example of execution of the `cjstartapp` command are as follows:

Format of execution

```
cjstartapp J2EE-server-name -name J2EE-application-name
```

Example of execution

```
cjstartapp MyServer -name App1
```

3.13 Connection with other resources

This section describes the connections with resources other than those described in the previous sections.

3.13.1 Resource adapters used for connection with other resources

With Application Server, you can use the resource adapters conforming to the Connector 1.0 specifications or Connector 1.5 specifications and connect to any resource.

The Connector 1.5 specifications determine the following two models for the communication between the J2EE server and resource adapter. With Application Server, you can use a resource adapter that is compatible with these communication models.

- **Outbound**

A communication model that accesses the resource adapter from the J2EE server.

- **Inbound**

A communication model that accesses the J2EE server from the resource adapter.

Application Server supports the following contracts of the Connector 1.5 specifications:

- **Lifecycle Management**

This contract is used to manage the start or termination processing of the resource adapter.

- **Work Management**

This contract is used by the resource adapter to handle threads.

- **Message inflow**

This contract is used to receive the messages from the EIS, and use the Message-driven Beans from the resource adapter.

- **Transaction inflow**

This contract is used to handle the transactions in the message inflow.

For details on the Application Server functionality based on these contracts, see *3.16.1 Managing the resource adapter lifecycle*, *3.16.2 Managing the resource adapter work*, *3.16.3 Message inflow*, and *3.16.4 Transaction inflow*.

Application Server also supports the functionality added by the Connector 1.5 specifications for the following existing contracts:

- **Connection Management**

You can detect invalid connections.

For details on the functionality for detecting invalid connections, see *3.15.1 Detecting the connection errors*.

- **Common Client Interface**

You can use the message inflow-related APIs. For details on the unique specifications provided for these APIs in Application Server, see *3.16.7 Application Server-specific Connector 1.5 API specifications*.

3.13.2 Functionality available for other resource connections

This subsection describes the functionality that can be used with Application Server from among the functionality provided in Connector 1.5. The available functionality varies for Outbound and for Inbound. For details on the Application Server functionality available with the resource adapters conforming to the Connector 1.5 specifications, see *3.3.4 Resource adapter functionality*.

(1) Functionality available with Outbound

The following functionality is available with Outbound:

- Transaction management by local transactions and global transactions
- Using the message queue

(2) Functionality available with Inbound

With Inbound, you can use Non-Transacted Delivery (message delivery in which the EIS does not participate in the transaction) and Transacted Delivery (message delivery in which the EIS participates in the global transaction of Application Server).

Reference note

Transaction inflow (message delivery in which the invocation of the Message-driven Bean participates in the EIS transaction) is only supported by the TP1 inbound integrated function.

(3) Mapping the usage and transaction levels of the resource adapters

The following table describes the transaction levels that can be specified when a resource adapter is used with Outbound.

Table 3-43: Transaction levels that can be specified when a resource adapter is used with Outbound

Used communication model	Transaction management method	Transaction attribute	Transaction support level		
			NoTransaction	LocalTransaction	XATransaction [#]
Outbound	CMT	Required RequiresNew Supports NotSupported Mandatory Never	Y	Y	Y
	BMT	--	Y	Y	Y

Legend:

Y: Available

--: Not applicable

[#]: If the resource adapter is included in a J2EE application, XATransaction is not available.

Tip

The following table describes the transaction attributes that can be specified when a resource adapter is used with Inbound.

Table 3-44: Transaction attributes that can be specified when a resource adapter is used with Inbound

Used communication model	Transaction management method	Transaction attribute
Inbound	CMT	Required [#]
		NotSupported
	BMT	--

Legend:

--: Not applicable

[#]: You cannot specify this attribute for the CJMSP resource adapter or FTP inbound adapter.

3.14 Functionality for performance tuning

This section describes the functionality for performance tuning.

The following table describes the organization of this section.

Table 3-45: Organization of this section (Functionality for performance tuning)

Category	Title	Reference location
Explanation	Connection pooling	3.14.1
	Functionality available with connection pooling	3.14.2
	Connection sharing or association	3.14.3
	Statement pooling	3.14.4
	Light transaction	3.14.5
	In-process transaction service	3.14.6
	Caching the DataSource objects	3.14.7
	Optimizing the container-managed sign-on for DB Connector	3.14.8
Implementation	Definitions in <code>cosminexus.xml</code>	3.14.9
Settings	Settings in the execution environment	3.14.10

Note: The functionality-specific explanation is not available for "Operations".

3.14.1 Connection pooling

This functionality pools the resource connections (JDBC connection and resource adapter connections) according to the volume of resource access from the J2EE components, such as servlets, JSPs, and EJBs. By pooling these connections, the resource connection requests from the user application are processed at a high speed.

(1) Preconditions

The connection pooling functionality might be available or unavailable depending on the types of resource and the combinations of connection methods. The following table describes the usage of the connection pooling functionality.

Table 3-46: Usability of the connection pooling functionality

Types of resources	Connection method	Usability
Database	DB Connector	Y
Database queue	DB Connector for Cosminexus RM and Cosminexus RM	Y
OpenTP1	uCosminexus TP1 Connector	Y
	TP1/Message Queue - Access	Y
SMTP server	Mail configuration	N
JavaBeans resources	--	N
Other resources	Resource adapters conforming to the Connector 1.0 specifications or Connector 1.5 specifications	Y

Legend:

Y: Available

N: Not available

--: Not applicable

(2) Generating and initializing the connection pool

The connection pool is generated and initialized during the start processing of the resource. If the connection pool warming up functionality is enabled, a connection is generated at this point. If the connection pool warming up functionality is disabled, a connection is not generated when the resource starts, but is generated when the first connection request occurs.

The connection pool is generated as follows:

- When you use resource adapters conforming to the Connector 1.0 specifications, one connection pool is generated for the resources.
- When you use resource adapters conforming to the Connector 1.5 specifications, one connection pool is generated per connection definition.

(3) Terminating the connection pool

When you undeploy resources or when you terminate the J2EE server, you delete all the connections in the connection pool as well as the connection pool itself. Note that when the connection pool is terminated, all the connection-related transactions are assumed to have concluded.

(4) Destroying the connections with exceptions

If an error such as a database error occurs, the connections stored in the connection pool can no longer be used, and must be quickly discarded from the connection pool.

If an exception occurs in a connection, or in the processing of a product from the connection such as `Statement`, Application Server destroys the relevant connection from the connection pool when the connection is closed. However, if the conclusion of the local transaction terminates normally, the connection is determined to be normal and is not destroyed.

If an exception occurs in a connection or a product from the connection when a connection is maintained normally, and if you are using a global transaction, Application Server destroys the connection without returning the connection to the connection pool even if the transaction is concluded normally. Therefore, extra connections are generated and might affect the performance.

Note that when a transaction times out, the connections are destroyed from the connection pool regardless of the transaction type.

(5) Notes on using a connection pool

Note the following when you use a connection pool:

- Do not use the functionality for forceful disconnection (such as timeout) from the database server.
- The connections managed in a connection pool store the authentication information (such as user name and password) used for obtaining the connection, so you must take precautions according to the sign-on form.

For container-managed sign-on

Normally, single authentication information is used for a connection request to one connection pool, so no particular precautions need to be taken.

For component-managed sign-on

You must take precautions when a combination of multiple user names and passwords is used. There is one connection pool for each resource, so when multiple users use one resource, the multiple users share one connection pool. In this case, one user cannot use the connections equal to the value specified as the maximum connection pool value.

Furthermore, if there are no connections with matching authentication information among the unused connections in the connection pool, the following operations are performed depending on the total number of connections in the pool:

If the total number of connections reaches the specified maximum connection pool value, the unused connections are destroyed and new connections are generated.

If the total number of connections does not reach the specified maximum connection pool value, new connections are generated.

(6) Connection pooling operations

The following table describes the connection pooling operations of the resource adapters.

Table 3-47: Connection pool status and operations

Processing of the user application program	Connection pool state	Connection pool operations
Connection request	There are unused connections in the pool.	The connection that is unused for the longest time is selected and passed to the user application program. The status of the selected connection in the pool changes to in-use.
	There are no unused connections in the pool, and the total number of connections in the pool is less than the value of <code>MaxPoolSize</code> .	A new connection is established. The established connection is passed to the user application, and the status of the connection in the pool changes to in-use.
	There are no unused connections in the pool, and the total number of connections in the pool has reached the value of <code>MaxPoolSize</code> .	An exception is reported to the user application program, and the connection cannot be obtained. To obtain the connection again, specify <code>Retry Count</code> and <code>Retry Interval</code> .
	There are unused connections in the pool, but there are no connections with matching authentication information.	The following processing is implemented depending on the total number of connections in the pool: <ul style="list-style-type: none"> If the number of connections in the pool is less than the <code>MaxPoolSize</code> value A new connection is established and is passed to the user application. If the number of connections in the pool reaches the <code>MaxPoolSize</code> value Among the unused connections, the connections that are pooled first are destroyed, and then new connections are created and passed to the user application.
Releasing the connections	There are no errors in the released connections and the connections can be reused.	The status of this connection in the pool changes to unused.
	The released connection cannot be reused.	This connection is destroyed.
--	The connection pool warming up functionality is used.	When you start a resource adapter or start a server when the resource adapter is already running, connections are generated and pooled up to the value in <code>MinPoolSize</code> . To use the connection pool warming up functionality, specify <code>Warmup</code> .
	Waiting for a connection when connections deplete	If the maximum number of connections are pooled in the connection pool and there are no usable connections in the pool (connection depletion), you can set the connection request to pending. The pending connection request can obtain a connection as soon as a connection is released. To wait for a connection when connections deplete, specify <code>RequestQueueEnable</code> and <code>RequestQueueTimeout</code> .

Legend:

--: Not applicable

The following table lists the notes on using the connection pool warming up functionality.

Table 3–48: Notes on using the connection pool warming up functionality

Condition	Notes
Specifying the authentication information (such as user name and password) for container authentication [#]	<p>You must take precautions when you use a combination of multiple user names and passwords with the component-managed sign-on. There is one connection pool for each resource, so when multiple users use one resource, the multiple users share one connection pool. In this case, one user cannot use the connections equal to the value specified as the maximum connection pool value.</p> <p>In a container-managed sign-on, normally single authentication information is used for a connection request to one connection pool, so no particular precaution needs to be taken.</p>
Specifying <code>MinPoolSize</code>	<p>In the following cases, the pooled connections might be lesser than the value specified in <code>MinPoolSize</code>:</p> <ul style="list-style-type: none"> • When the connection sweeper executes the connection release processing • When you execute the <code>cjclearpool</code> (deleting the connections in the connection pool) command • When an error occurs in the connection <p>When a resource starts, the number of connections specified in <code>MinPoolSize</code> is generated; therefore, starting the server or resource adapter might take more time as compared to when the connection pool warming up functionality is not used.</p> <p>Also, at this time "<i>value-specified-in-bufSize x memory-for-the-number-of-connections-generated</i>" is allocated in the Java heap area.</p> <p>If you use the warming up functionality specifying a more than necessary large value in <code>MinPoolSize</code>, the Java heap is used up when you allocate the memory and <code>OutOfMemoryError</code> might occur.</p> <p>Therefore, set the value of <code>MinPoolSize</code> to no more than the maximum number of concurrent connections of the resource manager used.</p>

[#]: The connections managed with the connection pool store the authentication information (such as user name and password) for container authentication used during the operation of the warming up functionality.

3.14.2 Functionality available with connection pooling

When you use the connection pooling functionality, the following functionality is additionally available:

(1) Connection pool warming up

The Connection pool warming up is a functionality that pools the minimum number of connections defined in the connection pool settings, in advance, during the resource start processing executed when the server or resource adapter is started. Due to this, you can improve the response to the connection requests made immediately after you begin using the connection pool.

You set up the connection pool warming up functionality as a resource adapter attribute (property). For details on the content that can be specified in the resource adapter property definitions, see *3.14.10 Settings in the execution environment*.

(2) Connection count adjustment functionality

The connection count adjustment functionality reduces the unnecessary connections in the connection pool from the minimum specified connection pool value up to the maximum value in a phased manner. If you enable this functionality, the number of connections in the pool is gradually reduced up to a number appropriate for the actual operation results, so you can reduce the cost of generating the connections and save resources when the minimum specified connection pool value is exceeded.

You can also set a timeout value for the response time for deleting the connections during the connection count adjustment. If a server or network error occurs and no response returns from the resource, a response might not be returned even for the connection deletion processing. In this way, even if the response does not return from the resource, you can terminate the connection deletion processing and continue the rest of the processing by specifying a timeout value.

Operations of the connection count adjustment functionality

If you use the connection count adjustment functionality, the number of connections in the pool is adjusted at the maximum number of connections used concurrently during the period between the previous connection count adjustment and the next connection count adjustment. During the connection count adjustment, if the number of connections in the pool is greater than the maximum number of concurrently used connections, the connection deletion processing will operate. For example, if the number of connections in the pool is 8 currently, and the maximum number of connections used concurrently during the period between the previous connection count adjustment and the next connection count adjustment is 5, there are 3 more connections in the pool, so three connections are deleted from the connection pool, and the adjusted connection count becomes 5.

Note that the connection count adjustment functionality operates at regular intervals.

For details on the settings for the connection count adjustment functionality, see *3.14.10 Settings in the execution environment*.

Timeout in the connection deletion processing

You can set a timeout value for the response time for the connection deletion processing of the connection count adjustment functionality. Note that you can specify any value (default value is 5 seconds) as the timeout value for the connection deletion processing, in the keys specified for the J2EE server in the Easy Setup definition file.

However, if the maximum value of the connection pool is unlimited, the timeout value of the connection deletion processing is disabled.

Also, the connection management threads are used for the connection deletion processing timeout; therefore, if you specify a timeout for the connection deletion processing, a lot of memory is consumed compared to when the timeout is not set. If you want to set a timeout, estimate the required amount of memory appropriately.

For details on the connection management threads, see *3.15.1 Detecting the connection errors*.

(3) Connection sweeper operations

The connection sweeper functionality that is used to destroy the unused connections in the connection pool at regular intervals operates as follows:

- The connection sweeper operates after the lapse of the `SweeperInterval` value from the termination of the previous connection sweeper operation.
- The connection sweeper monitors the unused connections in the pool.

The unused connections, for which the time elapsed since the last usage is the `ConnectionTimeout` value or more, are destroyed. The connection sweeper does nothing in the case of the unused connections, for which the time elapsed since the last usage is less than the `ConnectionTimeout` value.

3.14.3 Connection sharing or association

The connection sharing and connection association are the shared connection functionality. By using the shared connection functionality, you can use the resources effectively and improve the performance. Enable the connection association functionality as and when required.

When you manage transactions using local transactions, you must access the resource, such as the database, with one resource connection only. If you use the shared connection functionality, you can access the resources with one resource connection only without any consideration, in the user application.

Similarly in a global transaction, the operation is optimized to a single-phase commit when only one resource connection participates in a transaction; therefore, you can reduce the cost of concluding the transaction.

(1) Physical connection and logical connection

A *physical connection* indicates the connection to a resource you want to connect to. Normally the J2EE components such as servlets and Enterprise Beans are operated by a container, instead of being operated directly. For example, `ManagedConnection` (`javax.resource.spi.ManagedConnection`) for a resource adapter.

A *logical connection* indicates a connection that the J2EE components such as servlets and Enterprise Beans operate directly. For example, in the case of a resource adapter, `javax.resource.cci.Connection` or a connection independently provided by a resource adapter.

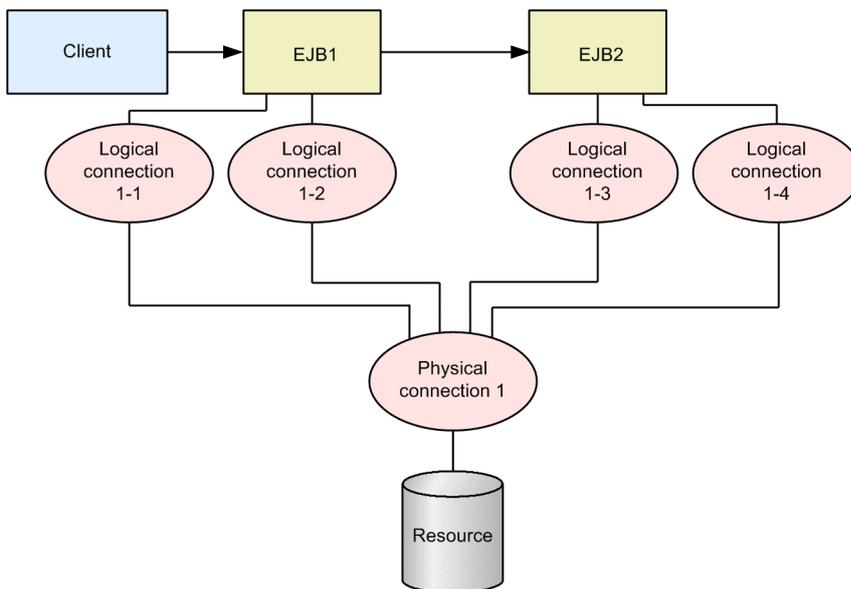
The relationship between a physical connection and logical connection is that, generally, a logical connection is generated from a physical connection. The physical connections are managed with a connection pool and the connection pool obtains and closes the physical connections.

For a connection request from the J2EE components such as servlets and Enterprise Beans, the connection pool generates and returns the logical connections from the physical connection. For a request to release connections, the connection pool closes the logical connections only, and manages the physical connections in the pool, without closing the connections.

(2) Connection sharing

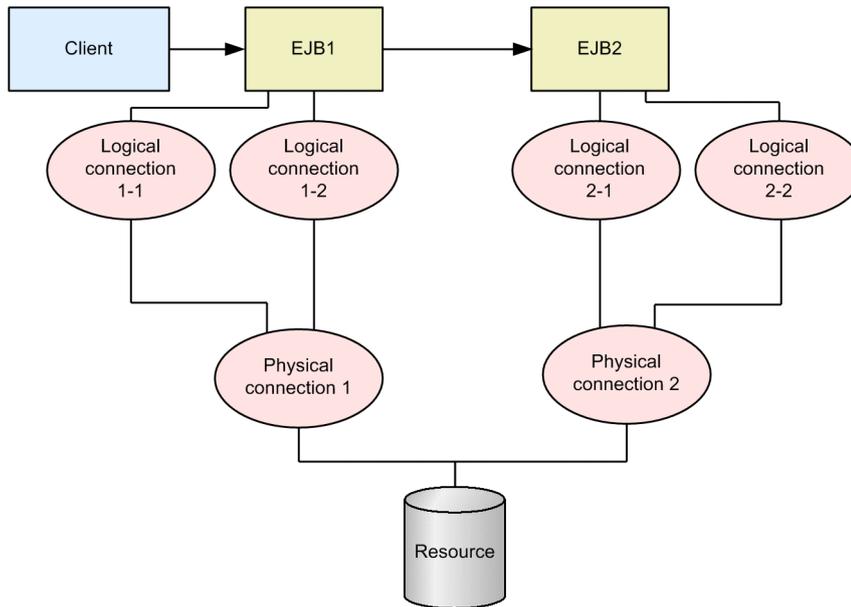
If the connection association functionality is not enabled, the connections are shared by the Application Server-managed transactions. The connection sharing in a transaction uses the resource connections most effectively. The following figure shows the connection sharing in a transaction.

Figure 3-32: Relationship between the logical connection and physical connection (connection sharing in a transaction)



When the connection association functionality is enabled and the connections are shared outside the Application Server-managed transactions, the connection sharing is performed by the instances of the J2EE components such as servlets and Enterprise Beans. The following figure shows the connection sharing in components.

Figure 3-33: Relationship between the logical connection and physical connection (connection sharing in components)



(a) Conditions for connection sharing

For connection sharing, all the following conditions must be satisfied:

- The connection sharing is performed on the same J2EE server.
- The connection sharing is performed for the same resource.
- The sign-on method and security information (user name and password) are the same.
- Shareable is specified in `<res-sharing-scope>` of the standard DD.
- The connection sharing is performed in the same Application Server-managed transaction #

#

You can also perform connection sharing outside the Application Server-managed transactions.

To enable connection sharing outside the Application Server-managed transactions, specify the settings by customizing the J2EE server properties. For details on customizing the J2EE server operation settings, see *3.14.10 Settings in the execution environment*.

Note that connection sharing is not performed if `NoTransaction` is specified as the transaction support level of the resource adapter.

For details on defining connection sharing, see *3.14.9 Definitions in cosminexus.xml*.

(b) Defining connection sharing

In the `<res-sharing-scope>` tag of the standard DD or `cosminexus.xml` of the servlets and Enterprise Beans, you specify whether the connections will be shared. You can specify the settings for each resource reference. The connection sharing is enabled by default. To disable connection sharing, specify `Unshareable` in `<res-sharing-scope>`.

For details on the J2EE application settings, see *3.14.9 Definitions in cosminexus.xml*.

(c) Notes

You cannot reuse `java.sql.Connection` between multiple transactions.

To use `java.sql.Connection` in a transaction, use the `getConnection()` method from `javax.sql.DataSource` and obtain the connections for each transaction.

Note that `java.sql.Connection` cannot be reused from inside the transaction to outside the transaction.

(3) Connection association

When you use a connection with persistence that has exceeded the transaction scope, connection sharing in a transaction is not available. In such a case, enable the connection association functionality.

The connection association switches the relationship between the logical connection and physical connection and implements resource access using one resource connection.

Figure 3-34: Relationship between the logical connection and physical connection (connection association)

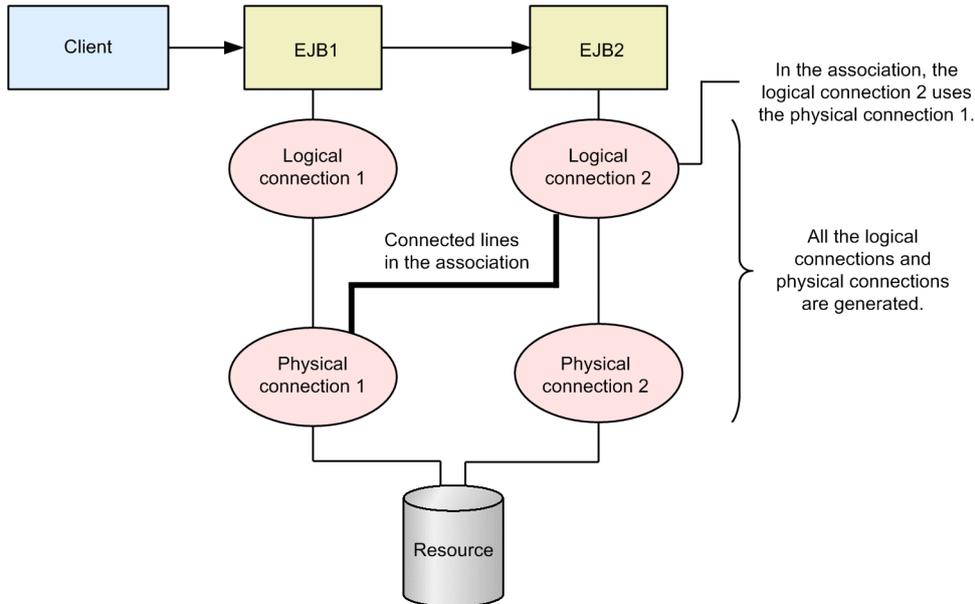
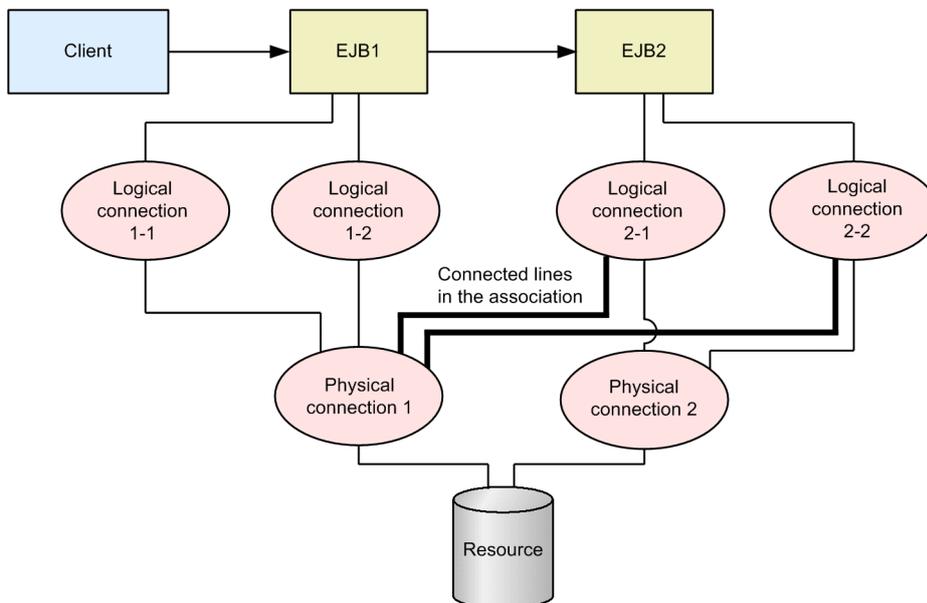


Figure 3-35: Relationship between the logical connection and physical connection (when connection association and connection sharing are used together)



(a) Conditions for connection association

For connection association, all the following conditions must be satisfied:

- The connection association functionality is enabled.

- The connection association is performed on the same J2EE server.
- The connection association is performed for the same resource.
- The sign-on method and security information (user name and password) are the same.
- `Shareable` is specified in `<res-sharing-scope>` of the standard DD. #
- The connection association is performed in the same Application Server-managed transaction

#

If `true` is set in the `ejbserver.connectionpool.association.enabledDespiteUnshareableSetting` key of `usrconf.properties`, the connection association is performed even if `Unshareable` is specified in `<res-sharing-scope>` of the standard DD.

This property has only been provided for downward compatibility.

(b) Defining connection association

The connection association is disabled by default.

Specify the settings for enabling connection association by customizing the J2EE server properties. For details on customizing the J2EE server operation settings, see *3.14.10 Settings in the execution environment*.

(c) Notes

The products from `java.sql.Connection` (example: `java.sql.Statement`) cannot be used exceeding the transaction scope.

3.14.4 Statement pooling

When you use DB Connector, you can use the pooling functionality that reuses JDBC APIs `java.sql.PreparedStatement` and `java.sql.CallableStatement`. By the statement pooling functionality, you can try to improve the performance when `PreparedStatement` and `CallableStatement` are used. Note that you specify the pool size for `PreparedStatement` and `CallableStatement` in the DB Connector settings. For details on the guidelines for specifying the pool size when a statement pooling is used, see *8.5.2 Using statement pooling* in the *uCosminexus Application Server System Design Guide*. Also, for details on defining the DB Connector properties, see *4.2.2 Defining the DB Connector properties* in the *uCosminexus Application Server Application Setup Guide*.

With the statement pooling functionality, the statement is initialized when the statement is reused. You specify the content to be initialized by customizing the J2EE server properties. For details on the settings of the statement pooling functionality, see *3.14.10 Settings in the execution environment*.

To use the statement pooling functionality, you must use the connection pooling functionality. Also, if global transaction is specified in the transaction support level, the statement pooling functionality might be unavailable depending on the HiRDB version.

! Important note

When you use HiRDB Type4 JDBC Driver, the pool size of `PreparedStatement` and `CallableStatement` is restricted. For details on the specifiable pool size, see *4.1.10 Properties that can be specified in the <config-property> tag set in DB Connector* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

(1) Preconditions

The following table describes the usage of statement pooling based on the mapping with the transaction support level, the use of connection pooling, and the database type used.

Table 3–49: Usage of statement pooling

Transaction support level	Connection pooling is used				Connection pooling is not used
	HiRDB#1	Oracle	SQL Server	XDM/RD E2	HiRDB/Oracle/SQL Server
No transaction (NoTransaction)	Y	Y	Y	Y	N
Local transaction (LocalTransaction)	Y	Y	Y	Y	N
Global transaction (XATransaction)	Y#2	Y	N	N	N

Legend:

Y: Can be used

N: Cannot be used

#1: Do not execute the definition SQL statement when you use the statement pooling functionality. When you execute the definition SQL statement, you cannot use the statement pooling functionality. Also, when you execute the definition SQL statement, you must set up PDDDLDEAPRP=YES as the HiRDB client environment variable.

#2: This can be used if the return value of DatabaseMetaData#supportsStatementPooling() of the JDBC is true.

Note that when you use XDM/RD E2, the statement pooling functionality is available only when you use XDM/RD E2 11-03 or later and HiRDB Type4 JDBC Driver 08-02 or later. For details on the settings of the statement pooling functionality, see 3.14.10 Settings in the execution environment.

(2) Statement pooling operations

This section describes the statement pooling operations specified in the resource adapter configuration properties.

The following table describes the operations of the statement pooling functionality.

Table 3–50: Statement pool states and operations

User application program processing	Statement pool state	Statement pool operations
Requesting the generation of PreparedStatement and CallableStatement	There are unused PreparedStatement and CallableStatement in the pool	Either of the unused PreparedStatement or CallableStatement are selected in the pool and passed to the user application program. The status of the selected PreparedStatement or CallableStatement in the pool changes to in-use.
	There are no unused PreparedStatement and CallableStatement in the pool, and the total number of PreparedStatement and CallableStatement in the pool is less than the value of PreparedStatementPoolSize and CallableStatementPoolSize	New PreparedStatement and CallableStatement are generated. The generated PreparedStatement and CallableStatement are passed to the user application, and the status of PreparedStatement and CallableStatement in the pool changes to in-use.
	There are no unused PreparedStatement and CallableStatement in the pool, and the total number of PreparedStatement and CallableStatement in the	PreparedStatement and CallableStatement with the oldest time stamp# are deleted from the pool, and then new PreparedStatement and CallableStatement are generated. The generated PreparedStatement and

User application program processing	Statement pool state	Statement pool operations
Requesting the generation of <code>PreparedStatement</code> and <code>CallableStatement</code>	pool is the value of <code>PreparedStatementPoolSize</code> and <code>CallableStatementPoolSize</code> or more	<code>CallableStatement</code> are passed to the user application, and the status of <code>PreparedStatement</code> and <code>CallableStatement</code> in the pool changes to in-use.
Releasing <code>PreparedStatement</code> and <code>CallableStatement</code>	--	<code>PreparedStatement</code> and <code>CallableStatement</code> are returned to unused in the pool.

Legend:

--: Not applicable

#: The time stamps for `PreparedStatement` and `CallableStatement` in the pool are updated at the following times:

- When the new `PreparedStatement` and `CallableStatement` are added in the pool
- When the status of `PreparedStatement` and `CallableStatement` in the pool changes to in-use

(3) Notes on using the statement pooling functionality

The following table describes the notes on using the statement pooling functionality.

Table 3-51: Notes on using the statement pooling functionality

Condition	Notes
--	<ul style="list-style-type: none"> • Compared to when the statement pooling functionality is not used, the memory is only consumed for the number of pooled <code>PreparedStatement</code> and <code>CallableStatement</code>. For details on the memory used for each statement, see the documentation for JDBC driver in use. • There is a statement pool for each connection in the connection pool, so the maximum number of pooled <code>PreparedStatement</code> and <code>CallableStatement</code> becomes $\text{MaxPoolSize} \times (\text{PreparedStatementPoolSize} + \text{CallableStatementPoolSize})$.
--	<ul style="list-style-type: none"> • If you use the statement pooling functionality together with the connection count adjustment functionality or the connection error detection functionality, the unused connections removed from the connection pool are not counted as the number of pooled <code>PreparedStatement</code> and <code>CallableStatement</code> might temporarily exceed the <i>maximum-connection-pool-value</i> \times <code>PreparedStatementPoolSize</code>, and <i>maximum-connection-pool-value</i> \times <code>CallableStatementPoolSize</code>. • If you use the statement pooling functionality to reuse the statements, the value specified for <code>setMaxFieldSize</code> might not be initialized. The value is not initialized when all of the following conditions are satisfied: <ul style="list-style-type: none"> - DB Connector supporting Oracle JDBC Thin Driver is used - The <code>setMaxFieldSize</code> method is used to change the value of <code>java.sql.PreparedStatement</code> or <code>java.sql.CallableStatement</code>
Connecting to Oracle or SQL Server	<ul style="list-style-type: none"> • When you use the statement pooling functionality and the connection error detection functionality together One <code>PreparedStatement</code>, used for detecting the connection errors, is pooled for each connection. Therefore, when you determine <code>PreparedStatementPoolSize</code> and <code>CallableStatementPoolSize</code>, add <code>PreparedStatement</code> for detecting the connection errors in the estimation of the number of resources, and set <code>CallableStatementPoolSize</code> to a value that is less than the maximum number of statements. # • When you use the statement pooling functionality and the connection pool warming up functionality together

Condition	Notes
Connecting to Oracle or SQL Server	<p>When the connections are generated and pooled, one <code>PreparedStatement</code>, generated with the SQL statement used for detecting the connection errors, is pooled for each connection.</p> <ul style="list-style-type: none"> • When you use the statement pooling functionality and the result set holding functionality together <p>When you use the result set holding functionality with an Oracle connection, specify the result set holding functionality in the argument of the <code>Connection.prepareStatement()</code> method or <code>prepareCall()</code> method. You cannot specify the result set holding functionality with the <code>Connection.setHoldability()</code> method.</p>

Legend:

--: Not applicable

#

When you use the connection error detection functionality, set `CallableStatementPoolSize` to a value less than the maximum number of statements.

If you use the connection error detection functionality, the process of detecting the connection errors is executed when the number of pooled `CallableStatement` reaches the maximum number of statements. At this time, if `CallableStatementPoolSize = maximum-number-of-statements` is set, the maximum number of resources that can be used in one JDBC driver connection is exceeded, so an exception occurs. If an exception occurs, an error is determined; therefore, that connection is deleted from the connection pool and simultaneously the statement pool is also destroyed. In other words, using the statement pooling functionality becomes meaningless.

3.14.5 Light transaction

The light transaction is a functionality that provides an optimum environment for the local transactions. As a result, an excellent local transaction performance is obtained. The light transaction can be applied only when local transactions are used. If the light transaction functionality is enabled, the EJBs can be invoked remotely in a transaction only when the invocation destination is BMT.

The light transaction functionality is enabled by default. An error occurs if you use a global transaction when the light transaction functionality is enabled. Therefore, when you want to use a global transaction, you must disable the light transaction functionality.

You specify the settings for enabling the light transaction functionality by customizing the J2EE server properties. For details on customizing the settings for the J2EE server operations, see *3.4.12 Settings in the execution environment*.

3.14.6 In-process transaction service

With Application Server, you can start a transaction service in the J2EE server in-process. If you start the transaction service in the in-process, the transaction processing is optimized so that the processing is executed in the J2EE server process, so a high-performance system can be implemented.

The in-process transaction service supports the OTS 1.3 specifications provided with CORBA.

3.14.7 Caching the DataSource objects

When you access a database from a J2EE application, the JNDI interface is used to request the search of the `javax.sql.DataSource` type objects (hereafter, `DataSource` objects). With the default J2EE server operations, when the applicable `DataSource` is registered, the instances of the `DataSource` object are generated and returned to the application.

At this time, if you use `DataSource` object caching, the J2EE server caches the instances of the registered `DataSource` object, and returns the same instances for a search request. If you use `DataSource` object caching, the search time for the `DataSource` objects is shortened.

You specify the settings for caching the `DataSource` objects by customizing the J2EE server properties. For details on customizing the settings for the J2EE server operations, see *3.14.10 Settings in the execution environment*.

(1) Preconditions

The `DataSource` object caching functionality is enabled when `DataSource` is looked up with the business method of the Enterprise Beans and the service method of the servlets/JSPs, and the relevant instance is not stored in the member variable.

The functionality is not enabled when the `DataSource` object looked up with the business method and service method is stored in the member variable and is also used with other methods. The functionality is also not enabled when the `DataSource` object looked up in the initialization methods such as the `ejbCreate` method of Enterprise Bean and the `init` method of servlets/JSPs is stored in the member variable and is used in the business method and service method.

(2) Notes

If you use a resource adapter such as that described below when the `DataSource` object caching functionality is used, the `<resource-ref>` definition in the property file might not be enabled during operations:

- You use a resource adapter in which multiple `<resource-ref>` tags are defined for the same resource from the same J2EE component, and a different value is specified in the `<res-sharing-scope>` tag and `<res-auth>` tag existing in each `<resource-ref>` tag.

To define multiple `<resource-ref>` tags with the same J2EE component, make sure you use another resource (resource adapter with a different deployment unit).

3.14.8 Optimizing the container-managed sign-on for DB Connector

DB Connector supports container-managed sign-on and component-managed sign-on.

The features of each method are as follows:

In container-managed sign-on

When you use a container-managed sign-on, the user name and password set for DB Connector is used to access the database.

In component-managed sign-on

When you use a component-managed sign-on, the user name and password passed to the `getConnection` method of the connection factory is used to access the database.

Note that when you use `DBConnector_DABJ_XA.rar` (when you use a global transaction), only the user name and password specified in the `XAOpen` string is enabled. Therefore, you cannot perform a component-managed sign-on where you specify the user name and password with the `getConnection` method.

If you use the sign-on optimization functionality when you perform a container-managed sign-on, the container-managed sign-on operations are optimized and the performance of obtaining a database connection improves.

You specify the settings for optimizing the container-managed sign-on for DB Connector by customizing the J2EE server properties. For details on customizing the settings for the J2EE server operations, see *3.14.10 Settings in the execution environment*.

! Important note

- Use the container-managed sign-on optimization when you do not want to perform a component-managed sign-on. If you perform the container-managed sign-on optimization, the component-managed sign-on cannot be used.
- When you use the connection pool clustering functionality, you cannot mix container-managed sign-on and component-managed sign-on in one clustered connection pool.

3.14.9 Definitions in cosminexus.xml

From among the functionality for performance tuning, you define connection sharing in the `<rar>` tag of `cosminexus.xml`. The following table describes the definition of the functionality for performance tuning in `cosminexus.xml`.

Table 3-52: Definition of the functionality for performance tuning in `cosminexus.xml`

Specified tag	Settings
<code><resource-external-property>-<res-sharing-scope></code> tag	Specify whether the obtained connections will be shared.

For details on `cosminexus.xml`, see 2. *Application Property File (cosminexus.xml)* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

3.14.10 Settings in the execution environment

To use the functionality for performance tuning, you must specify the settings for the J2EE server, resource adapters, and J2EE applications.

(1) J2EE server settings

You implement the J2EE server settings with the Easy Setup definition file. Specify the performance tuning definition in the `<configuration>` tag of the logical J2EE server (`j2ee-server`) in the Easy Setup definition file.

The following table describes the definition of the functionality for performance tuning in the Easy Setup definition file.

Table 3-53: Definition of the functionality for performance tuning in the Easy Setup definition file

Item	Specified parameters	Settings
Enabling connection sharing outside the Application Server-managed transactions	<code>ejbserver.connectionpool.sharingOutsideTransactionScope.enabled</code>	Specifies the connection sharing operations to be performed when connections are obtained multiple times outside the Application Server-managed transactions.
Connection association	<code>ejbserver.connectionpool.association.enabled</code>	Specifies whether to use the connection association functionality.
	<code>ejbserver.connectionpool.association.enabledDespiteUnshareableSetting</code>	Specifies whether to use the connection association functionality even when non-sharing [#] of resources is set in the system properties.
Statement initialization in the statement pooling functionality	<code>ejbserver.connector.statementpool.clear.backcompat</code>	Specifies the content of the statement to be initialized when a statement is reused in the statement pooling functionality.
Caching the DataSource objects	<code>ejbserver.jndi.cache.reference</code>	Specifies whether to enable DataSource object caching.
Optimizing the container-managed sign-on for DB Connector	<code>ejbserver.connectionpool.applicationAuthentication.disabled</code>	Specifies whether to enable the optimization

Item	Specified parameters	Settings
Optimizing the container-managed sign-on for DB Connector	<code>ejbserver.connectionpool.applicationAuthentication.disabled</code>	functionality of container-managed sign-on.
Timeout in error detection	<code>ejbserver.connectionpool.validation.timeout</code>	Specifies the timeout value for the connection error detection functionality and the timeout value for deleting the connections by using the connection count adjustment functionality.

#: When `Unshareable` (resources are not share) is specified in the `<res-sharing-scope>` tag of the WAR property file, Entity Bean property file, Session Bean property file, or Message-driven Bean property file.

For details on the light transaction settings, see *3.4.12 Settings in the execution environment*.

Note that the in-process transaction service functionality is used with the default settings. You need not specify the settings.

For details on the Easy Setup Definition file and parameters, see *4.6 Easy Setup Definition file* in the *uCosminexus Application Server Definition Reference Guide*.

(2) Resource adapter settings

Use the server management commands and property files to specify the resource adapter settings in the execution environment. Use the HITACHI Connector Property file to define the functionality for performance tuning.

The following table describes the definition of the functionality for performance tuning in the HITACHI Connector Property file.

Table 3-54: Definition of the functionality for performance tuning in the HITACHI Connector Property file

Category	Items	Specified tags	Settings
Connection pooling	Minimum and maximum number of connections	<code>MinPoolSize</code> and <code>MaxPoolSize</code> in the <code><property></code> tag	Specifies the minimum and maximum number of connections to be pooled in the connection pool.
	Connection pool warming up	<code>Warmup</code> in the <code><property></code> tag	Specifies the use of the connection pool warming up functionality.
	Connection management threads	<code>NetworkFailureTimeout</code> in the <code><property></code> tag	Specifies the use of the connection management threads. When you use the connection management threads, a timeout is set for the connection error detection functionality and the connection count adjustment functionality. The timeout value is fixed at 5 seconds with version 07-60 or earlier. With version 08-00 or later, you can specify any timeout value (default value is 5 seconds).
	Connection count adjustment functionality	<code>ConnectionPoolAdjustmentInterval</code> in the <code><property></code> tag	Specifies the interval at which the connection count adjustment functionality will be operated. Note that when a timeout is set for the connection count adjustment functionality, the use of connection management threads is enabled with <code>NetworkFailureTimeout</code> . ^{#1}

Category	Items	Specified tags	Settings
Statement pooling	Pool size of PreparedStatement _{t#2}	PreparedStatementPool Size in the <config-property> tag	Specifies the pool size of PreparedStatement.
	Pool size of CallableStatement _{t#2}	CallableStatementPool Size in the <config-property> tag	Specifies the pool size of CallableStatement.

#1: The key is the same as that for the timeout in the connection error detection functionality. Therefore, when you use a timeout for the connection error detection functionality, a timeout is also used for the connection count adjustment functionality.

#2: In XDM/RD E2 version 11-01 or earlier, the statement pooling functionality cannot be used; therefore, so specify 0 for these properties.

For details on the HITACHI Connector Property file, see *4.1 HITACHI Connector Property file* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

(3) J2EE application settings

You implement the J2EE application settings in the execution environment using the server management commands and property files. To define the functionality for performance tuning, use the WAR property file, Session Bean property file, Entity Bean property file, or Message-driven Bean property file.

The tags specified in these property files correspond to `cosminexus.xml` or `DD`. For details on the definitions in `cosminexus.xml`, see *3.14.9 Definitions in cosminexus.xml*.

3.15 Functionality for fault tolerance

This section describes the functionality for fault tolerance.

The following table describes the organization of this section.

Table 3-55: Organization of this section (Functionality for fault tolerance)

Category	Title	Reference location
Explanation	Detecting the connection errors	3.15.1
	Waiting for a connection when connections deplete	3.15.2
	Retrying to obtain a connection	3.15.3
	Displaying the connection pool information	3.15.4
	Clearing the connection pool	3.15.5
	Automatically closing the connections	3.15.6
	Connection sweeper	3.15.7
	Transaction timeout and statement cancellation	3.15.8
	Transaction recovery	3.15.9
	Output of the SQL statement for troubleshooting	3.15.10
	Automatically closing the objects	3.15.11
Implementation	Definitions in <code>cosminexus.xml</code>	3.15.12
Settings	Settings in the execution environment	3.15.13

Note: The functionality-specific explanation is not available for "Operations".

3.15.1 Detecting the connection errors

When you use connection pooling, an error connection might be returned for a connection request from a user program if a resource is down or a network error occurs. If you use the connection error detection functionality, you can check whether an error has occurred in the pooled connections and make sure that an error connection is not returned unless otherwise necessary.

You can use the connection error detection functionality together with the connection retry functionality. In this case, if a connection error is detected during a connection request from the user program, you can try to obtain a new connection and return a connection to the user program as soon as the error is restored.

Note that creating and returning a new connection is only enabled when the time for detecting the connection errors is set to "when the connection is requested".

For the times at which the connection errors are detected, see (2) *Timing at which error detection is implemented*.

(1) Preconditions

The following table describes the usability of the connection error detection functionality.

Table 3-56: Usability of the connection error detection functionality

Resource	Connection method	Usability
Database	DB Connector	Y
Database queue	DB Connector for Cosminexus RM and Cosminexus RM	Y
OpenTP1	uCosminexus TP1 Connector	N ^{#1}

Resource	Connection method	Usability
OpenTP1	TP1/Message Queue - Access	N ^{#1}
SMTP server	mail configuration	N
JavaBeans resource	--	N
Other resources	Resource adapters conforming to the Connector 1.0 specifications or Connector 1.5 specifications	R ^{#2}

Legend:

- Y: Available
- N: Not available
- R: Depends on conditions
- : Not applicable

#1: The connection error detection functionality provided by Application Server is not available, but uCosminexus TP1 Connector or TP1/Message Queue - Access provide functionality similar to the connection error detection functionality.

#2: Available when you are using the resource adapters conforming to the Connector 1.5 specifications and when the resource adapter implements the `getInvalidConnections` method of the `ValidatingManagedConnectionFactory` interface.

(2) Timing at which error detection is implemented

You can choose one of the timings listed in the table as the timing for detecting the connection errors.

Table 3-57: Timing for detecting the connection errors

Error detection timing	Contents
When a connection is requested	Error detection is implemented every time a connection is obtained. In this case, the response performance of the connection request deteriorates, but the probability of obtaining an error connection will lower. This is useful when the frequency of connection requests is less and obtaining an error connection is not permitted.
Regular intervals	Error detection is implemented at fixed time intervals. By lengthening the error detection interval to some extent, you can reduce the performance deterioration caused by the error detection processing. However, the probability of obtaining an error connection is higher. This is useful when the frequency of connection requests is high and obtaining an error connection is permitted to some extent.

Note that by default error detection is implemented for a connection request. You can also specify settings to not detect the connection errors.

You specify the settings to switch between the enabling and disabling of connection error detection and the time at which errors are detected as the resource adapter properties. For details on the resource adapter settings, see *3.15.13 Settings in the execution environment*.

(3) Timeout in error detection

You can specify a timeout for the response time to detect the connection errors.

If a server error or network error occurs and no response returns from the resource, a response might not be returned even for the execution of the connection error detection. By setting a timeout, the connection check is terminated and the processing can be continued even if no response returns from the resource. Note that you can specify any value for the timeout in error detection in the key specified for the J2EE server in the Easy Setup definition file (the default value is 5 seconds).

The connection management threads are used to operate the timeout for detecting the connection errors. The connection management threads are created in the system according to the maximum number of connections in the connection pool, when the resource adapter starts. The relationship between the number of connection management threads and the maximum number of connections in the connection pool is as follows:

$$\text{Number-of-connection-management-threads} = \text{maximum-number-of-connections-in-the-connection-pool} \times 2$$

Therefore, if you set a timeout for detecting the connection errors, a lot of memory is consumed compared to when the timeout is not set. Estimate the required amount of memory appropriately.

Also, if the settings for the maximum number of connections in the connection pool are unlimited, a message is output and the timeout for detecting the connection errors is disabled.

For notes on enabling the timeout for detecting the connection errors, see (5) *Notes*.

You specify the settings for enabling and disabling the timeout for detecting the connection errors by customizing the resource adapter properties. For details on the resource adapter property settings, see 3.15.13 *Settings in the execution environment*.

(4) Checking operations

This section describes the operations when the detection of connection errors is implemented for a connection request, and when the detection of connection errors is implemented at regular intervals.

- **When the detection of connection errors is implemented for a connection request**

The operations when the detection of connection errors is implemented for a connection request are as follows:

1. The unused connections are obtained from a connection pool.
2. The connection check method is used to check if there are any errors in the connections.
 - If the timeout for detecting the connection errors is disabled, this check is implemented by extending the connection request.
 - If the timeout for detecting the connection errors is enabled, this check is implemented with the connection management threads. Due to an error such as a server error or network error, when the response from the check method does not return within the timeout value, a connection error is determined to have occurred. At this time, a message is output.
 - Note that if the connection check method is not implemented for a resource adapter, the connection state is not checked.
3. If an error occurs in the connection, the connection used for the check is destroyed and a new connection is created and returned to the user program.
 - If there is no error in the connection, the connection used for the check is returned to the user program.

- **When the detection of connection errors is implemented at regular intervals**

The operations when the detection of connection errors is implemented at regular intervals are as follows:

1. One unused connection is obtained from the connection pool and checked for validity.
 - However, if the unused connections are not pooled, the connections are not checked.
2. If the timeout for detecting the connection errors is disabled, the connection check method is used to check if there are any errors in the connection, at regular intervals.
 - If the timeout for detecting the connection errors is enabled, the connection management threads are used to implement a check at regular intervals. Due to an error such as a server error or network error, when the response from the check method does not return within the timeout value, a connection error is determined to have occurred. At this time, a message is output.
 - Note that if the connection check method is not implemented for a resource adapter, the connection state is not checked.
3. If there is an error in the connection and if the timeout for detecting the connection errors is disabled, the connection used for the connection check is destroyed, and the in-use connection in the connection pool is marked as non-reusable. Also, the unused connection is destroyed.
 - If the timeout for detecting the connection errors is enabled, the connection used for the connection check is destroyed and the connection management threads mark the in-use connection in the connection pool as non-reusable. Also, the unused connection is removed from the connection pool and destroyed.
 - If there is no error in the connection, the connection used for the connection check is returned to the connection pool.

(5) Notes

The notes on the error detection functionality are as follows:

(a) Notes on using the resources in a parallel server configuration

When the detection of connection errors is implemented at regular intervals, a sample check is performed for the connections in the connection pool, so even if errors occur on some of the servers, the errors might not be detected.

(b) Notes on enabling the timeout for detecting the connection errors

The notes on enabling the timeout for detecting the connection errors are as follows:

- If you set a timeout for the detection of connection errors, error detection threads are generated in the system according to the number of connections in the connection pool. Therefore, if you set a timeout for the detection of connection errors, note that a lot of memory is consumed compared to when the timeout is not set.
You can only create the error detection threads equal to twice the maximum number of connections in the connection pool. Estimate the required amount of memory appropriately.
- If the system operations are continued when the errors, such as the server error or network error, occur repeatedly, the connection management threads that are executing the connection check method continue to increase and all the connection management threads prepared in the system might be used up. If no connection management threads are available, a message is output and the connection request results in an error.
- When the detection of connection errors is implemented, the unused connections removed from the connection pool are not counted as the number of connections in the connection pool. Therefore, the total of the connections in the connection pool and the unused connections removed from the connection pool might temporarily exceed the maximum number of connections in the connection pool.

(c) Notes on using resources that cannot use the error detection functionality

Set the following parameter in the HITACHI Connector Property file so that the connection management threads are not used for resources that cannot use the error detection functionality:

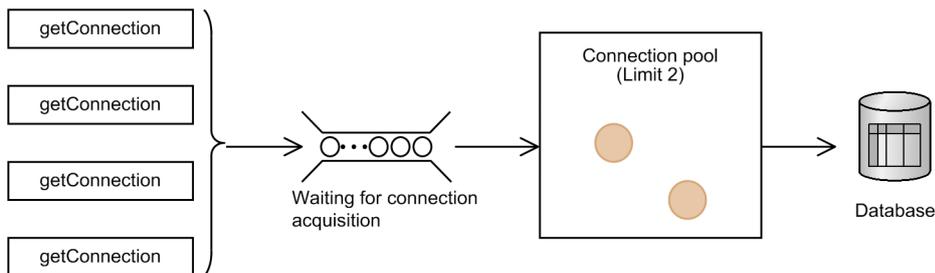
- Timeout for detecting the connection errors
`ValidationType=0` in `<property>` tag

3.15.2 Waiting for a connection when connections deplete

The state when the connections are pooled up to the maximum number specified for the connection pool and there are no usable connections in the pool is called *connection depletion*.

When the connections deplete, you can set the connection requests to a pending state. The pending connection requests can obtain a connection as soon as a connection is released. Due to this, connections can be obtained efficiently when the connections are depleted. The following figure shows the process of waiting to obtain a connection when connections deplete.

Figure 3-36: Waiting for a connection when connections deplete



In this figure, the maximum number of the connection pool is 2. Therefore, even if there are connection requests from four `getConnection` methods, only up to two requests can be processed. The first two connection requests can obtain the connections because there are connections in the connection pool. The remaining two are in the connection depletion state, so these requests enter the connection pending queue and wait for a connection to be released.

Note that you can specify settings to wait to obtain a connection when you use a connection pool.

To wait to obtain a connection, you must set up the following content as the resource adapter properties:

- Whether to wait to obtain a connection when the connections deplete
- Maximum connection pending time

For details on the resource adapter settings, see *3.15.13 Settings in the execution environment*.

(1) Operations for connection depletion

When you specify settings to wait to obtain a connection, the connection requests during connection depletion enter a pending state. If the waiting time for obtaining a connection request exceeds the maximum waiting time, an exception will be thrown in the user program.

Also, the connection requests are restarted at one of the following times:

- When a connection is released and an unused connection is available
- When a connection is destroyed and the number of connections is less than the maximum number

Note that if an error occurs after the connection requests are restarted, the retry processing is executed if the connection retry functionality is enabled.

3.15.3 Retrying to obtain a connection

Connection retry is a functionality that automatically retries to obtain a connection when there are no available connections in the connection pool and when the establishment of a physical connection fails. By using the connection retry functionality, if an attempt to obtain a connection fails, you need not use the user program to retry to obtain a connection.

You can retry to obtain a connection when one of the following conditions is satisfied:

- All the following conditions are satisfied and there are no available connections (unused connections) in the connection pool:
 - The line 'waiting for a connection when connections deplete' is disabled.
 - The total number of connections in the connection pool has reached the maximum number of connections to be pooled.
- When a physical connection cannot be established due to any of the following conditions:
 - The total number of connections in the connection pool has reached the maximum number of connections to be pooled, and there are no unused connections with matching authentication information.
 - The total number of connections in the connection pool has not reached the maximum number of connections to be pooled.
 - Connection pooling is disabled.

Note that if a connection cannot be obtained even after retrying, an exception is reported to the application program and the process of obtaining a connection fails.

For the operations to be performed when the connection pool depletes, follow the description in *3.15.2 Waiting for a connection when connections deplete*.

To implement the connection retry functionality, you must set up the following content as the resource adapter properties:

- **Retry count**
Specifies the number of times the process will be retried.
- **Retry interval**
Specifies the interval until the next retry attempt in seconds.

Note that if the retry count and retry interval is increased, the requests might have to wait when the processing to obtain connections accumulate.

For details on the resource adapter settings, see *3.15.13 Settings in the execution environment*.

(1) Preconditions

The following table describes the preconditions for using the connection retry functionality.

Table 3-58: Using the connection retry functionality

Resource	Connection method	Usability
Database	DB Connector	Y
Database queue	DB Connector for Cosminexus RM and Cosminexus RM	Y
OpenTP1	uCosminexus TP1 Connector	Y
	TP1/Message Queue - Access	Y
SMTP server	Mail configuration	N
JavaBeans resource	--	N
Other resources	Resource adapters conforming to the Connector 1.0 specifications or Connector 1.5 specifications	Y

Legend:

Y: Available

N: Not available

--: Not applicable

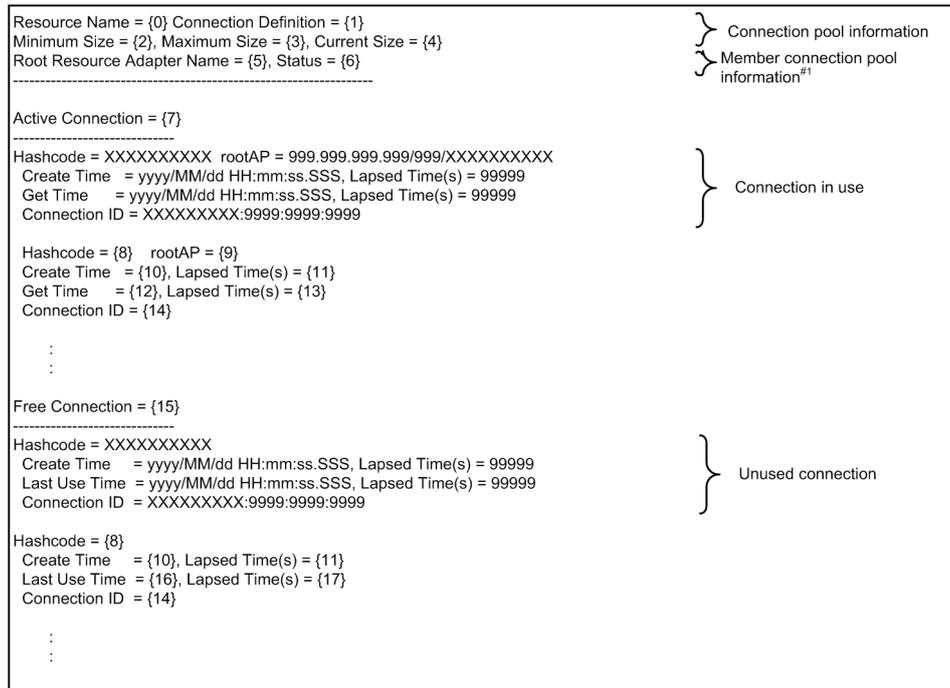
3.15.4 Displaying the connection pool information

You can use the `cjlistpool` command to display the information on the connections in the connection pool. You can also execute the `cjlistpool` command for the member resource adapters when the connection pool clustering functionality is used.

For the command details, see *cjlistpool (Displaying the list of connection pools)* in the *uCosminexus Application Server Command Reference Guide*. Also, for details on the connection pool clustering functionality, see *3.17 Functionality for connection pool clustering*.

An example of displaying the connection information of a member resource adapter is as follows. Note that if the connection pool is disabled, the connection information is not displayed.

Figure 3-37: Example of displaying the connection information of a member resource adapter



Legend:

{0}: Resource name	{10}: Connection generation time
{1}: Connection definition identifier	{11}: Connection generation lapsed time (unit: second)
{2}: Minimum value of the number of connections	{12}: Connection acquisition time
{3}: Maximum value of the number of connections	{13}: Connection use lapsed time (unit: second)
{4}: Current value of the number of connections	{14}: Connection ID ^{#4}
{5}: Root resource adapter name ^{#2}	{15}: Number of unused connections
{6}: State of the connection pool ^{#3}	{16}: Connection return time ^{#4}
{7}: Number of connections in use	{17}: Unused connection time (unit: second)
{8}: Connection hash code	
{9}: Root AP information	

#1

Member connection pool information is output only in the case of the member resource adapter.

#2

When the root resource adapter does not exist or has not started, "N/A" is output.

#3

The same state as the `cjlistar` command is output. Furthermore, when the state of the connection pool can not be acquired or the state is invalid, "invalid" is output.

#4

When the connection ID, connection return time, or connection definition identifier are not there, "N/A" is output.

3.15.5 Clearing the connection pool

If a connection is lost because a database server is down, the pooled connections remain in the connection pool. You can delete these connections using the `cjclearpool` command.

For the command details, see *cjclearpool (Deleting the connections in the connection pool)* in the *uCosminexus Application Server Command Reference Guide*.

3.15.6 Automatically closing the connections

The connection of a resource that is opened with the user program must be closed with the user program. With this functionality, the Web container and EJB container automatically close a connection if the user program cannot close the connection due to reasons such as the occurrence of an exception.

You can enable and disable the auto-close connection functionality based on the functionality. The following table describes the methods and switching of auto-close connection.

Table 3-59: Auto-close connection method and switching of the functionality

Auto-close method	Switching between enable and disable	Remarks
Web container-based auto-close connection	Y ^{#1, #2}	--
EJB container-based auto-close connection	N ^{#1}	Always enabled.

Legend:

Y: Can be switched

N: Cannot be switched

--: Not applicable

#1: The target resource is the resource adapter connection.

#2: The Web container-based auto-close connection functionality is enabled by default. You specify the settings for disabling the functionality by customizing the J2EE server properties. For details on customizing the settings for the J2EE server operations, see *3.15.13 Settings in the execution environment*.

(1) Web container-based auto-close connection

From among the JDBC connections that are obtained or opened in the servlet and JSP service methods, the connections that are not closed when the method execution finishes are automatically closed by the J2EE container. Due to this, the connections that are open and accumulating can be closed automatically.

Note that the auto-close connection functionality is also enabled when the connections are obtained or used by other methods in a servlet, or by the external classes.

The Web container-based auto-close connection is executed when the servlet and JSP service methods are completed.

(2) EJB container-based auto-close connection

With this functionality, the EJB container automatically closes and releases a connection if the user cannot close a resource connection obtained in the EJB, due to reasons such as the occurrence of an exception.

The EJB container-based auto-close connection is implemented at the following times:

- When the EJB is destroyed
- When a business method is returned with the Stateless Session Bean
- When the Stateful Session Bean is passivated
- When a business method is returned with the Singleton Session Bean

Note that auto-close connection is implemented at the following times with the Singleton Session Bean or Stateless Session Bean specifying the `@Asynchronous` annotation:

- When the Singleton Session Bean or Stateless Session Bean instances are destroyed
- When the execution of the method specifying the `@Asynchronous` annotation in the Singleton Session Bean or Stateless Session Bean is completed

(3) Method that confirms the execution of auto-close connection

You can use the message log or PRF trace to confirm that the auto-close connection functionality was executed.

We recommend that you use the message log to confirm the execution of the functionality when the performance need not be considered, such as during testing and troubleshooting; and the PRF trace when you want to confirm the execution of the functionality during real operations.

(a) Method of confirming with the message log

Set the log level to `Warning`.

When auto-close connection is executed, `KDJE31010-W` is output to the J2EE server message log.

For details on how to set the log level, see *3.3.6 Settings for collecting the J2EE server log in the uCosminexus Application Server Maintenance and Migration Guide*.

(b) Method of confirming with the PRF trace

Set the PRF trace collection level of the JCA container functionality layer to `Detail`.

When auto-close connection is executed, the following event IDs are output to the PRF trace file:

- 0x8B84
- 0x8B85

For details on how to set the PRF trace collection level, see 3.3.6 *Settings for collecting the J2EE server log* in the *uCosminexus Application Server Maintenance and Migration Guide*. Also, for details on how to set the PRF trace collection level for each functionality layer, see *cpflevel (Displaying and changing the PRF trace collection level)* or *cpfstart (Starting the PRF daemon)* in the *uCosminexus Application Server Command Reference Guide*.

(4) Notes

- The connections that are obtained using the threads generated by the user (user thread) in Servlets/ JSPs are not closed automatically.
- If there are unconcluded transactions when a connection is auto-closed, the transaction is rolled back. Also, if there are unconcluded transactions when a connection is closed with a user program, the transactions are committed implicitly.

3.15.7 Connection sweeper

Connection sweeper is a functionality that monitors the connection pool at regular intervals, and destroys the unused connections that are not used for a fixed period of time. If the time elapsed since the applicable connection was last used is equal to the specified time or more, the connection will be destroyed. This functionality can be used in cases when there is a gateway, such as a firewall, between Application Server and database server, and the connection is disconnected at a fixed time.

The connection sweeper functionality is disabled by default. Note that you specify the settings for the connection sweeper functionality as the resource adapter properties. For details on the resource adapter settings, see 3.15.13 *Settings in the execution environment*.

Note that when you use the connection sweeper functionality, specify settings so that the total value of `ConnectionTimeout` and `SweeperInterval` in the `<property>` tag of the HITACHI Connector Property file is less than the connection disconnecting time. Also, by setting a long `ConnectionTimeout`, you can extend the period for which a connection is stored.

3.15.8 Transaction timeout and statement cancellation

The transaction timeout functionality monitors the time elapsed since the transaction was started, and rolls back the transaction when the specified time has lapsed. The monitoring time is from the completion of transaction startup until the start of transaction conclusion. After the transaction times out, the access to the JTA and the JDBC interface in the applicable transaction reports an exception.

The transaction timeout functionality is enabled for the Application Server-managed transactions.

Note that after a transaction times out in the BMT, servlets, or JSPs, if you issue `UserTransaction.commit/rollback` for the applicable transaction, you can access the JTA and JDBC interface.

#: Whether a timeout has occurred is checked every second; therefore, a calculation error of a maximum of one second occurs for the time set up as a transaction timeout.

(1) Transaction timeout settings

You can specify the transaction timeout settings for the J2EE servers, `UserTransaction` interface, or `CMT` attribute of the `EJB`.

(a) Settings for the J2EE servers

You specify the transaction timeout settings for the J2EE servers by customizing the J2EE server properties. For details on customizing the settings for the J2EE server operations, see *3.15.13 Settings in the execution environment*.

(b) Settings for the UserTransaction interface

Use the `setTransactionTimeout` method of the `UserTransaction` interface to change the default values.

(c) Settings for the CMT attribute of the EJB

Use the server management commands to specify the settings for the CMT attribute of the EJB. You can choose to specify the settings for Beans, interfaces, and methods.

The settings are enabled for one of the following methods:

- Method in which the transaction attribute is set to `Required` and the EJB container starts the transaction
- Method in which the transaction attribute is set to `RequiresNew`

The timeout settings are disabled for the `Supports` attribute and `Mandatory` attribute operating in the transaction propagated from the client and for the `NotSupported` attribute and `Never` attribute that do not use transactions. For details on the transaction attributes, see *2.7.3 CMT in the uCosminexus Application Server EJB Container Functionality Guide*.

Note that the priority order of the settings is as follows:

1. Settings for methods
2. Settings for interfaces
3. Settings for Beans
4. Settings for J2EE servers

You specify the transaction timeout settings for Beans as the attributes (properties) of the Session Bean, Entity Bean, or Message-driven Bean included in the J2EE application. For details on the J2EE application settings, see *3.15.13 Settings in the execution environment*.

(2) Statement cancellation when a transaction timeout occurs

If a transaction times out when the SQL statement being executed has not returned, the statement is cancelled.

To enable statement cancellation, you must specify `true` in the `CancelStatement` property of DB Connector. For details on the resource adapter settings, see *3.15.13 Settings in the execution environment*.

3.15.9 Transaction recovery

This functionality concludes the 2-phase transactions that are in the prepared state or heuristic completion state due to the J2EE server and resource manager errors. This functionality is enabled when you use the global transactions.

When you start a J2EE server, the full transaction recovery processing is executed unconditionally for the imported resources. Also, partial recovery processing is executed even when the resource manager is down during the execution of a transaction.

Note that when you are using the in-process transaction service, you can display the transaction status with the server management command `cjlisttrn`. Also, you can use the `cjlisttrnfile` command to display the unconcluded transaction information remaining in the status file of a stopped J2EE server.

When the light transaction is enabled, the transaction recovery functionality is not available.

! Important note

If the transaction is not recovered due to the restarting of the J2EE server, the user must recover the transaction manually following the recovery procedure for each resource.

(1) Checking the unconcluded transactions and setting up a timeout when the J2EE server is terminated

When the J2EE server terminates normally, the J2EE server checks if there are any unconcluded transactions and then stops. If unconcluded transactions exist, the J2EE server waits infinitely until the transactions are completed. Also, the resources cannot be deleted until that transaction is concluded.

On the other hand, if a transaction need not be resolved quickly, such as during system development, you can set a timeout value for checking the unconcluded transactions. If a timeout occurs, the stop processing of the J2EE server is executed even if the checking of the unconcluded transactions is not complete. However, set a timeout value for operations such as J2EE application development. To guarantee transaction reliability during the operations of the J2EE applications, we recommend that you do not set a timeout value.

You specify the timeout settings by customizing the J2EE server properties. For details on customizing the settings for the J2EE server operations, see *3.15.13 Settings in the execution environment*.

(2) Notes

The notes on transaction recovery are as follows:

(a) Notes when the resource adapter fails to start when you start the J2EE server

When you start the J2EE server, if the resource adapter using the `XATransaction` fails to start, the J2EE server does not recover the transaction, outputs the message `KDJE48605-E`, and terminates forcefully. In this case, eliminate the reason due to which the resource adapter failed to start, and then restart the J2EE server. This concludes the prepared or heuristic transactions.

(b) Notes on restarting the J2EE server

- If you delete the resources after the J2EE server terminates forcefully or abnormally, the transaction cannot be recovered. Therefore, do not change the configuration of the resource when you restart the J2EE server.
- You must restart the J2EE server with the same port as the receiving port before forced or abnormal termination. Therefore, do not change the value of the parameter `ejbserver.distributedtx.recovery.port` in the `<configuration>` tag in the Easy Setup definition file. Note that if you set up the system without using Management Server, do not change the `ejbserver.distributedtx.recovery.port` key in `usrconf.properties`.

(c) Permissions for executing transaction recovery

- The default user set up in the resource (such as `XADatabaseSource`) executes the recovery. Specific permissions and settings are required to scan the unconcluded transactions depending on the resource manager. Also, when multiple users sign on, the default user must be assigned appropriate permissions in the resource manager that can conclude the transactions of the other users. For details, see the documentation for each resource.
- When a transaction is recovered in Oracle and Oracle JDBC Thin Driver is used as the JDBC driver, the user must have the following permissions:
 - `SELECT` permission for `SYS.DBA_PENDING_TRANSACTIONS`
 - `FORCE ANY TRANSACTION` permission
 - Permission to `EXECUTE SYS.DBMS_SYSTEM`

For details on the settings to use Oracle, see *4.1.7 Setting up the database connection environment (Oracle settings)* in the *uCosminexus Application Server System Setup and Operation Guide*.

(d) Number of connections used

Note the number of connections that are used when a transaction is recovered.

With J2EE server, you establish the following connections for one resource adapter that has the transaction support level `XATransaction`:

- Connections pooled in a connection pool
- Connection for recovery (one)

The maximum number of connections that the same resource manager requires is the value indicated by the following expression. Take precautions when there is an upper limit to the number of connections for the resource manager.

Maximum number of connections required for the same resource manager = $IR(1) + \dots + IR(N) + N$

$IR(i)$

Maximum number of connections in the pool settings for the i^{th} resource adapter.

$1 \leq i \leq N$.

N

- For the resource adapters conforming to the Connector 1.0 specifications, the number of resource adapters connected to the same resource manager.
- For the resource adapters conforming to the Connector 1.5 specifications, total number of connection definitions in the resource adapter connected to the same resource manager.

The target resource adapter is the one that is running and has the transaction support level `XATransaction`.

(3) Procedure for checking the transaction information

This section describes how to check the transaction information of a running and stopped J2EE server. You can check information such as the state of the transactions in a running J2EE server and the presence or absence of uncompleted transactions in a stopped J2EE server.

(a) Procedure for checking a running transaction

You can check the information for a running transaction on the J2EE server. You can check the information such as the transaction state, global transaction ID, time elapsed, and the branch type.

Use the `cjlisttrn` command to check a running transaction. The format and example of execution are as follows:

Format of execution

```
cjlisttrn [server-name] -bqual
```

Example of execution

```
cjlisttrn MyServer -bqual
```

You can also check the state of an uncompleted transaction. To check the information for an uncompleted transaction, specify `-pending` in the argument. The format and example of execution for checking the information on an uncompleted transaction are as follows:

Format of execution

```
cjlisttrn [server-name] -pending -bqual
```

Example of execution

```
cjlisttrn MyServer -pending -bqual
```

For details on the `cjlisttrn` command and the information that can be obtained, see *cjlisttrn (Displaying the transaction information of a running J2EE server)* in the *uCosminexus Application Server Command Reference Guide*.

(b) Procedure for checking a stopped transaction

You can check the transaction information of a stopped J2EE server. You can check the information such as the transaction state, global transaction ID, time elapsed, and the branch type. You can also check if any uncompleted transactions are remaining.

Use the `cjlisttrnfile` command to check a stopped transaction. The format and example of execution are as follows:

Format of execution

```
cjlisttrnfile [server-name] -bqual
```

Example of execution

```
cjlisttrnfile MyServer -bqual
```

If there are any uncompleted transactions when the J2EE server has stopped, conclude the transaction by executing the following processing as and when required:

- Restart the J2EE server (`cjstartsv` command)
- Start the J2EE server in the recovery mode (`cjstartrecover` command)

For details on the `cjlisttrnfile` command and the information that can be obtained, see *cjlisttrnfile (Displaying the transaction information of a stopped J2EE server)* in the *uCosminexus Application Server Command Reference Guide*.

3.15.10 Output of the SQL statement for troubleshooting

If a deadlock or slowdown error occurs, the issued SQL statement might be the cause of the error. Therefore, the output of the issued SQL statement to the log makes the error cause analysis easy. The SQL information output to the log is called the *SQL statement for troubleshooting*.

(1) Output timing

The SQL statement for troubleshooting is output at the following times:

- When the transaction times out
- When a J2EE application is terminated forcefully
- When the method cancellation command is executed
- When method cancellation is executed after a method timeout occurs

(2) Output destination

The SQL statement for troubleshooting is output to the resource adapter operation log and the trace based performance analysis.

With the resource adapter operation log, the SQL statement for troubleshooting is output to the KDJE50080-W message. For details, see *KDJE50080-W* in the *uCosminexus Application Server Messages*.

With the trace based performance analysis, the SQL statement for troubleshooting is output to the event ID 0x8C41. For details, see *8. Trace Collection Points and PRF Trace Collection Levels of the Trace Based Performance Analysis* in the *uCosminexus Application Server Maintenance and Migration Guide*.

(3) Output content

If the SQL statement is issued in the connection for output, the physical connection stores the SQL statement. This SQL statement stored by the physical connection is output as the SQL statement for troubleshooting.

Physical connection that stores the SQL statement

The physical connections storing the SQL statements, for the various timings at which the SQL statement for troubleshooting is output, are as follows:

- **When a transaction times out**
The physical connection supporting the connection that participates in the transaction.
- **When a J2EE application is terminated forcefully, when the method cancellation command is executed, or when method cancellation is executed after a method timeout occurs**

The physical connection supporting the connection that participates in the transaction when the transaction is being processed.

If no transactions are being used, then the physical connection supporting the connection being used by the instance that forcefully stops the application or executes method cancellation stores the SQL statement for troubleshooting. Note that the SQL statement for troubleshooting is not output to a closed connection.

APIs storing the SQL statement

When the following APIs are invoked by the user application, the SQL statement passed in the argument is stored in the physical connection. One SQL statement is stored for each physical connection. Whenever an API is invoked, the latest SQL statement is overwritten. The following table describes the APIs storing the SQL statement.

Table 3-60: List of APIs storing the SQL statement

Interface	Method
java.sql.Connection	prepareCall(String sql)
	prepareCall(String sql, int resultSetType, int resultSetConcurrency)
	prepareCall(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)
	prepareStatement(String sql)
	prepareStatement(String sql, int autoGeneratedKeys)
	prepareStatement(String sql, int[] columnIndexes)
	prepareStatement(String sql, int resultSetType, int resultSetConcurrency)
	prepareStatement(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability)
java.sql.Statement	addBatch(String sql)
	execute(String sql)
	execute(String sql, int autoGeneratedKeys)
	execute(String sql, int[] columnIndexes)
	execute(String sql, String[] columnNames)
	executeBatch() #1
	executeQuery(String sql)
	executeUpdate(String sql)
	executeUpdate(String sql, int autoGeneratedKeys)
	executeUpdate(String sql, int[] columnIndexes)
	executeUpdate(String sql, String[] columnNames)
java.sql.PreparedStatement	Display target method inherited from java.sql.Statement
	addBatch() #2
	execute() #2
	executeQuery() #2
	executeUpdate() #2
java.sql.CallableStatement	Display target method inherited from java.sql.PreparedStatement

#1: When you execute the `executeBatch()` method, the SQL statement finally added with the `addBatch(String sql)` method and `addBatch()` method is stored in the physical connection.

#2: When you execute the `addBatch()` method, `execute()` method, `executeQuery()` method, and `executeUpdate()` method, the SQL statement given by the arguments of the `prepareStatement` method and

`prepareCall` method of `java.sql.Connection` is stored in the physical connection. However, the `IN` parameter placeholder ("?") of the SQL statement is output as "?" without being substituted.

(4) Notes

The notes on the functionality to output the SQL statement for troubleshooting are as follows:

- This functionality is always enabled when you use DB Connector. The functionality is disabled when a resource adapter other than DB Connector is used.
- If the messages output to the SQL log for troubleshooting, in the resource adapter operation log, exceed 4 KB, the message is only output up to 4 KB.
- For the connections shared by connection sharing and connection association, only one SQL log for troubleshooting is output because there is one physical connection.

3.15.11 Automatically closing the objects

The objects such as the `Statement` objects that are opened with the user program must be closed with the user program. However, if the objects cannot be closed, DB Connector automatically closes the following objects that are generated from the user handle (`java.sql.Connection`):

- `Statement` object
- `PreparedStatement` object
- `CallableStatement` object
- `ResultSet` objects generated from various statements and `DatabaseMetaData`

3.15.12 Definitions in `cosminexus.xml`

From among the functionality for fault tolerance, you specify the CMT transaction timeout definition in the `<ejb-jar>` tag of `cosminexus.xml`. The following table describes the definition of the functionality for fault tolerance in `cosminexus.xml`.

Table 3-61: Definition of the functionality for fault tolerance in `cosminexus.xml`

Specified tag	Settings
<code><message>-<ejb-transaction-timeout></code> tag	Specifies the time at which the CMT transaction will time out.
<code><entity>-<ejb-transaction-timeout></code> tag	
<code><session>-<ejb-transaction-timeout></code> tag	

For details on `cosminexus.xml`, see 2. *Application Property File (cosminexus.xml)* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

3.15.13 Settings in the execution environment

This subsection describes the settings for using the functionality for fault tolerance.

Note that the settings for the following functionality need not be specified beforehand.

- Displaying the connection pool information
- Clearing the connection pool
- Output of the SQL statement for troubleshooting
- Automatically closing the objects

(1) J2EE server settings

You implement the J2EE server settings in the Easy Setup definition file. Specify the fault tolerance definition in the `<configuration>` tag of the logical J2EE server (`j2ee-server`) in the Easy Setup definition file.

The following table describes the definition of the functionality for fault tolerance in the Easy Setup definition file.

Table 3-62: Definition of the functionality for fault tolerance in the Easy Setup definition file

Category	Property	Settings
Automatically closing the connections	<code>ejbserver.webj2ee.connectionAutoClose.enabled</code>	Specifies whether the connection will be automatically closed by the Web application.
Transaction timeout (for J2EE servers) ^{#1}	<code>ejbserver.jta.TransactionManager.defaultTimeout</code>	Specifies the default timeout value for the transactions running on the J2EE server.
Transaction recovery	<code>ejbserver.distributedtx.recovery.port</code>	Specifies the fixed port number used for transaction recovery when a global transaction is used.
Timeout during the checking of uncompleted transactions ^{#2}	<code>ejbserver.distributedtx.recovery.completionCheckOnStopping.timeout</code>	Specifies the timeout value for checking if an in-progress transaction is complete, when the J2EE server is stopped.
Timeout in error detection	<code>ejbserver.connectionpool.validation.timeout</code>	Specifies the timeout value for the connection error detection functionality and the timeout value for the connection deletion processing by the connection count adjustment functionality.

#1: In the CMT, you can also specify the settings for the Enterprise Beans, interfaces, and methods. To specify the settings for Enterprise Beans, interfaces, and methods, use the server management commands to specify the settings in the property files when the J2EE applications are set up. For details on the J2EE application settings, see *3.15.12 Definitions in cosminexus.xml*.

#2: Set up the timeout during application development. To guarantee transaction reliability during the operations of the J2EE applications, we recommend that you do not set a timeout value.

For details on the Easy Setup Definition file and parameters, see *4.6 Easy Setup Definition file* in the *uCosminexus Application Server Definition Reference Guide*.

(2) Resource adapter settings

Use the server management commands and property files to specify the resource adapter settings in the execution environment. Use the HITACHI Connector Property file to define the functionality for fault tolerance.

The following table describes the definition of the functionality for fault tolerance in the HITACHI Connector Property file.

Table 3-63: Definition of the functionality for fault tolerance in the HITACHI Connector Property file

Items	Specified parameters	Settings
Detecting the connection errors	<code>ValidationType</code> and <code>ValidationInterval</code> in the <code><property></code> tag	Specifies the time at which the connection errors will be detected and the intervals at which the errors will be detected. Note that to set up a timeout for detecting the connection errors, you enable the usage of the connection management threads with <code>NetworkFailureTimeout</code> . ^{#1}
Connection management threads	<code>NetworkFailureTimeout</code> in the <code><property></code> tag	Specifies whether the connection management threads will be used. When you use the connection management threads, a timeout is set for the connection error detection functionality and the connection count adjustment functionality. The timeout value is fixed at 5 seconds with version 07-60 or earlier. With version 08-00

Items	Specified parameters	Settings
Connection management threads	NetworkFailureTimeout in the <property> tag	or later, you can specify any timeout value (default value is 5 seconds).
Waiting for a connection when connections deplete	RequestQueueEnable in the <property> tag	Specifies whether to enable the waiting for a connection when connections deplete.
	RequestQueueTimeout in the <property> tag	Specifies the waiting time for obtaining a connection.
Retrying to obtain a connection	RetryCount in the <property> tag	Specifies the retry count for a failure to obtain a connection.
	RetryInterval in the <property> tag	Specifies the retry interval for a failure to obtain a connection.
Connection sweeper	SweeperInterval in the <property> tag ^{#2}	Specifies the interval at which the automatic destruction of connections (connection sweeper) will operate.
	ConnectionTimeout in the <property> tag ^{#2}	Specifies the time from the last use of the connection until the determination of the automatic destruction of the connection.
Statement cancellation	CancelStatement in the <property> tag	Specifies whether to enable statement cancellation when a transaction times out.

#1: You set up the timeout for the connection error detection functionality and the connection count adjustment functionality in the same key. Therefore, when you use a timeout for the connection error detection functionality, a timeout is also used for the connection count adjustment functionality.

#2 If the set value is less than 3600 seconds, KDJE48604-W is output.

For details on the HITACHI Connector Property file, see *4.1 HITACHI Connector Property file* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

(3) J2EE application settings

You implement the J2EE application settings in the execution environment with the server management commands and property files. To define the functionality for fault tolerance, use the Session Bean property file, Entity Bean property file, or Message-driven Bean property file.

The tags specified in these property files are mapped in `cosminexus.xml`. For details on the definitions in `cosminexus.xml`, see *3.15.12 Definitions in cosminexus.xml*.

3.16 Other resource adapter functionality (For the resource adapters conforming to the Connector 1.5 specifications)

This section describes the resource adapter functionality other than the functionality described in *3.14 Functionality for performance tuning*, *3.15 Functionality for fault tolerance*, and *3.17 Functionality for connection pool clustering*.

The following table describes the organization of this section.

Table 3-64: Organization of this section (Other resource adapter functionality (For the resource adapters conforming to the Connector 1.5 specifications))

Category	Title	Reference location
Explanation	Managing the resource adapter lifecycle	<i>3.16.1</i>
	Managing the resource adapter work	<i>3.16.2</i>
	Message inflow	<i>3.16.3</i>
	Transaction inflow	<i>3.16.4</i>
	Looking up Administered objects	<i>3.16.5</i>
	Specifying multiple connection definitions	<i>3.16.6</i>
Implementation	Application Server-specific Connector 1.5 API specifications	<i>3.16.7</i>
Settings	Settings for using the resource adapters conforming to the Connector 1.5 specifications	<i>3.16.8</i>
	Examples of property file specification	<i>3.16.9</i>
Notes	Notes on using the resource adapters conforming to the Connector 1.5 specifications	<i>3.16.10</i>

Note: The functionality-specific explanation is not available for "Operations".

The above functionality is available with the resource adapters conforming to the Connector 1.5 specifications. For details on the resource adapters conforming to the Connector 1.5 specifications available with Application Server, see *3.3.2(2) Resource adapters conforming to the Connector 1.5 specifications*.

Reference note

When you use resource adapters other than those described in *3.3.2(1) Resource adapters conforming to the Connector 1.0 specifications*, such as the resource adapters conforming to the Connector 1.5 specifications, we recommend that the trace be output at the entry and exit of the resource adapter processing. The output trace will be used to isolate the causes when errors occur.

3.16.1 Managing the resource adapter lifecycle

When you use the resource adapters conforming to the Connector 1.5 specifications, you can use the J2EE server to manage the lifecycle of the resource adapters.

Resource adapter lifecycle management is functionality that manages the start and stop processing of the resource adapters with the J2EE servers.

(1) Preconditions

Resource adapter lifecycle management is enabled when the resource adapter satisfies the following conditions:

- The classes described in *(2) Classes used for lifecycle management* are implemented in the resource adapter.
- The JavaBeans class name implementing the `javax.resource.spi.ResourceAdapter` interface is specified in `<connector>-<resourceadapter>-<resourceadapter-class>` of the DD (`ra.xml`).

Note that if `<connector>-<resourceadapter>-<resourceadapter-class>` is not specified in the DD, the resource adapter lifecycle is not managed.

(2) Classes used for lifecycle management

This section describes the classes used for resource adapter lifecycle management. The classes to be used include the classes that must be implemented in the resource adapter and the classes provided by the J2EE servers.

■ Classes that must be implemented in the resource adapter

The following classes must be implemented with the resource adapter. Note that the Connector 1.5 specifications define that these classes be implemented as JavaBeans.

- **Implementation class of the `javax.resource.spi.ResourceAdapter` interface**
- **Implementation class of the `javax.resource.spi.ManagedConnectionFactory` interface**
- **Implementation class of `AdminObject` (Administered object)**

For the detailed implementation, see the Connector 1.5 specifications.

■ Classes provided by the J2EE servers

The J2EE server provides the following classes:

- **Implementation class of the `javax.resource.spi.work.WorkManager` interface**
The methods such as the `doWork(Work)` method, `scheduleWork(Work)` method, and `startWork(Work)` method are implemented in this class.
- **Implementation class of the `javax.resource.spi.BootstrapContext` interface**
The `createTimer()` method, `getWorkManager()` method, and `getXATerminator()` method are implemented in this class.
Note that if you invoke the `getXATerminator` method of the `BootstrapContext` interface with Application Server, null is returned.

(3) Controlling lifecycle management

This section describes the controlling of the start and stop processing of the resource adapters whose lifecycles are to be managed.

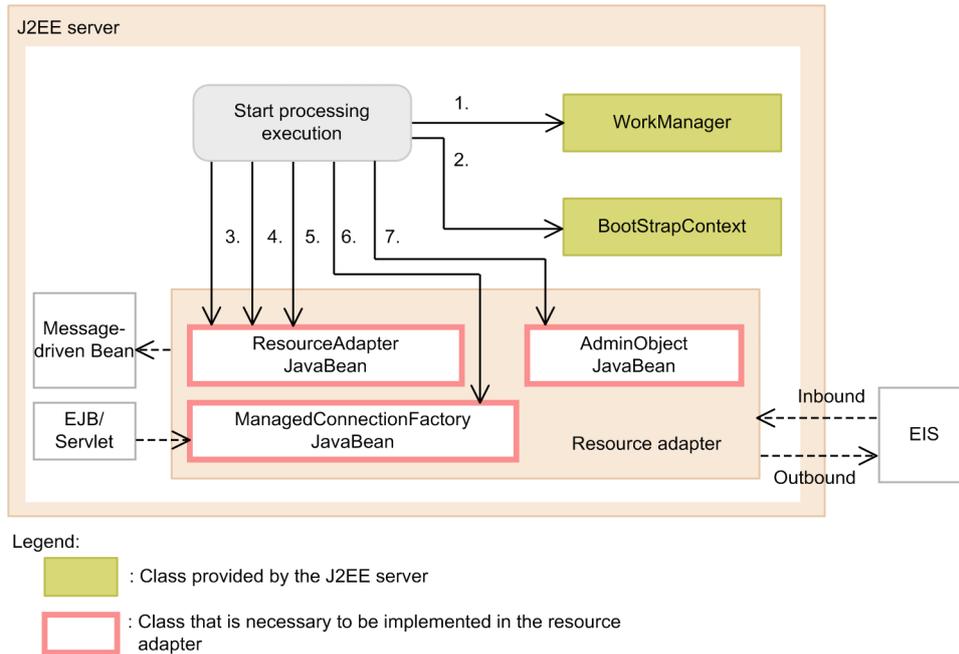
The start and stop processing of a resource adapter is executed at the following times:

- When you start or stop a J2EE resource adapter (when you execute the `cjstartrar` command or `cjstoprar` command)
- When you start or stop a J2EE server (when you execute the `cjstartsv` command or `cjstopsv` command)
- When you start or stop an application containing the resource adapter (when you execute the `cjstartapp` command or `cjstopapp` command)

■ Control procedure when the resource adapter start processing is executed

The following figure shows the control procedure when the resource adapter start processing is executed.

Figure 3-38: Control procedure when the resource adapter start processing is executed



The following points describe the control executed by the resource adapter start processing. Note that the item numbers correspond to the numbers in the figure.

1. `WorkManager` (implementation class of the `javax.resource.spi.work.WorkManager` interface) is generated.
2. `BootstrapContext` (implementation class of the `javax.resource.spi.BootstrapContext` interface) is generated.
3. `ResourceAdapterJavaBean` (implementation class of the `javax.resource.spi.ResourceAdapter` interface) is generated.

The implementation class generated as `ResourceAdapterJavaBean` is the class specified in `<connector>-<resourceadapter>-<resourceadapter-class>` in the DD (`ra.xml`) of the resource adapter.

If the class specified in this tag cannot be instantiated, the resource adapter fails to start[#]. In this case, the `KDJE48589-E` message is output.

4. The properties are set up in `ResourceAdapterJavaBean`.
The value specified in `<connector>-<resourceadapter>-<config-property>` in the DD (`ra.xml`) is set up in `ResourceAdapterJavaBean` that was generated in 3. The settings are executed with the setter method complying with the JavaBean specifications. If an exception occurs during the invocation of the setter method, the `KDJE48594-W` message is output. However, the processing continues.
5. The start method of the `javax.resource.spi.ResourceAdapter` interface is invoked and the resource adapter is started.

If the invocation of this method throws an exception, the resource adapter fails to start[#]. In this case, the `KDJE48590-E` message is output.

6. `ResourceAdapterJavaBean` and `ManagedConnectionFactoryJavaBean` are associated (for Outbound).

When you use the lifecycle management functionality, the implementation class of the `javax.resource.spi.ManagedConnectionFactory` interface implements the `javax.resource.spi.ResourceAdapterAssociation` interface. `ResourceAdapterJavaBean` and `ManagedConnectionFactory` are associated using the `setResourceAdapter(ResourceAdapter)` method of the `javax.resource.spi.ResourceAdapterAssociation` interface.

Also, in the following cases, the resource adapter fails to start[#]. In this case, `KDJE48591-E` is output.

- When the `javax.resource.spi.ResourceAdapterAssociation` interface is not implemented for `ManagedConnectionFactoryJavaBean`
- When an exception is thrown for the invocation of the `setResourceAdapter` method

7. `AdminObjectJavaBean` (Administered object) is generated and the properties are set up.

The implementation class generated as `AdminObjectJavaBean` is the class specified in `<connector>-<resourceadapter>-<adminobject>-<adminobject-class>` in the DD (`ra.xml`) of the resource adapter. If the class specified in this tag cannot be instantiated, the resource adapter fails to start[#]. In this case, the `KDJE48597-E` message is output.

Also, the value specified in `<connector>-<resourceadapter>-<adminobject>-<config-property>` in the DD (`ra.xml`) is set up in `AdminObjectJavaBean`. The settings are executed with the setter method complying with the JavaBean specifications. If an exception occurs during the invocation of the setter method of `AdminObjectJavaBean`, the `KDJE48598-W` message is output. However, the processing continues.

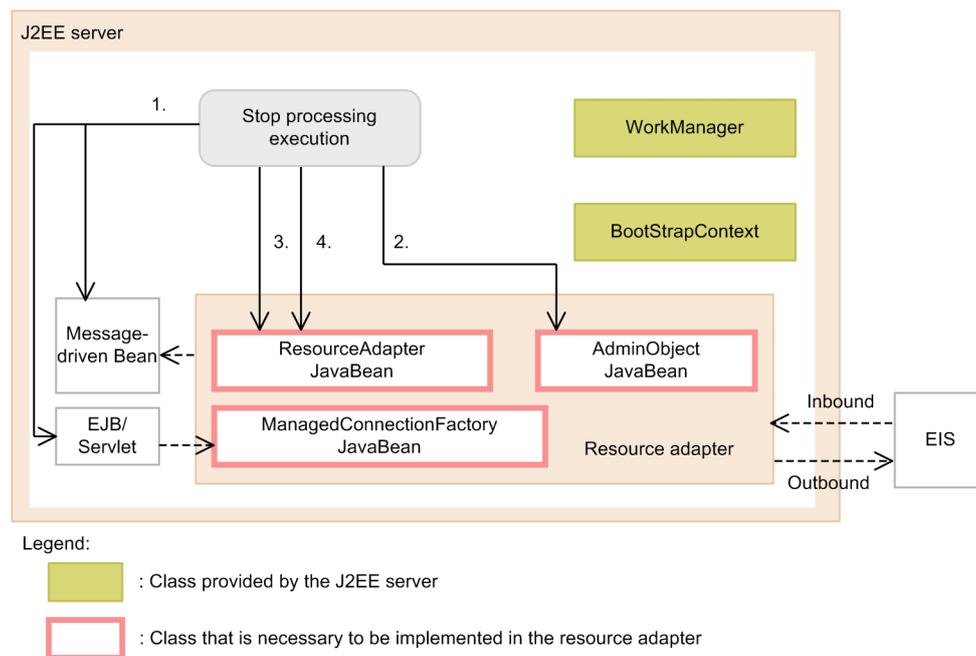
#

For the resource adapters included in an application, when the resource adapter fails to start, the start processing of the application containing that resource adapter also fails.

■ Control procedure when the resource adapter stop processing is executed

The following figure shows the control procedure when the resource adapter stop processing is executed.

Figure 3-39: Control procedure when the resource adapter stop processing is executed



The following points describe the control executed by the resource adapter stop processing. Note that the item numbers correspond to the numbers in the figure.

1. A check is executed to confirm that all the EJBs, servlets, and Message-driven Beans, referencing the resource adapter to be stopped, are stopped.
Make sure that the following elements are not being used:
 - Inbound resource adapter
 - Connection factory
 - Administered object
 If these elements are being used, the resource adapter stop processing is cancelled.
2. `AdminObjectJavaBean` (Administered object) is destroyed.

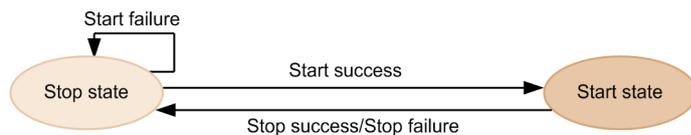
3. The `stop` method of the `javax.resource.spi.ResourceAdapter` interface is invoked and the resource adapter stops.
If the invocation of this method throws an exception, the KDJE48590-E message is output and the resource adapter stop processing fails.
4. `ResourceAdapterJavaBean` is destroyed.

■ Transition of the resource adapter state

The state of the resource adapter changes to `Started` or `Stopped` when the resource adapter start or stop processing is executed.

The following figure shows the transition of the resource adapter state.

Figure 3-40: Transition of the resource adapter state



(4) Notes on using the lifecycle management functionality

Note the following points when you use the lifecycle management functionality:

- If multiple resource adapters are deployed to a J2EE server, the order in which the start and stop processing is executed is not definite. Also, if multiple resource adapters are included in an application, the order in which the start and stop processing is executed is not definite. If a processing depends on the execution order, the operations are not guaranteed.
- For the resource adapters included in an application, the resource adapter start processing is executed before the start processing of the EJBs and Servlets included in the application. Also, the resource adapter stop processing is executed after the stop processing of the EJBs and Servlets included in the application.
- Execute the following processing appropriately with the processing of `ResourceAdapter.stop`:
 - Destroying `Timer` generated with `BootstrapContext.createTimer(Timer#cancel)`
 - Termination instruction to the `Work` object registered in `WorkManager(Work#release)`

3.16.2 Managing the resource adapter work

When you use the resource adapters conforming to the Connector 1.5 specifications, you can manage the threads used by the resource adapters through the J2EE server.

Resource adapter work management is a functionality for using the threads appropriately when a resource adapter is operated by multithreading. The J2EE server manages the threads in a pool, and allocates the threads to the necessary resource adapters.

! Important note

The J2EE application execution time is not monitored for the thread that executes `Work` for which work is to be managed.
For details on monitoring the J2EE application execution time, see *5.3 Monitoring and cancelling the J2EE application execution time* in the *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*.

(1) Preconditions

The resource adapter work management functionality is available when the lifecycle of the resource adapter is being managed. For details on the resource adapter lifecycle management, see *3.16.1 Managing the resource adapter lifecycle*.

(2) Classes used for work management

This section describes the classes used for managing the resource adapter work. The classes to be used include the classes that must be implemented in the resource adapter and the classes provided by the J2EE servers. Note that the classes provided by the J2EE servers are the same as the classes used for the lifecycle management. See 3.16.1(2) *Classes used for lifecycle management*.

■ Classes that must be implemented in the resource adapter

The following class must be implemented with the resource adapter:

- **Implementation class of the `javax.resource.spi.work.Work` interface**

You must implement the processing you want to execute in the `run` method of this class.

Note that if you want to use a resource adapter to manage the timing for work registration, allocation of work to threads, and work termination, execute the implementation class of the `javax.resource.spi.work.WorkListener` interface as well.

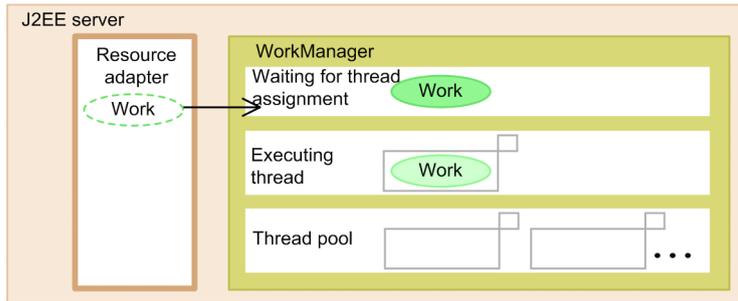
(3) Procedure for work management

This section describes the procedure for work management.

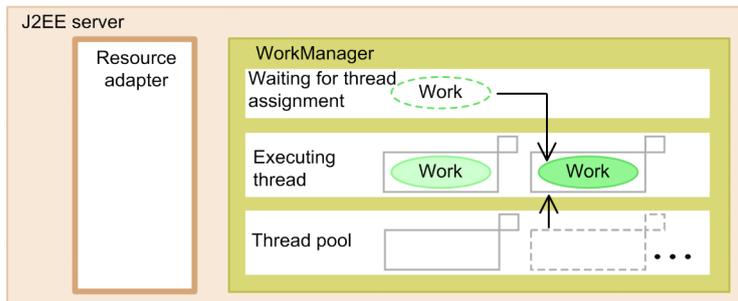
The following figure shows gives an overview of work management.

Figure 3-41: Overview of work management

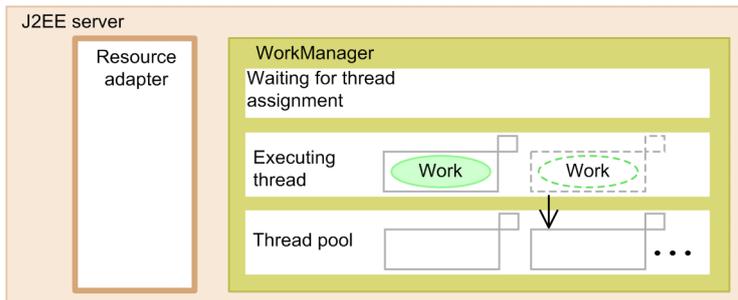
1. Work registration



2. Assignment to Work of an empty thread



3. End of Work and recovery of thread



The following points describe the procedure shown in the figure:

1. Registering `Work`

The resource adapter generates `Work` (implementation class of the `javax.resource.spi.work.Work` interface) and registers `Work` in `WorkManager`.

At this time, the `WorkManager` instance is passed to the resource adapter via `BootstrapContext`. For details on `WorkManager`, see 3.16.1 *Managing the resource adapter lifecycle*.

If you also register `WorkListener` (`javax.resource.spi.work.WorkListener`) in `WorkManager` at the same time as registering `Work`, thereafter, when the registration of `Work` is complete (when 1 is executed), when a thread is allocated to `Work` (when 2 is executed), and when the processing of `Work` is complete (when 3 is executed), you can obtain each event (implementation class of the `javax.resource.spi.work.WorkEvent` interface).

2. Allocating work to a free thread

The J2EE server allocates a free thread to `Work` registered in `WorkManager`, and executes the `run` method implemented in `Work`.

3. Terminating `Work` and calling back the threads

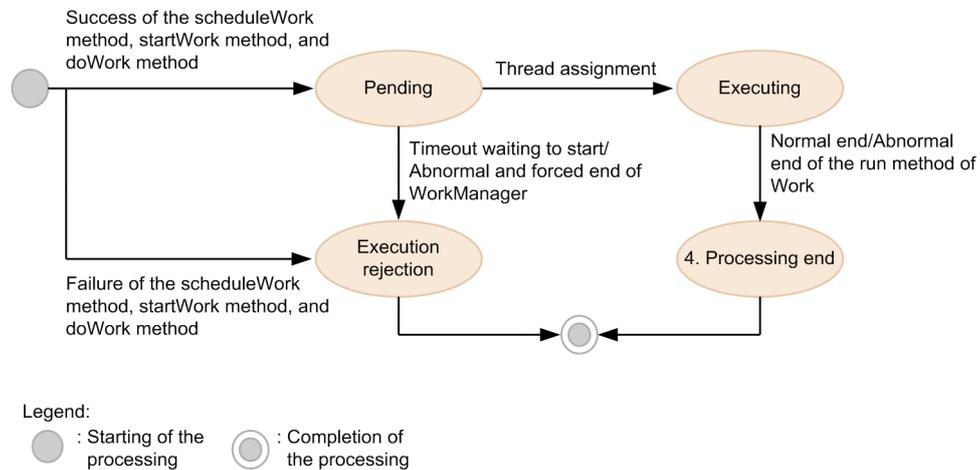
The J2EE server calls back the thread, which was allocated to the processed `Work`, to the thread pool.

The called back thread returns to the pool and waits or is released according to the settings for thread pooling.

Note that if an exception occurs in the methods of the `javax.resource.spi.work.Work` interface or `javax.resource.spi.work.WorkListener` interface during this procedure, the KDJE48592-E or KDJE48593-E message is output respectively.

The following figure shows the transition in the state of `Work` registered in `WorkManager`.

Figure 3-42: Transition of Work state



Note that the time at which a method is returned varies according to the method used for registering `Work` in `WorkManager`. The following table and figure show the times at which each method is returned. Note that apart from the time at which the method is returned, there is no difference in the processing of these methods.

Table 3-65: Times at which the methods are returned

Method	Return timing
<code>scheduleWork</code>	Registers <code>Work</code> and returns immediately.
<code>startWork</code>	Returns when a thread is allocated to <code>Work</code> .
<code>doWork</code>	Returns when the processing of <code>Work</code> is completed.

Figure 3-43: Time at which the scheduleWork method is returned

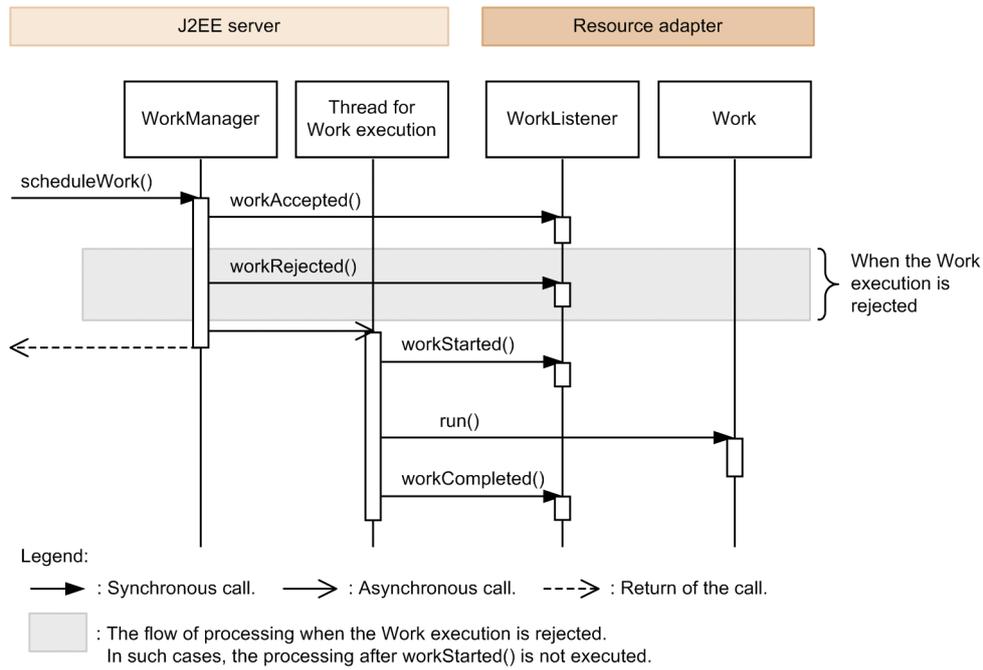


Figure 3-44: Time at which the startWork method is returned

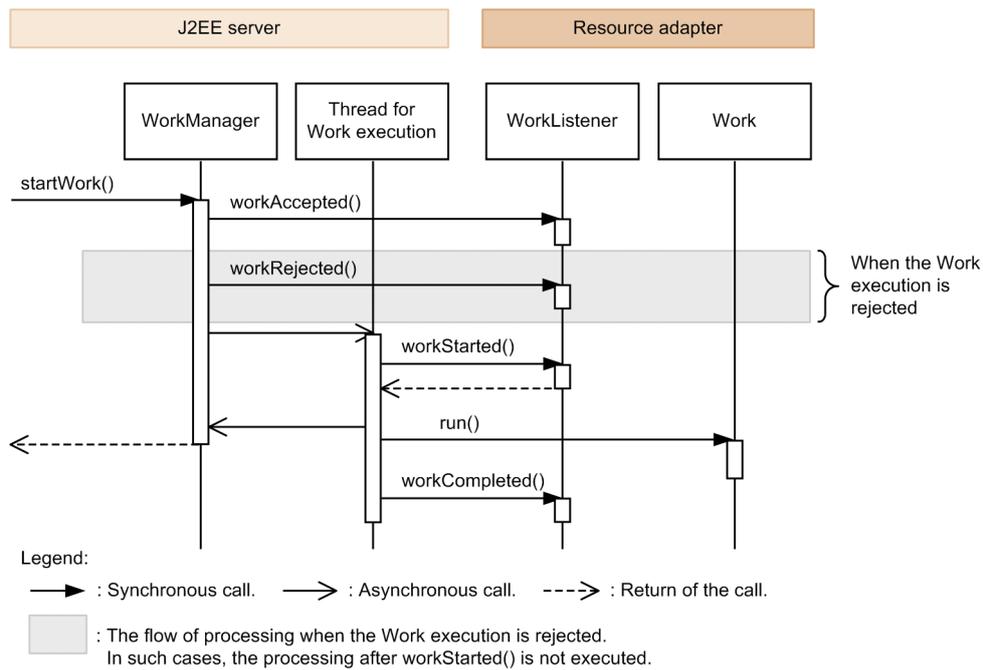
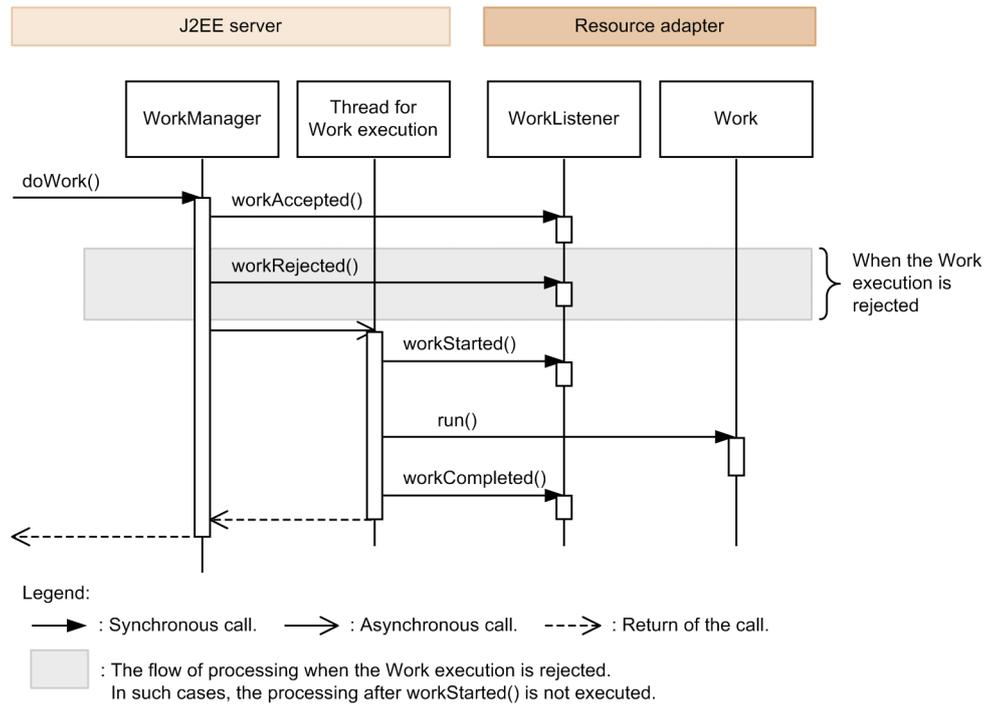


Figure 3-45: Time at which the doWork method is returned



(4) Thread pooling

The thread pooling functionality manages the threads allocated to work in a *thread pool*.

■ Settings for thread pooling

After you import the resource adapter into the J2EE server, you specify the settings for thread pooling in the `<hitachi-connector-property>-<resourceadapter-runtime>-<property>` tag of the HITACHI Connector Property file and set up as the property of each resource adapter. For details on the procedure for specifying the settings, see 5.4 *Resource adapter property definition* in the *uCosminexus Application Server Application Setup Guide*.

The following table describes the values that can be set in the thread pool.

Table 3-66: Values that can be set in the thread pool

Value that can be set (Property name)	Explanation
Maximum number of threads executed concurrently (MaxThreadPoolSize)	Specifies the maximum number of threads executed concurrently in WorkManager. If there are no free threads when Work is registered, new threads are generated and allocated to Work if the number of threads running in WorkManager is less than this value. If the number of running threads is equal to this value or more, Work is not accepted and <code>javax.resource.spi.work.WorkRejectedException</code> is thrown.
Minimum number of threads in the thread pool (MinThreadPoolSize)	Specifies the minimum number of threads pooled in the thread pool. Even if no Work is registered in WorkManager, threads equal to this value are always pooled. Also, if 0 is specified in this value, threads are not generated until Work is registered.
Maximum survival period until the thread to which no Work is allocated is released [unit: seconds] (TPoolKeepalive)	Specifies the period until the threads to which no Work is allocated are released from the thread pool, in seconds.

Value that can be set (Property name)	Explanation
Maximum survival period until the thread to which no work is allocated is released [unit: seconds] (TPoolKeepalive)	If no work is allocated even after the lapse of TPoolKeepalive seconds since the thread became free, that thread is released. However, the thread is not released if the number of threads existing in the thread pool is only up to the MinTPoolSize value.

For details on the specification method and the specifiable values, see 4.1 HITACHI Connector Property file in the uCosminexus Application Server Application and Resource Definition Reference Guide.

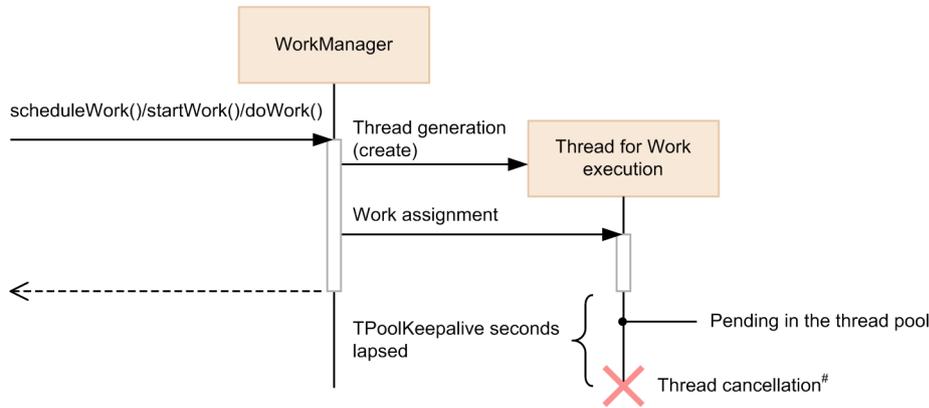
Note that if the lifecycle management functionality is not enabled (when <resourceadapter-class> is not specified in the DD (ra.xml) of the resource adapter), the value of the property that specifies thread pooling is ignored.

■ Lifecycle of the threads used for work management

The following figure shows the lifecycle of the threads used by the work management functionality.

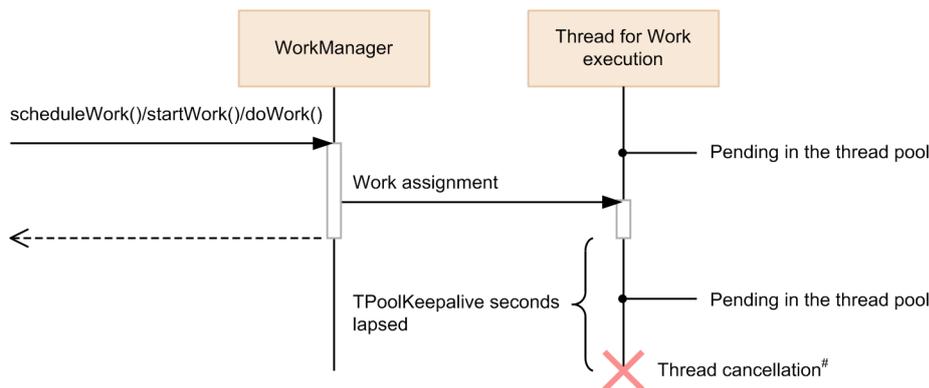
The lifecycle of a thread varies according to the state of the thread pool and the running threads.

Figure 3-46: Lifecycle of a thread (when there are no free threads in the thread pool and the number of running threads is less than MaxTPoolSize)



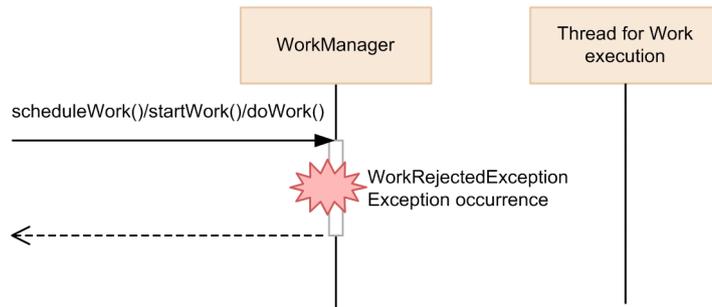
#: When the number of threads of the thread pool are MinTPoolSize, they are not canceled.

Figure 3-47: Lifecycle of a thread (when there are free threads in the thread pool)



#: When the number of threads of the thread pool are MinTPoolSize, they are not canceled.

Figure 3-48: Lifecycle of a thread (when there are no free threads in the thread pool and the number of running threads has reached MaxThreadPoolSize)



A thread is generated when the resource adapter invokes the `scheduleWork` method, `startWork` method, or `doWork` method and both the following states are applicable:

- There are no free threads in the thread pool.
- The number of running threads in `WorkManager` is less than `MaxThreadPoolSize`.

When the processing of `Work` is complete, the thread is in the waiting state in the thread pool. When the next `Work` is registered, the thread is again in the execution state.

In the thread pool, a thread is released if `Work` is not allocated even after the lapse of the seconds specified in `TPoolKeepalive`. However, by releasing that thread, if the number of threads in the thread pool becomes less than `MinThreadPoolSize`, the thread is not released.

■ Relationship between the Message-driven Bean instance pool and thread pooling

The operation when the resource adapter uses `Work` to invoke the Message-driven Beans varies according to the relationship between the maximum number of instances in the Message-driven Bean instance pool and `MaxThreadPoolSize` of the resource adapter.

The following table describes the operations for each relationship between the maximum number of instances in the Message-driven Bean instance pool and `MaxThreadPoolSize` of the resource adapter. This operation is executed when one Message-driven Bean is invoked from one resource adapter.

Table 3-67: Operations when the resource adapter invokes the Message-driven Beans using `Work`

Relationship between the maximum number of instances in the Message-driven Bean instance pool and <code>MaxThreadPoolSize</code> of the resource adapter	Operation
<code>MaxThreadPoolSize</code> > maximum number of instances in the Message-driven Bean instance pool	The Message-driven Beans can be executed from <code>Work</code> until the maximum number of instances in the Message-driven Bean instance pool is reached. The <code>Work</code> count exceeding the maximum number of instances in the Message-driven Bean instance pool enters a state of waiting to obtain a Message-driven Bean instance.
<code>MaxThreadPoolSize</code> = maximum number of instances in the Message-driven Bean instance pool	The Message-driven Beans can be executed from <code>Work</code> until the maximum number of instances in the Message-driven Bean instance pool is reached. The <code>javax.resource.spi.work.WorkRejectedException</code> exception is thrown for the <code>Work</code> count exceeding the maximum number of instances in the Message-driven Bean instance pool. The state does not change to waiting to obtain instances.
<code>MaxThreadPoolSize</code> < maximum number of instances in the Message-driven Bean instance pool	The number of concurrently available Message-driven Bean instances becomes the number of <code>MaxThreadPoolSize</code> . The Message-driven Beans can be executed from <code>Work</code> until the running <code>Work</code> count reaches <code>MaxThreadPoolSize</code> . The <code>javax.resource.spi.work.WorkRejectedException</code> exception is thrown for the <code>Work</code> count exceeding <code>MaxThreadPoolSize</code> . The state does not change to pending.

Tip

You specify the thread pooling of the work management functionality for the resource adapters. On the other hand, you set up the Message-driven Bean instance pool for the Message-driven Beans.

When the resource adapter invokes multiple Message-driven Beans, in the `MaxThreadPoolSize` of the resource adapter, you set up the total value considering the number of threads required for each Message-driven Bean.

(5) Work management start and termination processing

The work management start and termination processing is executed when the start and termination processing of the lifecycle management functionality is executed.

The operations executed in each processing are as follows:

■ Start processing

With the work management start processing, threads are generated according to the settings for thread pooling. If `MinThreadPoolSize` is 1 or more, the threads are generated for the value in `MinThreadPoolSize` and are stored in the thread pool as free threads.

■ Termination processing

Some of the content executed in the termination processing differs for a normal termination and for an abnormal or forced termination. Note that abnormal termination occurs when an exception is thrown during the execution of the `stop` method of the `ResourceAdapter` interface. Also, forced termination is the termination processing executed by extending the execution of the `cjstopapp -force` command or `cjstopapp -cancel` command.

Reference note

When you execute the `cjstopsv -f` command, the J2EE server is stopped without executing the resource adapter termination processing. Therefore, the processing described here is not executed.

1. The `stop` method of the `ResourceAdapter` interface is invoked for the resource adapter from the J2EE server, and the stop processing is instructed.
Also, the `release` method of `Work` is invoked by extending this processing, and `Work` is released.
2. The resource adapter stops accepting new `Work`.
This is after the processing in 1 is executed, so normally there is no `Work` registration request. If there is a `Work` registration request, `javax.resource.spi.work.WorkRejectedException` is thrown as an exception.
3. The system waits until the processing of the `Work` is completed.
The system waits until the entire execution of the `run` method of the running `Work` is completed.

Tip

- If method cancellation is executed for a Message-driven Bean invoked from `Work`, the Message-driven Bean is terminated forcefully. However, in that case, the processing of the running `Work` does not stop.
 - If a Message-driven Bean is invoked from `Work`, the Message-driven Bean is terminated before the resource adapter. If you invoke a Message-driven Bean from `Work` after the Message-driven Bean is terminated, and if you invoke the `createEndpoint` method of `javax.resource.spi.endpoint.MessageEndpointFactory`, an exception is thrown.
For details, see *3.16.3 Message inflow*.
-

(6) Notes on using the work management functionality

- With the `Work` and `WorkListener` methods, only the invocation of Message-driven Beans based on the message inflow contract is available as the J2EE server functionality.
- Make the `Work` and `WorkListener` methods thread-safe.
- Do not implement processing dependent on the executing thread in the `WorkListener` method. The operations might not function properly if the executing thread-dependent processing is included.

3.16.3 Message inflow

A *message inflow* is a contract between the resource adapters and Message-driven Beans. The resource adapters complying with the Connector 1.5 specifications can receive messages from the message providers such as EIS, and operate the message endpoint (Message-driven Bean) on Application Server. The message endpoint asynchronously processes the messages sent from the message provider.

You can use the following two message delivery methods with Application Server:

- Non-Transacted Delivery
- Transacted Delivery

The difference in the methods is the participation of the invocation source EIS in the transaction.

(1) Preconditions

The message inflow is enabled when the resource adapters and Message-driven Beans satisfy the following conditions:

- The Message-driven Beans invoked in the message inflow must conform to EJB 2.1 or later.
With EJB 2.1 or later, the Message-driven Bean can implement any message listener and not only `javax.jms.MessageListener`. A message listener is a listener used to deliver messages between the resource adapters and Message-driven Beans. By implementing a resource adapter-supported message listener in the Message-driven Bean, generic messages can be received.
If you try to execute message inflow in EJB 2.0 Message-driven Bean, an error occurs when the application starts, and the application fails to start. In this case, the KDJE42088-E message is output.
- To execute message inflow, you must specify the following settings as the attributes of the resource adapters and Message-driven Beans:
 - Information to be set up in the Administered object (resource adapters)
 - Mapping of the Message-driven Beans and resource adapters (Message-driven Beans and resource adapters)
 - Information to be set up in `ActivationSpec` (Message-driven Beans)
 - Interfaces used by the Message-driven Beans (Message-driven Beans and resource adapters)

For details on the resource adapter and J2EE application settings for executing message inflow, see *3.16.8 Settings for using the resource adapters conforming to the Connector 1.5 specifications*.

Also, a Transacted Delivery is enabled when the conditions described in the following table are satisfied.

Table 3-68: Conditions when a Transacted Delivery is enabled

J2EE server settings		Message-driven Bean settings		
		Transaction management type <transaction-type> is Container		Transaction management type <transaction-type> is Bean
		Transaction attribute <trans-attribute> is Required	Transaction attribute <trans-attribute> is NotSupported	
<code>ejbserver.distributedtx.XATransaction.enabled</code>	<code>true</code>	Y	N	N
	<code>false</code>	R	N	N

Legend:

Y: The EIS delivering the message is included in the global transaction and then the transaction is started.

R: The EIS delivering the message is not included in the transaction and the local transaction is started.

N: The transaction is not started.

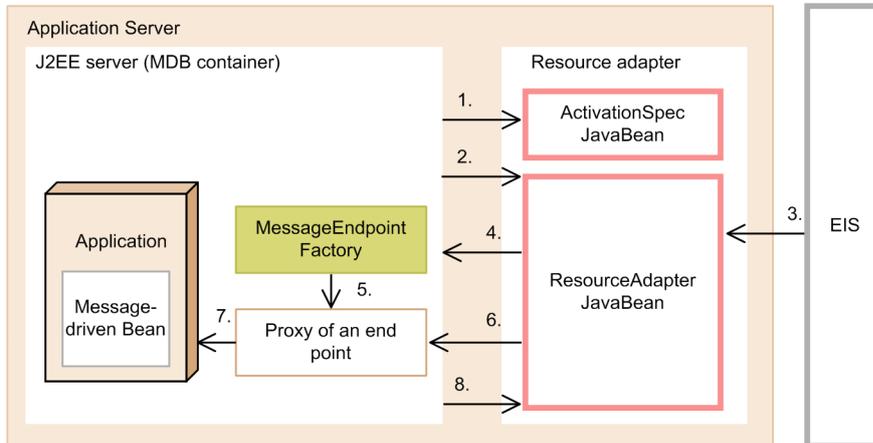
Also, to use a Transacted Delivery, you must use resource adapters conforming to the Connector 1.5 specifications and supporting Transacted Delivery. For details on the settings, see the documentation for the resource adapter in use.

(2) Procedure for controlling the message inflow (for a Non-Transacted Delivery)

A Non-Transacted Delivery is a message delivery in which the EIS that delivers the messages does not participate in the transaction.

The following figure shows the control procedure when the message inflow is used with a Non-Transacted Delivery.

Figure 3-49: Control procedure when the message inflow is used with a Non-Transacted Delivery



Legend:

: Class that is necessary to be implemented in the resource adapter

Note: The MDB container is a dedicated EJB container that has the functionality to execute the Message-driven Bean.

The following points describe the control executed when the message inflow is used with a Non-Transacted Delivery. Note that the item numbers correspond to the numbers in the figure.

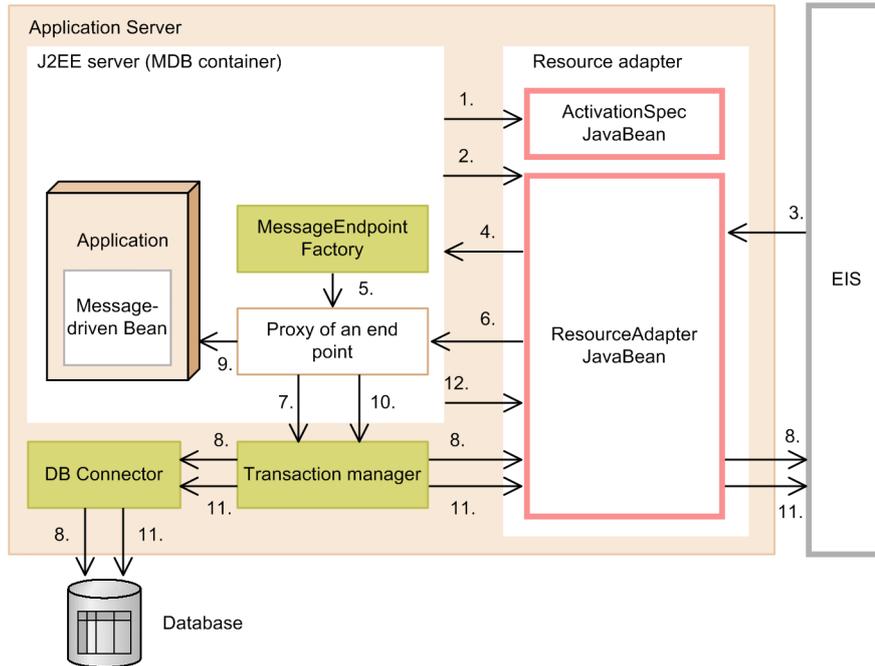
1. The properties are set up in `ActivationSpecJavaBean` of the resource adapter by the application start processing.
The content specified in the Message-driven Bean attribute `<activation-config>` is set up.
2. The J2EE server invokes the `endpointActivation` method. As a result, the starting of the message endpoint is reported to the resource adapter.
3. The EIS sends a message.
4. The resource adapter invokes the `createEndpoint` message for `MessageEndpointFactory`.
5. `MessageEndpointFactory` generates an endpoint proxy.
6. The resource adapter invokes a message listener method, such as the `onMessage` method, for the endpoint proxy.
7. The endpoint proxy invokes a message listener method, such as the `onMessage` method, for the Message-driven Bean.
8. When the processing is complete, the J2EE server invokes the `endpointDeactivation` method. As a result, the stopping of message endpoint is reported to the resource adapter.

(3) Procedure for controlling the message inflow (for a Transacted Delivery)

A Transacted Delivery is a message delivery in which the EIS that delivers the messages participates in the transaction.

The following figure shows the control procedure when the message inflow is used with a Transacted Delivery.

Figure 3-50: Control procedure when the message inflow is used with a Transacted Delivery



Legend:

: Class that is necessary to be implemented in the resource adapter

Note: The MDB container is a dedicated EJB container that has the functionality to execute the Message-driven Bean.

The following points describe the control executed when the message inflow is used with a Transacted Delivery. Note that the item numbers correspond to the numbers in the figure.

1. The properties are set up in ActivationSpecJavaBean of the resource adapter by the application start processing.
The content specified in the Message-driven Bean attribute <activation-config> is set up.
2. The J2EE server invokes the endpointActivation method. As a result, the starting of the message endpoint is reported to the resource adapter.
3. The EIS sends a message.
4. The resource adapter invokes the createEndpoint message for MessageEndpointFactory.
5. MessageEndpointFactory generates an endpoint proxy.
6. The resource adapter invokes a message listener method, such as the onMessage method, for the endpoint proxy.
7. The invoked endpoint proxy invokes a start transaction instruction for the transaction manager.
8. The transaction manager invokes the start method and starts the transaction.
9. The endpoint proxy invokes a message listener method, such as the onMessage method, for the Message-driven Bean.
10. The endpoint proxy instructs the conclusion of the transaction to the transaction manager.
11. The transaction manager invokes the transaction conclusion methods, such as the prepare method and commit method, to conclude the transaction.
12. When the processing is complete, the J2EE server invokes the endpointDeactivation method. As a result, the stopping of message endpoint is reported to the resource adapter.

(4) Deploying and undeploying the message endpoint

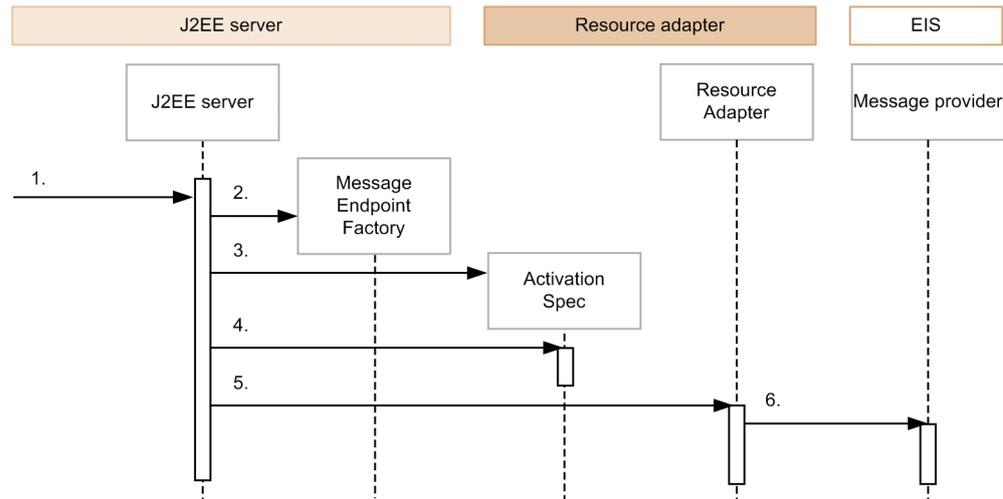
This section describes the processing executed for deploying and undeploying the message endpoint.

■ Deploying the message endpoint

This section describes the processing executed for deploying the message endpoint. The message endpoint is deployed when you start an application containing the Message-driven Beans when the resource adapter is running.

The following figure shows the processing for deploying the message endpoint.

Figure 3-51: Processing for deploying the message endpoint



The following points describe the processing executed for deploying the message endpoint. The item numbers correspond to the numbers in the figure.

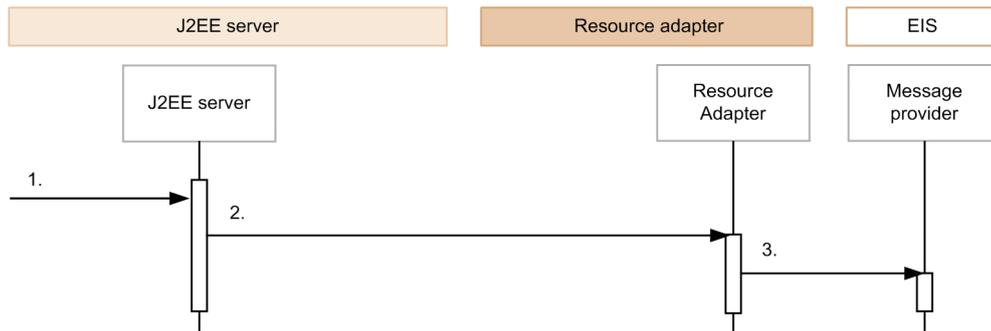
1. Start the J2EE application containing the Message-driven Beans when the resource adapter is running.
2. The J2EE server generates `MessageEndpointFactory`.
`MessageEndpointFactory` is an instance of `javax.resource.spi.endpoint.MessageEndpointFactory` provided by the J2EE server. `javax.resource.spi.endpoint.MessageEndpointFactory` is a factory class that provides an endpoint instance to the resource adapter.
3. The J2EE server generates `ActivationSpec`.
`ActivationSpec` is a `JavaBean` that sets up the information required for starting the Message-driven Beans (endpoint).
4. The J2EE server sets up the `ActivationSpec` property.
 The information specified in the `ActivationSpec` property is set up as an attribute of the application that contains the Message-driven Beans.
5. The J2EE server invokes the `javax.resource.spi.ResourceAdapter#endpointActivation(MessageEndpointFactory, ActivationSpec)` method.
 At this time, the generated and set up `MessageEndpointFactory` and `ActivationSpec` instances are specified as arguments. Note that if an exception occurs during the invocation of the `endpointActivation` method, the KDJE43174-E message is output and the starting of the application is canceled.
6. If the method specified in 5 is invoked, the resource adapter prepares to receive the messages from the message provider.

■ Undeploying the message endpoint

This section describes the processing executed for undeploying the message endpoint. The message endpoint is undeployed when you stop an application containing the Message-driven Beans.

The following figure shows the processing for undeploying the message endpoint.

Figure 3-52: Processing for undeploying the message endpoint



The following points describe the processing executed for undeploying the message endpoint. The item numbers correspond to the numbers in the figure.

1. Stop the J2EE application containing the Message-driven Beans.
2. The J2EE server invokes `javax.resource.spi.ResourceAdapter#endpointDeactivation(MessageEndpointFactory, ActivationSpec)`.
At this time, the same `MessageEndpointFactory` and `ActivationSpec` instances, which were specified for deployment, are specified as arguments. Note that if an exception occurs during the invocation of this method, the message KDJE43175-W is output. However, even if an exception occurs, the application stop processing continues.
3. If the method specified in 2 is invoked, the resource adapter executes the processing to terminate the receiving of messages from the message provider.

■ Processing of the resource adapter for delivering the messages

This section describes the processing of the resource adapter for delivering the messages.

The resource adapter invokes the message endpoint (Message-driven Bean) by using the message endpoint proxy. This proxy is obtained by invoking the `createEndpoint` method of `javax.resource.spi.endpoint.MessageEndpointFactory` from the resource adapter.

! Important note

- If you invoke the `MessageEndpointFactory` method after the application stops, the `javax.resource.spi.UnavailableException` exception is thrown. At this time, the message KDJE43177-E is output. Also, when the message endpoint method is invoked, the `java.lang.IllegalStateException` exception is thrown. At this time, the message KDJE43177-E is output.
- In a Transacted Delivery, if you stop a J2EE application during message delivery, the transaction might be rolled back. At this time, KDJE31011-E or KDJE31012-E error might occur. These errors might also occur when you stop the running J2EE application when the J2EE server is stopped.

3.16.4 Transaction inflow

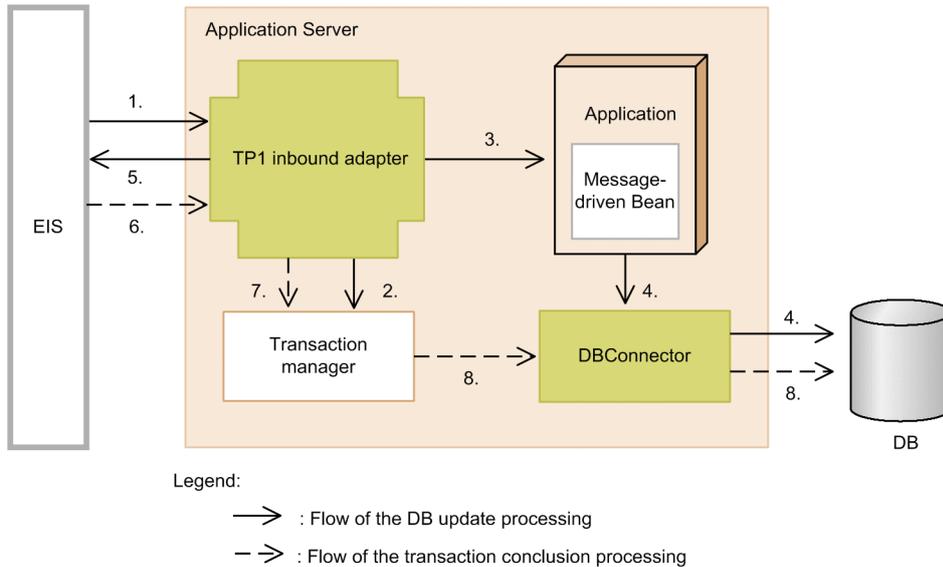
A *transaction inflow* is a contract between the resource adapters and Application Server when the message endpoint (Message-driven Bean) on Application Server is involved in a message provider transaction.

The message endpoint (Message-driven Bean) participates in the transaction associated with the transaction identifier of the message provider.

Note that a transaction inflow is only available with the TP1 inbound integrated function.

The following figure shows the procedure for controlling the transaction inflow.

Figure 3-53: Procedure for controlling the transaction inflow



The following points describe the flow of processing in the figure. Note that the item numbers correspond to the numbers in the figure.

1. The EIS sends a message to the TP1 inbound adapter.
2. The TP1 inbound adapter propagates the EIS transaction identifier to the transaction manager.
3. The TP1 inbound adapter executes the Message-driven Bean.
4. An SQL statement is executed from the Message-driven Bean and the DB is updated.
5. After the execution of Message-driven Bean, the TP1 inbound adapter sends the execution results to the EIS.
6. After receiving the execution results, the EIS sends the instruction to conclude the transaction to the TP1 inbound adapter.
7. The TP1 inbound adapter sends the instruction to conclude the transaction to the transaction manager.
8. The transaction is concluded.

3.16.5 Looking up Administered objects

You can obtain the Administered objects (`AdminObject`) using lookup. An Administered object is required to obtain the information of the message destination for sending messages from inside a J2EE application and for receiving the messages synchronously. To look up an Administered object, you must specify the resource adapter and J2EE application settings. This subsection gives an overview of the settings.

Reference note

The specifications of the Administered object depend on the resource adapter specifications. For details, follow the specifications of the resource adapter in use.

(1) Setting up the Administered objects to be looked up

You set up the information for the Administered objects to be looked up, as the resource adapter properties. Specify the resource adapter properties with the HITACHI Connector Property file.

To set an Administered object as a lookup target, make sure you specify the name of the Administered object in the `<adminobject-name>` tag. When multiple Administered objects are defined for each resource adapter, the Administered object name is used to identify the Administered object uniquely.

For details on the settings, see 3.16.8 *Settings for using the resource adapters conforming to the Connector 1.5 specifications*.

(2) J2EE application settings

The J2EE application settings for looking up an Administered object can be specified in the property files or annotations.

(a) When using a property file

Specify the following elements beneath the `<resource-env-ref>` tag:

Specify the name to be used for lookup beneath the `<resource-env-ref-name>` tag. Specify the type of the Administered object that will be referenced beneath the `<resource-env-ref-type>` tag. Specify the Administered object name of the resource adapter and the resource adapter display name beneath the `<linked-adminobject>` tag.

! Important note

In EJB 2.1 and later, the `<message-destination-ref>` tag provided as the element for referencing the Administered objects is not available with Application Server.

For details on the settings, see *3.16.8 Settings for using the resource adapters conforming to the Connector 1.5 specifications*.

(b) When using an annotation

Specify the name of the Administered object to be looked up in the `mappedName` attribute of `@Resource`. Specify the name in the following format. Use `!#` as the delimiter between the resource adapter display name and Administered object name.

```
@Resource(mappedName="resource-adapter-display-name!# Administered-object-name")
```

3.16.6 Specifying multiple connection definitions

When you use the resource adapters conforming to the Connector 1.5 specifications, you can specify multiple connection definitions for one resource adapter. You can specify the settings, such as those for the connection pool and log output, for each connection definition.

To identify a connection definition in a resource adapter, you use the **connection definition identifier**. The connection identifier is the value specified in `<connectionfactory-interface>` in the DD. The `<connectionfactory-interface>` value has a unique value for each connection definition in the resource adapter.

Note that you can check the connection definition identifier for each connection definition included in the resource adapter if you execute the following server management commands by specifying the `-outbound` option:

- `cjlistapp` command (for the RAR files included in an application)
For details, see *cjlistapp (Displaying a list of applications)* in the *uCosminexus Application Server Command Reference Guide*.
- `cjlistres` command
For details, see *cjlistres (Displaying a list of resources)* in the *uCosminexus Application Server Command Reference Guide*.
- `cjlistrar` command
For details, see *cjlistrar (Displaying a list of resource adapters)* in the *uCosminexus Application Server Command Reference Guide*.

This subsection describes the elements that can be specified in the connection definitions and the locations for specifying these elements. This subsection also describes the points to remember when you specify multiple connection definitions.

(1) Elements specifiable in the connection definitions and the location of specification

This section describes the elements that can be specified in the connection definitions and the locations for specifying these elements.

■ Location of specification in the DD and the elements that can be specified in the connection definition

You specify the connection definition beneath `<connection-definition>` in the DD complying with the Connector 1.5 specifications. The elements specified beneath `<connection-definition>` are as follows:

- `<managedconnectionfactory-class>`
- `<config-property>`
- `<connectionfactory-interface>`
- `<connectionfactory-impl-class>`
- `<connection-interface>`
- `<connection-impl-class>`

■ Mapping the hierarchical structure and connection definitions of the HITACHI Connector Property file

With the change in the hierarchical structure in the DD (`ra.xml`), a hierarchy corresponding to the connection definition is added in the property file as well.

For the resource adapters already imported into the J2EE server, you use the HITACHI Connector Property file to change the resource adapter properties. The values defined in the HITACHI Connector Property file are applied to the resource adapter on the J2EE server by using the server management commands. For details on the procedure for using the property file to specify properties, see 3.5 *Specifying properties using property files* in the *uCosminexus Application Server Application Setup Guide*.

Note that the resource adapter operation-related properties include items defined for each connection definition and items defined for the entire resource adapter. You specify the items defined for each connection definition beneath `<outbound-resourceadapter>-<connection-definition>` in the HITACHI Connector Property file. You specify the items defined for the entire resource adapter beneath `<resourceadapter-runtime>-<property>`.

For details on the method of specification, see 4.1.1 *Content specified in the HITACHI Connector Property file* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

■ How to define link resolution in J2EE applications (method of specifying in `<resource-ref>-<linked-to>`, `<cmp-map>-<datasource-name>`, or `mappedName`)

With the J2EE applications that reference the resource adapters conforming to the Connector 1.5 specifications, you must define the connection definition that will be referenced by the J2EE application for link resolution.

Specify the definition using one of the following methods:

- `<resource-ref>-<linked-to>` in the J2EE application property files (WAR property file, Session Bean property file, Entity Bean property file, or MessageDrivenBean property file)
- `<cmp-map>-<datasource-name>` in the J2EE application property file (Entity Bean property file)
- `mappedName` attribute of the annotation `@Resource`

! Important note

If the referenced resource is a resource adapter conforming to the Connector 1.5 specifications, make sure you specify the connection definition identifier. If the connection definition identifier is omitted, the link resolution fails. Also, if the referenced resource is a resource adapter conforming to the Connector 1.0 specifications, do not specify the connection definition identifier. If the connection definition identifier is specified, the link resolution fails.

Specify a referenced resource containing a connection definition identifier in the following format:

```
resource-adapter-display-name!connection-definition-identifier
```

A description of the specified contents is as follows:

resource-adapter-display-name

The value of the `<connector>-<display-name>` element in the DD (`ra.xml`) of the referenced resource adapter.

connection-definition-identifier

The value of the `<connector>-<resourceadapter>-<outbound-resourceadapter>-<connection-definition>-<connectionfactory-interface>` element in the DD (`ra.xml`) of the referenced resource adapter.

Note that you can also check the resource adapter display name and connection definition identifier with the following server management commands:

- `cjlistapp` command (for the RAR files included in an application)
- `cjlistres` command
- `cjlistrar` command

(2) Points to remember when using a connection pool

With a resource adapter for which multiple connection definitions are specified, you can manage the connection pools for the connection definitions. You specify the connection pool definition with the elements beneath `<outbound-resourceadapter>-<connection-definition>-<connector-runtime>-<property>` in the HITACHI Connector Property file.

The points to remember when there are multiple connection pools are as follows:

■ **Executing the commands for connection pools**

When you execute the server management commands for connection pools, you must specify the connection pool that you want to target.

When you execute the following commands, you must specify the connection definition identifier. You specify the connection definition identifier in the options of these commands.

- `cjclearpool` command
For details, see *cjclearpool (Deleting the connections in the connection pool)* in the *uCosminexus Application Server Command Reference Guide*.
- `cjlistpool` command
For details, see *cjlistpool (Displaying the list of connection pools)* in the *uCosminexus Application Server Command Reference Guide*.

Reference note

The commands used for connection pool clustering are not applicable. The commands used for connection pool clustering can only be executed with the resource adapters provided by Application Server.

■ **Processing for connection pool warming up**

The processing for connection pool warming up is executed for each connection definition.

When you execute the processing for connection pool warming up, even if connection pool warming up fails for a specific connection definition, the warming up processing for the other connection pools continues.

(3) Points to remember when a transaction is recovered

You execute transaction recovery for each connection definition. Therefore, if multiple connection definitions are specified, a resource is registered for each connection definition in the OTS.

(4) Points to remember when a connection test is performed for resources

When you execute a connection test for a resource adapter, even if an error occurs in a connection test for a specific connection definition, the processing is not cancelled. The connection tests are executed for all the connection definitions. However, if an error occurs in one of the connection definitions, the return value of the command is a return value indicating an abnormal termination.

(5) Points to remember when the operation information and the information for resource depletion monitoring is output

When multiple connection definitions are specified, the operation information and the information for resource depletion monitoring are output as follows:

Operation information (information output to the operation information file)

Only the information for the first connection definition is output. The first connection definition is the connection definition that is defined first in the DD (`ra.xml`).

Operation information (information output by using the management commands)

This information cannot be output. If an attempt is made to output the operation information for the resource adapters conforming to the Connector 1.5 specifications, the output content is not guaranteed.

Information for resource depletion monitoring

The information for all the connection definitions specified in the resource adapter is output.

(6) Points to remember when the resource adapter operation log is output

When the operation log of the resource adapters conforming to the Connector 1.5 specifications is output, the name of the output file is in the following format:

```
resource-adapter-display-name_connection-definition-order_serial-number-of-log-file.log
```

The *connection-definition-order* corresponds to the order in which the corresponding `<connection-definition>` occurs (1, 2, ...) in the DD (`ra.xml`).

3.16.7 Application Server-specific Connector 1.5 API specifications

This subsection describes the Application Server-specific specifications for the interfaces of the Connector 1.5 specifications.

(1) javax.resource.spi.endpoint.MessageEndpointFactory interface

Application Server provides specifications for two methods.

(a) createEndpoint method**Format**

```
public MessageEndpoint createEndpoint(XAResource xaResource) throws
    UnavailableException
```

Application Server-specific specifications

- If this method is invoked after you start the stop processing of a J2EE application containing the EJBs, `javax.resource.spi.UnavailableException` is thrown.
- The processing to obtain the Message-driven Bean instances is executed in this method. If the upper limit of the Message-driven Bean instance pool is reached, the Message-driven Bean instances can be obtained, or the method is not returned until the J2EE application stops.

Exception

The following table describes the exception operations provided with Application Server.

Exception	Exception timing
<code>javax.resource.spi.UnavailableException</code>	<ul style="list-style-type: none"> • When this method is invoked after you start the stop processing of a J2EE application containing the EJBs • When an attempt to obtain the Message-driven Bean instances fails

(b) `isDeliveryTransacted` method**Format**

```
public boolean isDeliveryTransacted(Method method) throws
NoSuchMethodException
```

Application Server-specific specifications

If you specify `true` in the `ejbserver.distributedtx.XATransaction.enabled` property of the user property file for J2EE servers, `true` is returned when `Required` is specified for CMT in the transaction attribute of the method passed by the argument.

(2) `javax.resource.spi.endpoint.MessageEndpoint` interface

Application Server provides specifications for three methods.

(a) `beforeDelivery` method**Format**

```
public void beforeDelivery(Method method) throws NoSuchMethodException,
ResourceException
```

Application Server-specific specifications

The specifications provide for exceptions.

Exception

The following table describes the exception operations provided with Application Server.

Exception	Exception timing
<code>java.lang.IllegalStateException</code>	<ul style="list-style-type: none"> When you invoke the message listener method after you start the stop processing of the J2EE application containing the EJBs When the <code>release()</code> method has already been invoked
<code>javax.resource.spi.IllegalStateException</code>	<ul style="list-style-type: none"> When the methods are invoked in an invalid order (when the <code>beforeDelivery</code> method is invoked again after invoking the <code>beforeDelivery</code> method or when the <code>beforeDelivery</code> method is invoked after invoking the message listener method)
<code>javax.resource.spi.UnavailableException</code>	<ul style="list-style-type: none"> When an unexpected exception occurs

(b) `afterDelivery` method**Format**

```
public void afterDelivery() throws ResourceException
```

Application Server-specific specifications

The specifications provide for exceptions.

Exception

The following table describes the exception operations provided with Application Server.

Exception	Exception timing
<code>java.lang.IllegalStateException</code>	<ul style="list-style-type: none"> When you invoke the message listener method after you start the stop processing of the J2EE application containing the EJBs When the <code>release()</code> method has already been invoked

Exception	Exception timing
<code>javax.resource.spi.IllegalStateException</code>	<ul style="list-style-type: none"> When the methods are invoked in an invalid order (when <code>afterDelivery()</code> is invoked at a different time after the message listener method Option B[#] is invoked)
<code>javax.resource.spi.UnavailableException</code>	<ul style="list-style-type: none"> When an unexpected exception occurs

#: A message delivery option coded in the Connector 1.5 specifications.

(c) release method

Format

```
public void release()
```

Application Server-specific specifications

If you invoke this method, the Message-driven Bean instance associated with the endpoint is released and returns to the instance pool.

When you finish using the endpoint, make sure you invoke this method and return the Message-driven Beans to the instance pool.

Exception

The following table describes the exception operations provided with Application Server.

Exception	Exception timing
<code>java.lang.IllegalStateException</code>	<ul style="list-style-type: none"> When you invoke the message listener method after you start the stop processing of the J2EE application containing the EJBs When the <code>release()</code> method has already been invoked

(3) Message listener method

The interface is defined with the message listener interface. The following table describes the exception operations provided with Application Server.

Exception	Exception timing
Exception specified in the message listener interface	<ul style="list-style-type: none"> Message-driven Bean execution time (other than when the system exception occurs)
<code>javax.ejb.EJBException</code>	<ul style="list-style-type: none"> Message-driven Bean execution time (when the system exception occurs)
<code>java.lang.IllegalStateException</code>	<ul style="list-style-type: none"> When you invoke the message listener method after you start the stop processing of the J2EE application containing the EJBs When the <code>release()</code> method has already been invoked When the methods are invoked in an invalid order (the message listener method is invoked continuously after invoking the message listener method in the message delivery of Option B[#].)

#: A message delivery option coded in the Connector 1.5 specifications.

3.16.8 Settings for using the resource adapters conforming to the Connector 1.5 specifications

This subsection describes the J2EE application and resource adapter settings for using the resource adapters conforming to the Connector 1.5 specifications.

You specify the settings in the HITACHI Application Integrated Property File[#] and HITACHI Connector Property file respectively.

#

The description in this section is entirely based on the use of HITACHI Application Integrated Property File. If you use the `cjgetappprop` command to obtain the Message-driven Bean property file, you can specify the settings in the Message-driven Bean property file. If you are using the Message-driven Bean property file, read "beneath the `<hitachi-connector-property>` tag in HITACHI Application Integrated Property File" as "Message-driven Bean property file".

(1) Items that can be set up

This section describes the items that can be set up when you use the resource adapters conforming to the Connector 1.5 specifications.

(a) J2EE application settings

The following table describes the content that can be set up in a J2EE application.

Table 3-69: J2EE application settings (for using the resource adapters conforming to the Connector 1.5 specifications)

Functionality	Item	Target	Settings
Looking up Administered objects	Information used for looking up Administered objects	WAR, Session Bean, Entity Bean, and Message-driven Bean	Specify the settings in the following tags beneath the <code><resource-env-ref></code> tag: <ul style="list-style-type: none"> In the <code><resource-env-ref-name></code> tag, specify the name to be used for lookup. In the <code><resource-env-ref-type></code> tag, specify the type of the Administered object. In the <code><linked-adminobject></code>-<code><resourceadapter-name></code> tag, specify the display name of the resource adapter in which the Administered object is included. In the <code><linked-adminobject></code>-<code><adminobject-name></code> tag, specify the object name of the Administered object.
Message inflow [#]	Mapping with the resource adapter	Message-driven Bean	In the <code><resource-adapter></code> tag beneath the <code><message-ref></code> - <code><connection-destination></code> tag, specify the mapped resource adapter display name.
	Interfaces used by the Message-driven Beans	Message-driven Bean	In the <code><messaging-type></code> tag, specify the interfaces implemented by the Message-driven Beans.
	ActivationSpec settings	Message-driven Bean	In the <code><activation-config-property-name></code> tag and <code><activation-config-property-value></code> tag beneath the <code><activation-config></code> - <code><activation-config-property></code> tag, specify the properties to be set for ActivationSpec.

#: Some of the set values must match the values set for the resource adapter to be used.

(b) Resource adapter settings

From among the items specified as the resource adapter properties, this section describes the main items that can be set up for the resource adapters conforming to the Connector 1.5 specifications and how to set up these items. Note that this section describes content specific to the resource adapters conforming to the Connector 1.5 specifications.

The following table describes the content that can be set up as the properties of the resource adapters conforming to the Connector 1.5 specifications.

Table 3-70: Content that can be set up in the property definitions of the resource adapters conforming to the Connector 1.5 specifications

Category	Item	Settings
Administered object	Administered object names #1	In the <code><adminobject></code> - <code><adminobject-name></code> tag #2, specify the object name to be used for lookup. Make sure you specify this item when you want to look up an Administered object. You cannot include continuous underscores "___" in the object name. Note that you must also specify the J2EE application settings to look up an Administered object. See (a) <i>J2EE application settings</i> .
	Interfaces	In the <code><adminobject></code> - <code><adminobject-interface></code> tag, specify the interface to be used. Specify items such as <code>javax.jms.Queue</code> or <code>javax.jms.Topic</code> .
	Implementation classes and properties	In the <code><adminobject-class></code> tag, specify the implementation class. Specify the Administered object properties beneath the <code><config-property></code> tag.

#1: This need not be specified for the Administered objects used with message inflow.

#2: This tag does not exist in the DD (`ra.xml`).

(2) Setting up the Administered objects

You code the settings for the Administered objects to be used with the message inflow in the HITACHI Connector Property file.

The content set up for the Administered object depends on the resource adapter. See the documentation for the resource adapter in use.

You map the Administered objects to the Message-driven Beans. For details on the mapping, see (5) *ActivationSpec settings*.

(3) Settings for mapping the Message-driven Beans and resource adapters

You specify the mapping between the Message-driven Beans and resource adapters in the `<resource-adapter>` tag of HITACHI Application Integrated Property File. Specify the resource adapter display name.

The hierarchy of `<resource-adapter>` is as follows:

```
<hitachi-message-bean-property>
  <message-ref>
  <connection-destination>
  <resource-adapter>
```

These tags are elements that do not exist in the DD (`ejb-jar.xml`). However, `<hitachi-message-bean-property>` corresponds to `<message-driven>` in the DD.

The mapping with the resource adapters is executed when you start the J2EE applications containing the Message-driven Beans. Note that the J2EE applications containing the Message-driven Beans fail to start in the following cases. In this case, the KDJE42359-E message is output:

- When the resource adapter display name is not specified in the `<resource-adapter>` tag
- When the resource adapter with the display name specified in the `<resource-adapter>` tag has not been started
- When the resource adapter with the display name specified in the `<resource-adapter>` tag is not found

The Message-driven Beans must be mapped to the Administered objects as well. For details on the mapping, see (5) *ActivationSpec settings*.

Reference note

You cannot specify the information of the `<message-driven>`-`<message-destination-link>` tag with Application Server.

(4) Setting up the interfaces used by the Message-driven Beans

You code the interfaces used by the Message-driven Beans in the `<messaging-type>` tag of HITACHI Application Integrated Property File. Specify the names of the interfaces implemented by the Message-driven Beans.

The hierarchy of `<messaging-type>` is as follows:

```
<hitachi-message-bean-property>
  <messaging-type>
```

The `<hitachi-message-bean-property>` tag corresponds to `<message-driven>` in the DD (`ejb-jar.xml`).

You must specify the interfaces supported by the resource adapter as the interfaces used by the Message-driven Beans. The interfaces supported by the resource adapter are specified in the `<messagelistener-type>` tag of the DD (`ra.xml`) for the resource adapter. If you specify an interface other than that specified in this tag, the J2EE application containing the Message-driven Beans fails to start. In this case, the KDJE43167-E message is output.

You can also check the interfaces supported by the resource adapter with the following server management commands. Execute these commands by specifying `-resname` and `-inbound` in the arguments.

- `cjlistres` command
For details, see *cjlistres (Displaying a list of resources)* in the *uCosminexus Application Server Command Reference Guide*.
- `cjlistrar` command
For details, see *cjlistrar (Displaying a list of resource adapters)* in the *uCosminexus Application Server Command Reference Guide*.
- `cjlistapp` command
For details, see *cjlistapp (Displaying a list of applications)* in the *uCosminexus Application Server Command Reference Guide*.

Note that if no value is specified in the `<messaging-type>` tag, the `javax.jms.MessageListener` interface is used as the default value.

Reference note

The `javax.jms.MessageListener` interface was used as a specific message listener interface until EJB 2.0.

(5) ActivationSpec settings

`ActivationSpec` is a `JavaBean` that contains settings required for activating the Message-driven Beans. `ActivationSpec` contains properties that must be set up according to the resource adapter settings.

(a) Overview of the ActivationSpec settings

You code the values to be set up for `ActivationSpec` beneath the `<activation-config>` tag of HITACHI Application Integrated Property File. Specify the property name and the property value. You can specify multiple values.

The hierarchy of `<activation-config>` is as follows:

```
<hitachi-message-bean-property>
  <activation-config>
    <activation-config-property>
      <activation-config-property-name>
      <activation-config-property-value>
```

You can specify the default `ActivationSpec` settings in the DD (`ra.xml`) for the resource adapter. The value set in the `<resourceadapter><config-property>` tag is used as the default value. When the default value is mentioned and if you set the `ActivationSpec` value in HITACHI Application Integrated Property File, the value is overwritten by the value specified beneath the `<hitachi-message-bean-property>` tag.

The properties that you must set up as the `ActivationSpec` properties are described in (b) *Properties that must be set up*.

(b) Properties that must be set up

With the `ActivationSpec` settings, you must set up the values of the properties specified in the `<activation-spec><required-config-property>` tag in the DD (`ra.xml`) for the resource adapter. You can also use the following server management commands to check the properties that must be set up. Execute these commands by specifying `-resname` and `-listenertype` in the arguments:

- `cjlistres` command
- `cjlistrar` command
- `cjlistapp` command

The other properties that can be set up for `ActivationSpec` depend on the resource adapter. See the documentation for the resource adapter in use.

(c) Properties that must be set up when a message listener supporting the JMS is used

When you use a message listener (message listener that uses the `javax.jms.MessageListener` interface) that supports the JMS, the following properties must be specified in `ActivationSpec`. Specify the properties beneath the `<activation-config>` tag.

- `destination`
- `destinationType`

Note that Application Server does not check whether these properties are specified. You check whether the property settings are appropriate with the `ActivationSpec#validate` method provided by the resource adapter.

Reference note

The following items specified as elements beneath the `<message-driven>` tag in EJB 2.0 have been deleted in EJB version 2.1 and later:

- `<message-selector>`
- `<acknowledge-mode>`
- `<message-driven-destination><subscription-durability>`

The Connector 1.5 specifications recommend that these elements can be specified as properties beneath the `<activation-config>`. However, whether these elements can be specified depends on the resource adapter in use.

For details on the `ActivationSpec` specifications when you use a message listener that supports the JMS, see the documentation for the Connector 1.5 specifications.

(d) Errors that occur during `ActivationSpec` generation and setup

When you generate and set up `ActivationSpec`, errors occur when the states described in the following table are reached. If an error occurs, the J2EE application fails to start.

The following table describes the cases in which errors occur and the messages that are output.

Table 3-71: Mapping the cases in which errors occur and the output messages

Cases in which errors occur	Output messages
An attempt to generate an instance of the <code>ActivationSpec</code> implementation class fails	KDJE43168-E

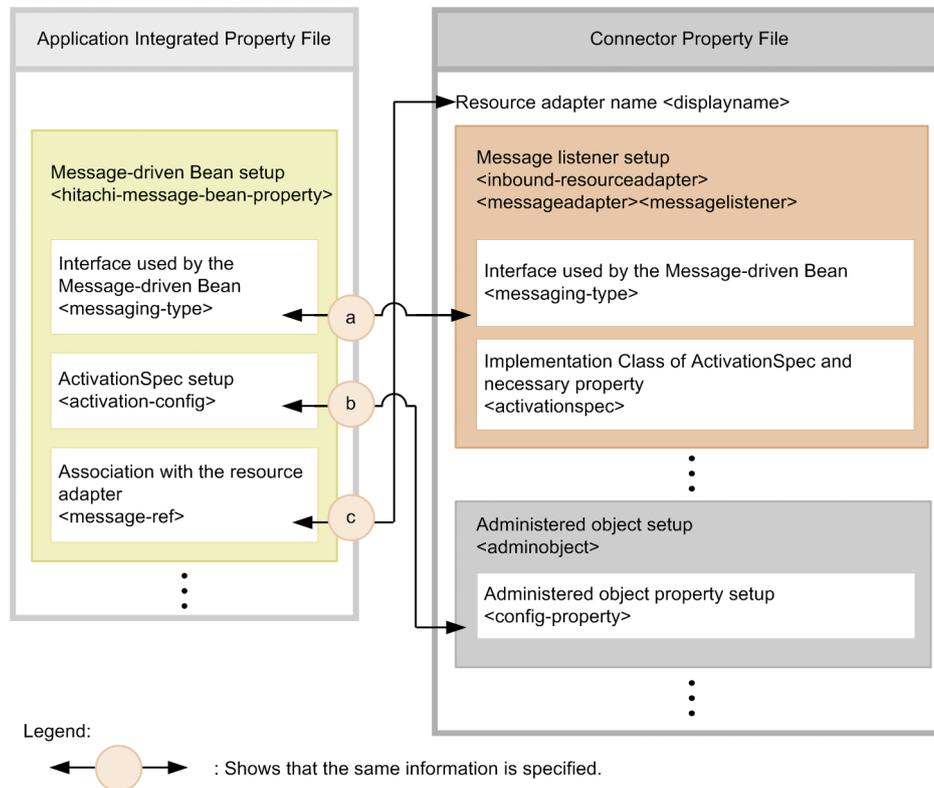
Cases in which errors occur	Output messages
If the <code>javax.resource.spi.ActivationSpec</code> interface is not implemented for a class specified in <code>ActivationSpec</code>	KDJE43172-E
If the setter method of the property corresponding to the <code><activation-config-property-name></code> tag does not exist in <code>ActivationSpec</code>	KDJE43169-E
If an exception is thrown in the invocation of the method that checks whether the settings are appropriate (<code>ActivationSpec#validate</code> method), after the properties are set up in <code>ActivationSpec</code>	KDJE43170-E
If the settings corresponding to the property <code><required-config-property></code> tag specified in the DD (<code>ra.xml</code>) for the resource adapter are not coded as the <code><activation-config></code> tag of HITACHI Application Integrated Property File	KDJE43171-E
If an exception is thrown in the <code>setResourceAdapter</code> method of <code>ActivationSpec#</code>	KDJE43173-E

#: The `setResourceAdapter` method is a method defined in the `javax.resource.spi.ResourceAdapterAssociation` interface. The `javax.resource.spi.ResourceAdapterAssociation` interface is a super interface of the `javax.resource.spi.ActivationSpec` interface.

3.16.9 Examples of property file specification

This subsection uses examples to describe how to specify the property files when you execute the message inflow. The settings with the HITACHI Connector Property file and HITACHI Application Integrated Property File must match the settings at the locations shown in the following figure.

Figure 3-54: Locations specifying the same values in the HITACHI Connector Property file and HITACHI Application Integrated Property File



The specified contents are as follows:

- In a, you specify the interfaces used by the Message-driven Beans.
- In b, you specify the mapping of `ActivationSpec` and `Administered` objects.
- In c, you specify the mapping of the Message-driven Beans and resource adapters.

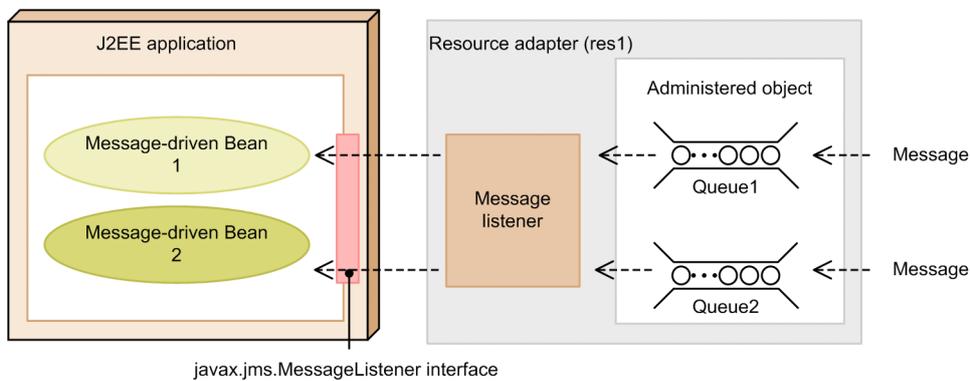
The following points are described based on the content specified from a to c:

(1) For the Message-driven Beans and resource adapters using the `javax.jms.MessageListener` interface

The examples of property file specification when you use the `javax.jms.MessageListener` interface as the message listener interface are as follows.

The description uses the configuration shown in the following figure as an example. In this example, two Message-driven Beans corresponding to the JMS receive messages from different Administered objects (`javax.jms.Queue`) respectively.

Figure 3-55: Example configuration of the Message-driven Beans and resource adapter using the `javax.jms.MessageListener` interface



The following is an example specification of HITACHI Application Integrated Property File.

Figure 3-56: Example specification of HITACHI Application Integrated Property File (for using the javax.jms.MessageListener interface)



The following is a description of the figure:

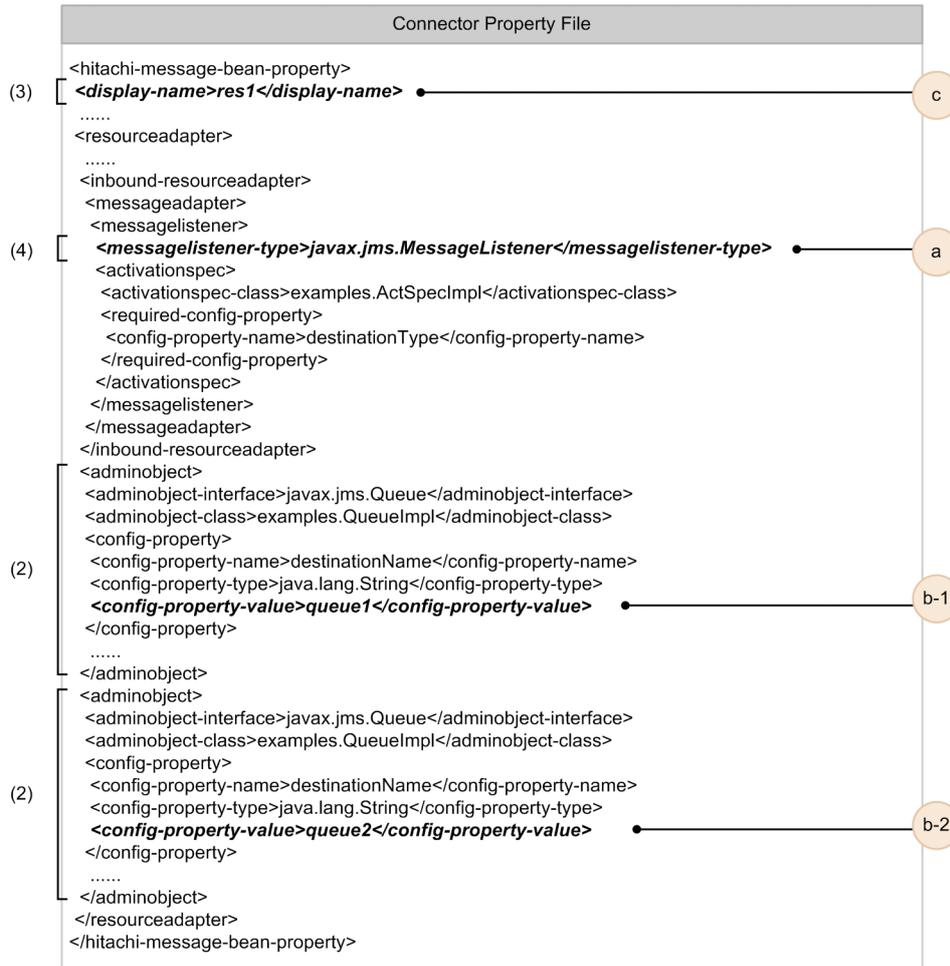
The points (3) to (5) in the figure correspond to the item numbers described in 3.16.8 *Settings for using the resource adapters conforming to the Connector 1.5 specifications*. Also, a, b-1, b-2, and c indicate the following settings respectively:

- a: Interfaces used by the Message-driven Beans
- b-1, b-2: Mapping of ActivationSpec and Administered object
- c: Mapping of the Message-driven Beans and resource adapter

Note that the values set up for a, b-1, b-2, and c correspond to Figure 3-56.

The following figure shows an example specification of the HITACHI Connector Property file.

Figure 3-57: Example specification of the HITACHI Connector Property file (for using the javax.jms.MessageListener interface)



The following is a description of the figure:

The points (2) to (4) in the figure correspond to the item numbers described in 3.16.8 *Settings for using the resource adapters conforming to the Connector 1.5 specifications*. Also, a, b-1, b-2, and c indicate the following settings respectively:

- a: Interfaces used by the Message-driven Beans
- b-1, b-2: Mapping of ActivationSpec and Administered object
- c: Mapping of the Message-driven Beans and resource adapter

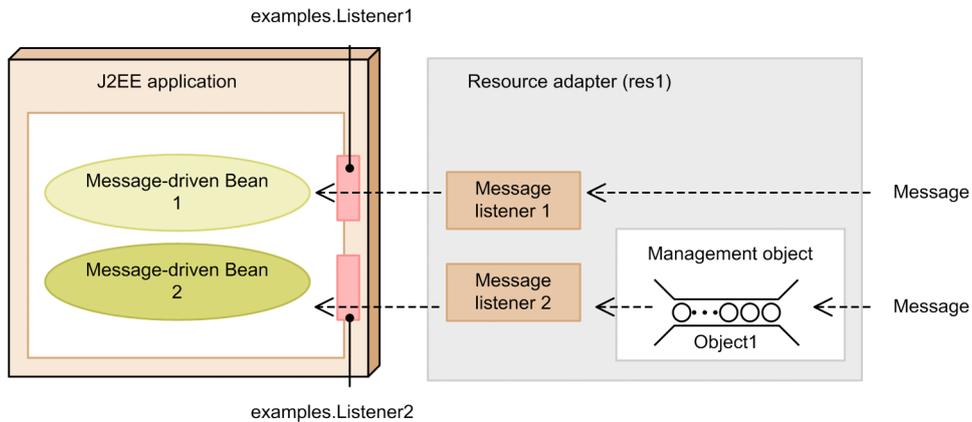
Note that the values set up for a, b-1, b-2, and c correspond to Figure 3-57.

(2) For the Message-driven Beans and resource adapters using any message listener interface

The examples of property file specification when you use any interface as the message listener interface are as follows.

The description uses the configuration shown in the following figure as an example.

Figure 3-58: Example configuration of the Message-driven Beans and resource adapter using any message listener interface

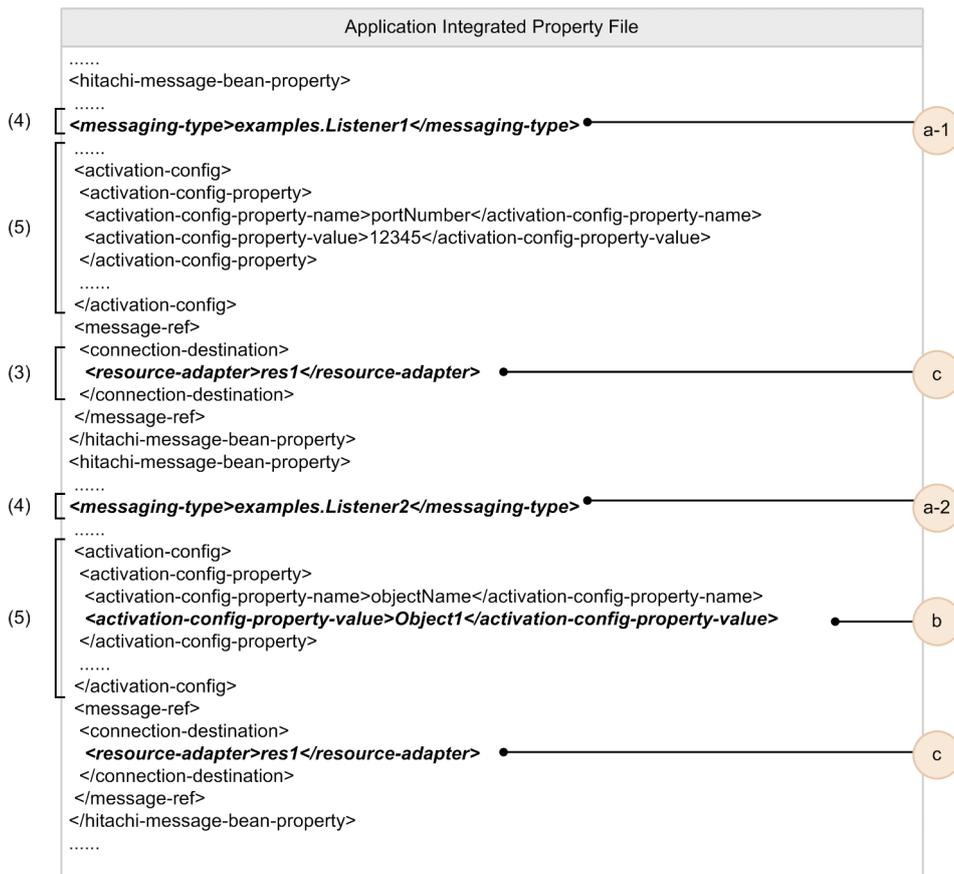


In this example, the resource adapter supports the following two unique interfaces:

- examples.Listener1 is a message listener interface used independently of the Administered objects.
- examples.Listener2 is a message listener interface used in association with the Administered objects.

The following figure shows an example specification of HITACHI Application Integrated Property File.

Figure 3-59: Example specification of HITACHI Application Integrated Property File (for using any message listener interface)



The following is a description of the figure:

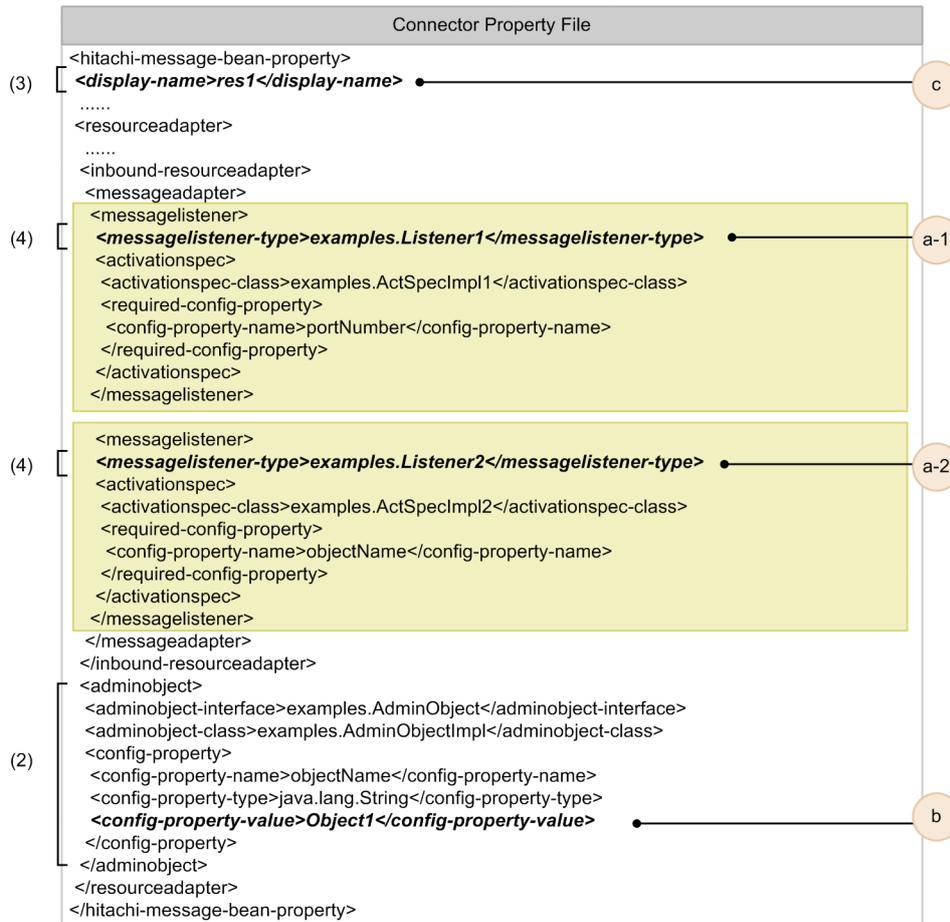
The points (3) to (5) in the figure correspond to the item numbers described in 3.16.8 *Settings for using the resource adapters conforming to the Connector 1.5 specifications*. Also, a-1, a-2, b, and c indicate the following settings respectively:

- a-1, a-2: Interfaces used by the Message-driven Beans
- b: Mapping of ActivationSpec and Administered object
- c: Mapping of the Message-driven Beans and resource adapter

Note that the values set up for a-1, a-2, b, and c correspond to Figure 3-59.

The following figure shows an example specification of the HITACHI Connector Property file.

Figure 3-60: Example specification of the HITACHI Connector Property file (for using any message listener interface)



Legend:

: This is the range of the definition of each message listener.

The following is a description of the figure:

The points (2) to (4) in the figure correspond to the item numbers described in 3.16.8 *Settings for using the resource adapters conforming to the Connector 1.5 specifications*. Also, a-1, a-2, b, and c indicate the following settings respectively:

- a-1, a-2: Interfaces used by the Message-driven Beans
- b: Mapping of ActivationSpec and Administered object
- c: Mapping of the Message-driven Beans and resource adapter

Note that the values set up for a-1, a-2, b, and c correspond to Figure 3-60.

3.16.10 Notes on using the resource adapters conforming to the Connector 1.5 specifications

The notes on using the resource adapters conforming to the Connector 1.5 specifications are as follows. The notes vary depending on how you use the resource adapters.

(1) Common notes

- You cannot obtain operation information related to the resource adapters conforming to the Connector 1.5 specifications even if you use the management portal or the `mngsvrutil` command.
- If you use the management portal or the `mngsvrutil` command to obtain status information of the resource adapters conforming to the Connector 1.5 specifications, information indicating that the resource adapter is stopped is returned even if the resource adapter is running.
- If you execute the `cmx_stop_resource` command for the resource adapters conforming to the Connector 1.5 specifications, the resource adapter is not stopped even if the command is always successful.

(2) When you deploy and use the resource adapter as a J2EE resource adapter (when the resource adapter is not included in the J2EE application)

When you specify an Administered object in an annotation, you must add the Administered object interface in the container extension library.

For details on the container extension library, see *14. Container Extension Library*.

(3) When you include and use the resource adapter in the J2EE application

- When you use the reload functionality, you must add the following classes (interfaces) in the container extension library:
 - Implementation class of the `javax.resource.spi.ActivationSpec` interface
 - Message listener interface supported by the resource adapter
- When you specify an Administered object in an annotation, you must add the Administered object interface in the container extension library.
- The global transaction is not available when you use the Inbound resource adapter.

For details on the container extension library, see *14. Container Extension Library*.

3.17 Functionality for connection pool clustering

This section describes the connection pool clustering functionality.

The following table describes the organization of this section.

Table 3-72: Organization of this section (Connection pool clustering functionality)

Category	Title	Reference location
Explanation	Connecting to Oracle using Oracle RAC	3.17.1
	Overview of connection pool clustering	3.17.2
	Resource adapters used	3.17.3
	Connection pool clustering operations	3.17.4
	Procedure for stopping or starting a connection pool manually	3.17.5
Settings	Settings required for clustering a connection pool	3.17.6

Note: The functionality-specific explanation is not available for "Implementation" and "Operations".

The connection pool clustering functionality optimizes the operations in a system where the database is used in a cluster configuration. You can use this functionality when you connect to Oracle using Oracle RAC. By using the connection pool clustering functionality, you can prevent the drop in system availability during errors and during maintenance. The following points describe the operations when an error occurs in the database node and when maintenance is performed for the database node while you are using the connection pool clustering functionality.

- **When an error occurs in the database node**

When a connection cannot be obtained, such as during an OS, hardware, or software error, you can automatically suspend the connection pool connected to the database node where the error occurred (auto-suspension functionality). Even when a connection request is sent from the J2EE application to the resource adapter, no connection request is sent to the suspended connection pool, so the processing is not cancelled until a TCP/IP timeout occurs. This enables the J2EE application to continue business by obtaining a connection from a connection pool connected to another normal database node.

Also, when the database node error is recovered, you can automatically restart the connection pool (auto-restart functionality). If the connection pool is restarted, the automatically recovered database node is accessed again, so you need not execute the `cxclearpool` command to delete the connection pool for recovering the database node.

- **When maintenance is performed for the database node**

When you perform maintenance for a database node, you can use commands to suspend the member connection pool at any time (manual suspension functionality). Due to this, you can separate that database node and perform maintenance.

Also, when you restart the database node after maintenance finishes, you can use commands to restart the connection pool at any time (manual restart functionality).

This section describes how to connect to Oracle clustered by using the Oracle RAC functionality, and the features and functionality of a connection pool when the connection pool is clustered (connection pool clustering).

3.17.1 Connecting to Oracle using Oracle RAC

The method of connecting to Oracle using Oracle RAC differs depending on the Oracle version or the functionality used for load balancing. Note that the connectable transaction type is a local transaction.

For details on the Oracle versions, the functionality used for load balancing, and the mapping of the RAR files used, see *3.6.6 Preconditions and notes on connecting to Oracle*.

This subsection describes the Oracle connection method for each functionality used for load balancing.

(1) Connections using the Application Server functionality

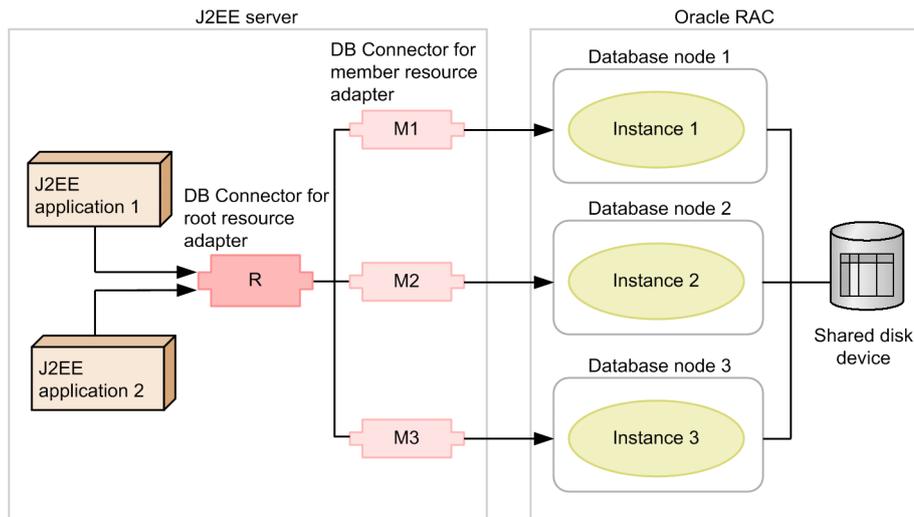
You use the connection pool clustering functionality of Application Server to connect to Oracle RAC. Application Server distributes the load of database access.

For details on the connection pool clustering functionality, see the description in section 3.17.2 and later.

(a) Load balancing procedure and settings

The following figure shows the load balancing procedure and settings when you use the connection pool clustering functionality.

Figure 3-61: Connection using the connection pool clustering functionality



Legend:

→ : Flow of database access

This figure shows an Oracle RAC system with a 3-node configuration where the database node 1 contains instance 1, database node 2 contains instance 2, and database node 3 contains instance 3. The settings for connecting to the database are as follows:

1. Generate DB Connector M1, M2, and M3 for the member resource adapters mapped to instances 1, 2, and 3. Also, generate DB Connector R for the root resource adapter that contains the functionality for distribution to the member resource adapters.
2. Associate the J2EE applications with DB Connector R for the root resource adapter.

With these settings, the database access from the J2EE applications 1 and 2 is distributed to the database nodes 1, 2, and 3.

(b) Operations during database errors and recovery

When a database error occurs, Application Server detects the error. The member resource adapter mapped to the database where the error occurred is blocked and the processing is continued with the remaining instances.

When the database error is recovered, Application Server automatically releases the blockade. You can also release the blockade manually.

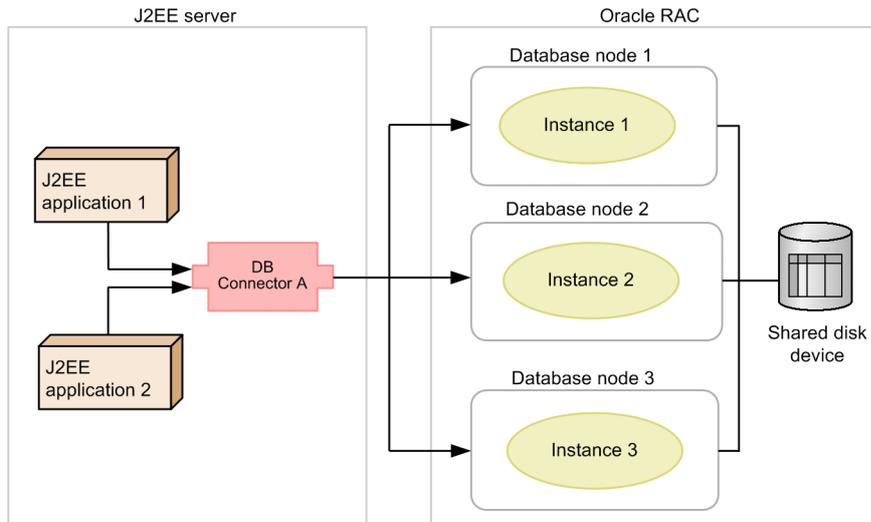
(2) Connections using the Oracle functionality

You connect to Oracle RAC from DB Connector and distribute the database access load with the Oracle RAC functionality.

(a) Load balancing procedure and settings

The following figure shows the load balancing procedure and settings when you use the Oracle RAC functionality.

Figure 3-62: Connection using the Oracle RAC functionality



Legend:

→ : Flow of database access

This figure shows an Oracle RAC system with a 3-node configuration where the database node 1 contains instance 1, database node 2 contains instance 2, and database node 3 contains instance 3. The settings for connecting to the database are as follows:

1. Generate DB Connector A that distributes and connects to each instance.
2. Set up DB Connector A to use a global database name or service name.
3. Associate J2EE application 1 and 2 with DB Connector A.

With these settings, the database access from the J2EE applications 1 and 2 is distributed to the database nodes 1, 2, and 3.

(b) Operations during database errors and recovery

When a database error occurs, the Oracle RAC functionality separates the instance where the error occurred and the processing is continued with the remaining instances.

If you are using an Application Server connection pool, execute one of the following operations during database recovery. The connection pool is cleared and the subsequent access is distributed normally.

- Execute the `cjclearpool` command.
- Restart the J2EE server.

3.17.2 Overview of connection pool clustering

A connection pool that is clustered is called *connection pool clustering*. This subsection describes the connection pool clustering configuration and the functionality available with connection pool clustering.

(1) Connection pool clustering configuration

The member resource adapters connected to each database node and the root resource adapter that bundles these multiple member resource adapters together configure a clustered connection pool. A description of the root resource adapter and member resource adapter is as follows:

- **Root resource adapter**

This resource adapter is accessed from the J2EE applications when a clustered connection pool (connection pool clustering) is used. The root resource adapter bundles the member resource adapters together. With the root

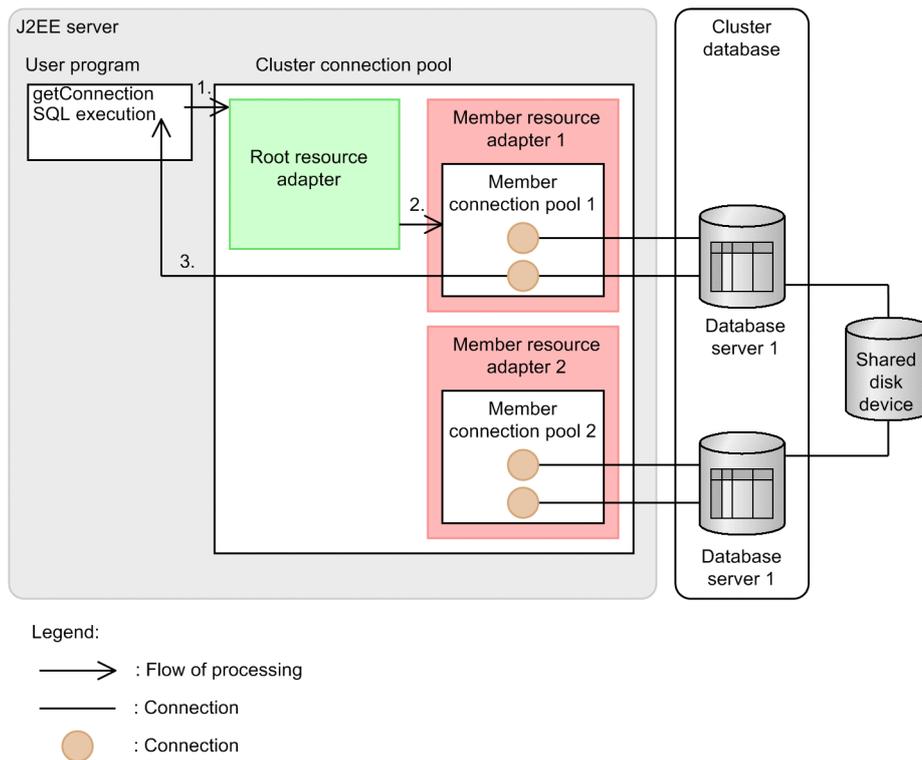
resource adapter, the processing requests to the root resource adapter are distributed to the member resource adapters. Note that the root resource adapter does not have a connection pool.

• **Member resource adapter**

This resource adapter is connected to clustered individual database nodes. A member resource adapter is necessarily accessed via the root resource adapter. The connection pool of a member resource adapter is called a *member connection pool*.

The following figure shows the flow of processing for obtaining a connection when the connection pool is clustered.

Figure 3-63: Flow of processing for obtaining a connection



1. The J2EE application makes a connection request to the root resource adapter.
2. The root resource adapter selects one member connection pool and sends the connection request.
3. A connection is selected from the member connection pool and returned to the J2EE application.

(2) Preconditions

The database that can use the connection pool clustering functionality is Oracle11g only when the RAC functionality is used. The available JDBC driver is Oracle JDBC Thin Driver.

(3) J2EE components and functionality available for database connections

The following table describes the J2EE components and functionality available for database connections when the connection pool clustering functionality is used.

Table 3-73: J2EE components and functionality available for database connections (Connection pool clustering functionality)

Items		Oracle11g (when the connection pool clustering functionality is used)
J2EE components	Servlet/JSP	Y

Items	Oracle11g (when the connection pool clustering functionality is used)	
J2EE components	Stateless Session Bean	Y
	Stateful Session Bean	Y
	Singleton Session Bean	Y
	Entity Bean (BMP)	Y
	Entity Bean (CMP 1.1)	N
	Entity Bean (CMP 2.0)	N
	Message-driven Bean	N
Available functionality	Connection pooling	Y
	Connection pool warming up	Y
	Displaying the connection pool information (cjlistpool command)	Y
	Clearing the connection pool	Y
	Connection test for resources	Y
	Detecting the connection errors	Y
	Statement pooling	Y
	Statement cancellation	Y [#]
	Statement <code>setQueryTimeout</code> method	Y [#]
	PRF trace output for the connection ID	Y
	Output of the SQL statement for troubleshooting	Y

Legend:

Y: Available

N: Not available

[#] Precautions need to be taken when you connect to Oracle. For details, see *3.6.6 Preconditions and notes on connecting to Oracle*.

Important note

To use a resource adapter, you must resolve the references from the J2EE application to the resource adapter. When you customize a J2EE application that uses the resource adapters, resolve the references from the J2EE application to the resource adapter.

(4) Available resource connection and transaction management functionality

For details on the functionality available with the root resource adapter and member resource adapter, see *3.3.4 Resource adapter functionality*.

3.17.3 Resource adapters used

This subsection describes the commands that can be executed with the root resource adapter and member resource adapter, gives an overview of the settings, and describes the notes.

(1) Executable commands

The following table describes the executability of commands for the root resource adapter and member resource adapter. Note that the commands other than those listed in the table can be executed with any resource adapter.

Table 3-74: Executability of commands in root resource adapter and member resource adapter

Command	Types of resource adapters	
	Root resource adapter	Member resource adapter
<code>cjstartrar</code>	Y	Y
<code>cjstoprar</code>	Y	Y
<code>cjtestres</code>	Y	Y
<code>cjlistrar</code>	Y	Y
<code>cjclearpool</code>	N	Y
<code>cjlistpool</code>	N	Y
<code>cjsuspendpool</code>	N	Y
<code>cjresumepool</code>	N	Y

Legend:

Y: Can be executed

N: Cannot be executed

(2) DB Connector settings

The following table describes the items specified in DB Connector for the root resource adapter and member resource adapter.

Table 3-75: Items specified in DB Connector for the root resource adapter and member resource adapter

Settings	Types of resource adapters	
	Root resource adapter	Member resource adapter
Transaction support level	N	Y#1
Can the log be collected	Y	Y
Database connection information	--	Y
DB Connector-specific settings (such as the statement pool)	--	Y
Security information (user name and password)	N	Y#2
Connection pool size	N	Y
Connection pool clustering-specific settings	Y	--

Legend:

Y: Settings must be specified

N: Settings need not be specified

--: There are no settings

#1: The transaction support levels of all the member resource adapters configuring one clustered connection pool must be the same.

#2: The user names of all the member resource adapters configuring one clustered connection pool must be the same.

For details on the DB Connector settings, see 4.2 *Settings for connecting to a database* in the *uCosminexus Application Server Application Setup Guide*, or 4.1 *HITACHI Connector Property file* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

(3) Notes on the root resource adapters

- You cannot execute the `cjclearpool` command. If the command is executed, a message is output and the command terminates abnormally. To clear the connections in a clustered connection pool with the `cjclearpool` command, execute the command for the member resource adapter.
- You cannot execute the `cjlistpool` command. If the command is executed, a message is output and the command terminates abnormally. If you want to display the information of a connection pool within the clustered connection pool, execute the `cjlistpool` command by specifying the member resource adapter display name in `-resname`, or by specifying `-resall`.
- You cannot mix and use a container-managed sign-on and component-managed sign-on in one clustered connection pool. When you use a component-managed sign-on, leave the user names of all the member resource adapters belonging to the root resource adapter blank.
- You can start a root resource adapter when all the member resource adapters belonging to the root resource adapter are running. If you start the root resource adapter in the other cases, an error message is displayed on the console or on the screen, and the resource adapter fails to start.

(4) Notes on the member resource adapters

- With the member resource adapters, the prerequisite functionalities are as follows. These functionalities are enabled by default and cannot be disabled.
 - Connection pooling
Specify a value greater than 0 as the maximum connection pool value. If 0 or less is specified, the operations are performed assuming that the default values are specified for the maximum and minimum connection pool values.
 - Detecting the connection errors when connections are obtained
Detecting the connection errors when connections are obtained and timeout for detecting the connection errors are enabled regardless of the set values.
 - Waiting for a connection when connections deplete
Waiting for a connection when connections deplete is enabled regardless of the set value.
 - loginTimeout
Specify a value greater than 0 in the `loginTimeout` property. If 0 or less is specified, the operations are performed assuming that the default value is specified.
- The following functionalities cannot be used with the member resource adapters. These functionalities are disabled by default and cannot be enabled.
 - Retrying to obtain a connection
 - User-specified Namespace for the J2EE resources
- All the connections in the member connection pool must be connected to the same database node; therefore, do not use the functionality to change the connection destination for a database. The examples of these functionalities are as follows. For the settings on disabling the functionality, see the Oracle documentation.
 - Client load balancing functionality
 - Connection failover functionality
 - Database service
 - Listener-based load balancing functionality
- The member resource adapters are necessarily accessed via the root resource adapter. Therefore, you cannot specify the member resource adapters in the following locations:
 - Resource references of the J2EE applications
 - Mapping definitions of the CMP Entity Bean
- You can stop a member resource adapter when the root resource adapter to which the member resource adapter belongs is stopped. If you stop a member resource adapter when the root resource adapter is running, an error message is displayed on the console or on the screen, and the resource adapter fails to stop.
- When a member connection pool is blocked and suspended, to destroy the connections the unused connections are removed from the connection pool. At this time, if you restart the connection pool when the destruction of

connections is not yet complete, the total number of connections in the connection pool and the unused connections removed from the connection pool might temporarily exceed the maximum number of connections in the connection pool.

3.17.4 Connection pool clustering operations

This subsection describes the available functionality and operations when a connection pool is clustered. You can execute the following functionality with a clustered connection pool:

- Suspending a connection pool
- Restarting a connection pool

This subsection also describes the connection pool states and how to select a connection pool when a connection request is received.

(1) Suspending a connection pool

You can block and suspend a member connection pool. If you execute suspend, a member connection pool is blocked and suspended.

When a J2EE application sends a connection request to the root resource adapter, no connection request is sent to a blocked or suspended member connection pool.

You suspend the member connection pool in the following cases:

- When an error occurs in a database node
- When maintenance is performed for a database node

Note that the suspension methods include the following two methods:

- Auto-suspension
- Manual suspension

Reference note

With the suspension processing, the connections that the J2EE application has finished using are destroyed, all the connections in the connection pool are destroyed, and then the connection pool is suspended. When a network error or a database node error occurs, a connection is destroyed after waiting for the network to timeout, so the processing from blocking to suspending the connection pool might take time.

(a) Auto-suspension

You can automatically suspend the member connection pool when a database node error occurs. When an error is detected, the member connection pool is automatically blocked and then suspended.

If the following events occur when a connection is obtained, a database node error is determined, and the member connection pool is automatically suspended:

- When a timeout occurs during the detection of the connection errors while a connection is being obtained
- When a physical connection cannot be obtained
- When the obtaining of a physical connection times out
- When the connection management threads deplete

This functionality is enabled by default. Specify the settings for enabling or disabling this functionality as a root resource adapter property. For details on the resource adapter settings, see *5.4 Resource adapter property definitions* in the *uCosminexus Application Server Application Setup Guide*.

(b) Manual suspension

When you perform operations such as database node maintenance, you execute the `cjsuspendpool` command to manually suspend a connection pool. You can execute manual suspension when the status of a member connection pool is Start, Reserved Start, Auto-suspend, and Reserved Auto-suspend. If you execute the `cjsuspendpool`

command when the member connection pool is in an auto-suspension state, the status changes to manual suspension. Due to this, you can execute operations so that the member connection pool is not restarted automatically after auto-suspension. For details on the manual suspension procedure, see 3.17.5(2) *Suspending a connection pool*. For command details, see *cjsuspendpool (Suspending a member connection pool)* in the *uCosminexus Application Server Command Reference Guide*.

! Important note

A connection pool that is suspended manually with the `cjsuspendpool` command is not restarted automatically. Restart the connection pool manually.

(2) Restarting a connection pool

You can restart a suspended member connection pool. When the J2EE application makes a connection request to the root resource adapter, the connection requests are now sent to the restarted member connection pool again.

Note that to restart a connection pool, the number of unused connection management threads must be equal to or more than the maximum number of connections in the connection pool.

You restart the connection pool in the following cases:

- When the database node error is recovered
- When the maintenance of the database node finishes

The restart methods include the following two methods:

- Auto-restart
- Manual restart

Reference note

- If the number of unused connection management threads becomes equal to the maximum number of connections in the connection pool when the connection pool is stopped, a message is output. When the execution of the `cjresumepool` command fails, check the displayed message and then re-execute the command.
 - When the connection pool warming up functionality is enabled, the member connection pool starts after the minimum number of connections defined in the connection pool settings is pooled. If an attempt to generate a connection fails while the connections are being pooled, the member connection pool returns to the suspended state. Also, if the connection pool warming up functionality is disabled, the member connection pool starts immediately.
-

(a) Auto-restart

You can automatically restart an auto-suspended member connection pool.

The auto-suspended member connection pool sends a physical connection request at regular intervals to check the state of the database node. At this time, if a connection is obtained successfully, the database node is determined to have recovered, and the restart processing is performed automatically. Also, if you do not want to automatically restart an auto-suspended member connection pool, manually suspend the member connection pool after the pool is suspended automatically. To restart the pool, use manual restart.

Note that if the root resource adapter is stopped, the auto-restart processing is not performed. However, if you stop the root resource adapter while the member connection pool is auto-restarting, the running auto-restart processing is continued.

This functionality is enabled by default. Specify the settings for switching between enabling and disabling and for the interval to check the database node state as the root resource adapter properties. For details on the resource adapter settings, see 5.4 *Resource adapter property definitions* in the *uCosminexus Application Server Application Setup Guide*.

(b) Manual restart

You can manually restart an auto-suspended or manually suspended connection pool. To restart manually, use the `cjresumepool` command. You can execute manual restart when the status of the connection pool is as follows:

- Auto-suspension
- Manual suspension

- Reserved auto-suspension
- Reserved manual suspension

For details on the manual restart procedure, see 3.17.5(3) *Restarting a connection pool*. For the command details, see *cjresumepool (Restarting a member connection pool)* in the *uCosminexus Application Server Command Reference Guide*.

(3) States of the connection pool

A connection pool state only exists for the member connection pools. Note that the connection pool state is maintained even after you restart the J2EE server and resource adapter.

You can check the state of the member connection pool with the following methods. When you execute manual suspension or manual restart, check the connection pool state before you execute the commands.

- `cjlistrar` command
The `cjlistrar` command outputs the names and states of all the deployed resource adapters to the standard output. If you specify `-clusterpool`, you can also display the states of the member connection pools.
For the command details, see *cjlistrar (Displaying a list of resource adapters)* in the *uCosminexus Application Server Command Reference Guide*.
- `cjlistpool` command
The `cjlistpool` command displays the connection pool information. The command can also display the states of the member connection pools for the member resource adapters. For the examples on displaying the connection information for the member resource adapters, see 3.15.4 *Displaying the connection pool information*.
For the command details, see *cjlistpool (Displaying the list of connection pools)* in the *uCosminexus Application Server Command Reference Guide*.

This subsection describes the states of the member connection pools.

The state of the member connection pool is maintained even if you restart the J2EE server and member resource adapter. For the procedure for checking the connection pool states, see 3.17.5(1) *Checking the connection pool states*.

Note that the connection pools other than those for the member resource adapters do not have a state.

(a) Transition of the connection pool state

The following figure shows the transition of the member connection pool state.

Figure 3-64: Transition of the member connection pool state (when the pool is suspended manually)

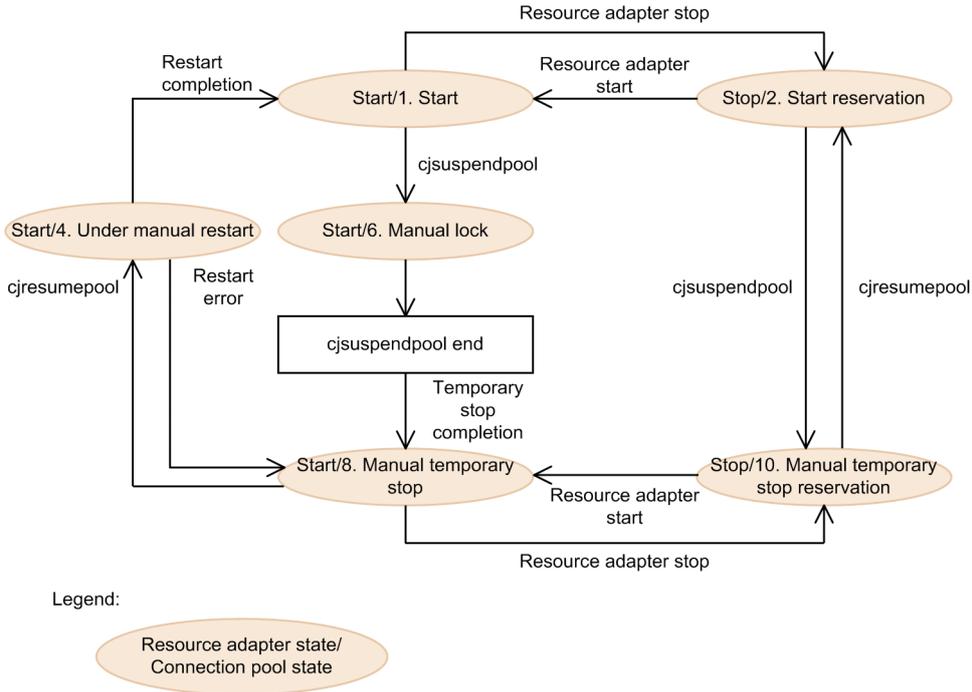
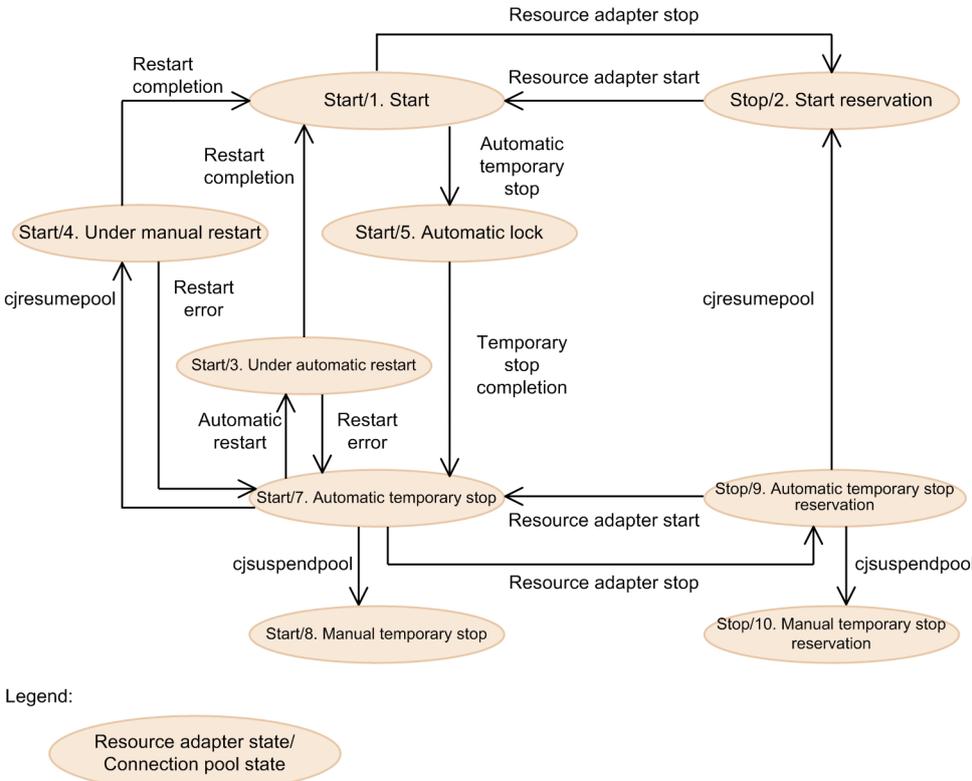


Figure 3-65: Transition of the member connection pool state (when the pool is suspended automatically)



Note that when the J2EE server starts, if the status of the connection pool is Resuming or Blocked, the connection pool status transits to Suspended.

The following table describes each status.

Table 3-76: Status of the member connection pools

No.	Status	Explanation
1	Start status	A state in which the connection pool receives processing. When a connection request is made to the root resource adapter, the processing is only performed for a started connection pool.
2	Reserved Start status	A state in which the resource adapter is stopped when the connection pool has the Start status. The status of a connection pool with the Reserved Start status changes to the Start status when you start the resource adapter. This status is reached immediately after the resource adapter is deployed.
3	Auto-restarting status	A state in which the connection pool performs the restart processing using the auto-restart functionality. If the restart processing is successful, the status changes to the Start status. If the restart processing fails, the status returns to the Auto-suspension status. No request is sent to a connection pool with the Auto-restarting status when a connection request is sent to the root resource adapter.
4	Manual Restarting status	A state in which the connection pool performs the restart processing by executing the <code>cjresumepool</code> command in the Auto-suspension status or Manual Suspension status. If the restart processing is successful, the status changes to the Start status. If the restart processing fails, the status returns to the one before the command was executed. No request is sent to a connection pool with the Manual Restarting status when a connection request is sent to the root resource adapter.
5	Automatically Blocked status	A state in which the auto-suspension functionality is used to block a connection pool and perform the suspension processing. When the suspension processing finishes, the status changes to the Auto-suspension status. No request is sent to a connection pool with the Automatically Blocked status when a connection request is sent to the root resource adapter.
6	Manually Blocked status	A state in which the <code>cjsuspendpool</code> command is used to block a connection pool and perform the suspension processing. When the suspension processing finishes, the status changes to the Manual Suspension status. No request is sent to a connection pool with the Manually Blocked status when a connection request is sent to the root resource adapter.
7	Auto-suspension status	A state in which the auto-suspension functionality is used to suspend the connection pool. There are no connections in the connection pool. No request is sent to a connection pool with the Auto-suspension status when a connection request is sent to the root resource adapter.
8	Manual Suspension status	A state in which the <code>cjsuspendpool</code> command is used to suspend the connection pool. There are no connections in the connection pool. No request is sent to a connection pool with the Manual Suspension status when a connection request is sent to the root resource adapter.
9	Reserved Auto-suspension status	A state in which the resource adapter is stopped when the connection pool is in the Auto-suspension status. A connection pool with the Reserved Auto-suspension status changes to the Auto-suspension status when the resource adapter is started. Note that at this time the connection pool warming up functionality is disabled.
10	Reserved Manual Suspension status	A state in which the resource adapter is stopped when the connection pool is in the Manual Suspension status. A connection pool with the Reserved Manual Suspension status changes to the Manual Suspension status when the resource adapter is started. Note that at this time the connection pool warming up functionality is disabled.

Note: The numbers indicate the numbers in Figure 3-64 and Figure 3-65.

(b) Executability of commands based on the connection pool status

Depending on the state of the connection pool, some commands can be executed and some cannot be executed. The following table describes whether the commands can be executed for each connection pool state.

Table 3-77: Executability of commands based on the connection pool status

Command	Connection pool status							
	Start	Reserve d Start	Auto- restarting / Manual Restartin g	Automati cally Blocked/ Manually Blocked	Auto- suspensi on	Manual Suspensi on	Reserve d Auto- suspensi on	Reserve d Manual Suspensi on
cjstartrar	N	Y	N	N	N	N	Y	Y
cjstoprar	Y	N	R#1	R#2	Y	Y	N	N
cjclearpool	Y	N	N	N	N	N	N	N
cjlistrar	Y	Y	Y	Y	Y	Y	Y	Y
cjsuspendpool	Y	Y	N	N	Y	N	Y	N
cjresumepool	N	N	N	N	Y	Y	Y	Y
cjstopsv (for a normal termination)	Y	Y	R#1	R#2	Y	Y	Y	Y
cjstopsv (for a forced termination)	Y	Y	Y	Y	Y	Y	Y	Y

Legend:

- Y: Can be executed
- R: Restricted
- N: Cannot be executed

#1: The command is received, but the processing is executed when the status changes to Start or Suspend.

#2: The command is received, but the processing is executed when the status changes to Suspend.

(4) How to select a connection pool

When the J2EE application sends a connection request to the root resource adapter, one member connection pool is selected. The method by which the member connection pool is selected at this time is the round-robin method.

The connection pool that is selected is a member connection pool with the Start status. A member connection pool that has free connections is selected on priority.

However, if only a member connection pool with depleted connections is available, the connection request changes to pending. Furthermore, if a timeout occurs in a pending connection, the connection cannot be obtained. Also, if a connection pool is blocked when a connection request is pending, the connection request is restarted and an attempt is made to obtain a connection from the member connection pool that is next in the priority order. If a connection cannot be obtained from any of the member connection pools, the attempt to obtain the connection fails.

Note that if there is a connection request when a member connection pool with the Start status does not exist, the attempt to obtain the connection fails.

To specify the maximum size of each member connection pool, use the following guideline:

Maximum-size-of-the-member-connection-pool-(number) = Maximum-number-of-concurrent-connections-permitted-for-the-system / number-of-database-nodes

3.17.5 Procedure for stopping or starting a connection pool manually

This subsection describes the procedure for connection pool clustering. By manually stopping and restarting some of the connection pools when you handle the errors occurring in a database and when you perform database maintenance, you can perform database maintenance without stopping the entire system. To perform database maintenance by manually stopping and restarting some of the connection pools:

1. Check the connection pool status (see 3.17.5(1))
Use the server management commands to execute the operation.

2. Suspend the connection pool (see 3.17.5(2))
Use the server management commands to execute the operation.
3. Restart the connection pool (see 3.17.5(3))
Use the server management commands to execute the operation.
4. Check the connection pool status (see 3.17.5(1))
Use the server management commands to execute the operation.

(1) Checking the connection pool state

You check the connection pool state before and after you perform maintenance for a database in a cluster configuration.

For details on the connection pool states, see 3.17.4(3) *States of the connection pool*.

You check the connection pool state with the `cjlistrar` command.

The format and example of execution of the `cjlistrar` command are as follows:

Format of execution

```
cjlistrar server-name -clusterpool
```

Example of execution

```
cjlistrar MyServer -clusterpool
```

(2) Suspending the connection pool

This section describes the procedure for suspending the connection pool.

You can suspend a connection pool with the `cjsuspendpool` command. When you execute the `cjsuspendpool` command, the connection pool is blocked and suspended and the connection requests are no longer received.

You can execute the `cjsuspendpool` command when the status of the connection pool is as follows:

- Start
- Reserved Start
- Auto-suspension
- Reserved Auto-suspension

For details on how to check the state of the connection pool, see (1) *Checking the connection pool state*.

The format and example of execution of the `cjsuspendpool` command are as follows:

Format of execution

```
cjsuspendpool server-name -resname display-name-of-member-resource-adapter-to-be-suspended
```

Example of execution

```
cjsuspendpool MyServer -resname DB_Connector_for_Oracle_ClusterPool_Member
```

(3) Restarting the connection pool

This section describes the procedure for restarting the connection pool.

You can restart the connection pool with the `cjresumepool` command. When you execute the `cjresumepool` command, the connection pool status changes to Manual Restarting and the restart processing is executed. When the restart processing finishes, the connection pool status changes to Start and the connection requests are received.

You can execute the `cjresumepool` command when the state of the connection pool is as follows:

- The number of unused connection management threads is equal to the maximum number of connections in the connection pool or more
- The status of the connection pool is Manual Suspension, Reserved Manual Suspension, Auto-suspension, or Reserved Auto-suspension

For details on how to check the state of the connection pool, see (1) *Checking the connection pool state*.

The format and example of execution of the `cjresumepool` command are as follows:

Format of execution

```
cjresumepool server-name -resname display-name-of-member-resource-adapter-to-be-restarted
```

Example of execution

```
cjresumepool MyServer -resname DB_Connector_for_Oracle_ClusterPool_Member
```

Note that after you execute the `cjresumepool` command, you must check the status of the connection pool to check whether the restart processing was executed correctly. For details on how to check the state of the connection pool, see (1) *Checking the connection pool state*.

3.17.6 Settings required for clustering a connection pool

To cluster a connection pool, you must set up the DB Connector properties. For details on setting up the DB Connector properties, see 4.3 *Settings for connecting to the database (for connection pool clustering)* in the *uCosminexus Application Server Application Setup Guide*.

3.18 Connection test for resources

After you set up the resource adapter properties, you can verify whether the set up content is correct. This is called the *connection test functionality*. The content verified by a connection test differs for each connected resource. The following table describes the content to be verified in the connection test for each resource.

For details on the procedure for executing the connection test and resource adapter settings, see 3.3.8 *Procedure for resource adapter settings (To deploy and use the resource adapter as a J2EE resource adapter)* and 3.3.9 *Procedure for resource adapter settings (To include and use the resource adapter in a J2EE application)*.

Table 3-78: Content to be verified in the connection test

Resource type	Connection method	Content to be verified in the connection test
Database	DB Connector	<ul style="list-style-type: none"> The mode is the 1.4 mode The specified transaction support level is available.^{#1, #2} The DB Connector settings are correct. The settings for the JDBC driver to be used are correct. A connection can be established with the database. The SQL statement was issued successfully.
	DB Connector (Root resource adapter when the connection pool clustering functionality is used)	<ul style="list-style-type: none"> The mode is the 1.4 mode. The specified transaction support level is available.^{#1} The DB Connector settings are correct. The settings required for starting the root resource adapter are specified.^{#3}
	DB Connector (Member resource adapter when the connection pool clustering functionality is used)	<ul style="list-style-type: none"> The mode is the 1.4 mode. The specified transaction support level is available.^{#1} The DB Connector settings are correct. The Oracle JDBC Thin Driver settings are correct. A connection can be established with the database. The SQL statement was issued successfully. The prerequisite functionality for the member resource adapter is being used.^{#4} The functionality that is not available with the member resource adapters is not used.^{#4}
Database queue	Cosminexus RM	<ul style="list-style-type: none"> The mode is the 1.4 mode. The specified transaction support level is available.^{#1} The settings for Cosminexus RM are correct. DB Connector for Cosminexus RM to be integrated is running. The settings for DB Connector for Cosminexus RM to be integrated are correct.

3. Resource Connections and Transaction Management

Resource type	Connection method	Content to be verified in the connection test
Database queue	Cosminexus RM	<ul style="list-style-type: none"> A connection can be established with the database to be used.
	DB Connector for Cosminexus RM	<ul style="list-style-type: none"> The settings for DB Connector for Cosminexus RM are correct. DB Connector for Cosminexus RM is running. The settings for Cosminexus RM to be integrated are correct. Cosminexus RM to be integrated is running. The settings for the JDBC driver to be used are correct. A connection can be established with the database to be used. The SQL statement was issued successfully.
OpenTP1	TP1/Message Queue - Access	<ul style="list-style-type: none"> The mode is the 1.4 mode. The specified transaction support level is available.^{#1} The settings for TP1/Message Queue - Access are correct. TP1/Message Queue - Access can establish a connection to TP1/Message Queue.
	uCosminexus TP1 Connector	<ul style="list-style-type: none"> The mode is the 1.4 mode. The specified transaction support level is available.^{#1} The settings for uCosminexus TP1 Connector are correct.
	TP1 inbound adapter	<ul style="list-style-type: none"> A connection test cannot be performed because of the communication with Inbound. If a connection test is attempted, the KDJE48606-E message is output.
CJMSP Broker	CJMSP resource adapter	<ul style="list-style-type: none"> The mode is the 1.4 mode. A connection can be established with CJMSP Broker.
SMTP server	Mail configuration	<ul style="list-style-type: none"> A connection can be established with the SMTP server.
Other	Resource adapters from other companies	<ul style="list-style-type: none"> The mode is the 1.4 mode. A connection can be established with the resource ^{#5} (if the <code>createManagedConnection</code> method of the implementation class of <code>javax.resource.spi.ManagedConnectionFactory</code> is invoked successfully, the connection with the resource is assumed to have been established). The resource adapters conforming to Connector 1.5 can be started and stopped when the resource adapter is not running (if the <code>start</code> method and <code>stop</code> method of the implementation class of

Resource type	Connection method	Content to be verified in the connection test
Other	Resource adapters from other companies	<code>javax.resource.spi.ResourceAdapter</code> are invoked successfully, the resource adapter is assumed to have been started or stopped).

#1: Available if a global transaction (`XATransaction`) is specified as the transaction support level for a resource adapter and the `ejbserver.distributedtx.XATransaction.enabled` key in `usrconf.properties` is `true`. For details on the transaction support levels, see [3.4.3 Transaction types available for each resource](#).

#2: You cannot use a global transaction (`XATransaction`) when the database to be connected to is XDM/RD E2. The physical connection for transaction recovery cannot be obtained, so DB Connector cannot be started.

#3: For details on the settings required for starting the root resource adapter, see [4.3.3 DB Connector settings for the root resource adapter](#) in the *uCosminexus Application Server Application Setup Guide*.

#4: For details on the prerequisite functionality and the functionality that is not available with member resource adapters, see [3.17.3 Resource adapters used](#).

#5: If the connection definition required for connecting to a resource does not exist when you perform Outbound communication, the message KDJE48606-E is output and the test fails.

3.19 Functionality for operations in the firewall environment

When you embed the firewall in a system, you must fix the port to be used for communication, and permit communication using the fixed port only.

The following table describes the organization of this section.

Table 3-79: Organization of this section (Functionality for operations in the firewall environment)

Category	Title	reference location
Explanation	Communication port for transaction recovery	3.19.1
	Communication port used by Smart Agent	3.19.2
Settings	Settings for operations in the firewall environment	3.19.3

Note: The functionality-specific explanation is not available for "Implementation" and "Operations".

Reference note

Preventing the deterioration of communication performance using connection sweeper

A firewall has a functionality to forcefully disconnect a session that has not communicated for a fixed period of time or more. A firewall is allocated between Application Server and the database server, and when the connection pool is enabled, the connections that are not used for a while might be disconnected by the firewall. There is a long waiting period for the subsequent use of the disconnected connection. You can prevent this event by specifying settings in the connection sweeper functionality to destroy the unused connections faster than the time taken by the firewall to disconnect a non-communicating session.

3.19.1 Communication port for transaction recovery

When you use a global transaction, you use the communication port for the transaction recovery processing. When you embed a firewall in the system, you must permit communication from this port depending on the firewall configuration. By default, 20302 is used as the port number, but you can change this number.

You specify the settings of the communication port for transaction recovery by customizing the J2EE server properties. For details on customizing the settings for the J2EE server operations, see *3.19.3 Settings for operations in the firewall environment*.

3.19.2 Communication port used by Smart Agent

The TPBroker Smart Agent (OSAgent) uses a communication port. When you embed a firewall in the system, you must permit communication using this port depending on the firewall configuration.

You specify the settings for the communication port used by Smart Agent by customizing the J2EE server properties. For details on customizing the settings for the J2EE server operations, see *3.19.3 Settings for operations in the firewall environment*.

Note that the communication port specified in the J2EE server properties must also be specified in the environment variable `OSAGENT_PORT`. Make sure you specify the same value in the J2EE server properties and the environment variable `OSAGENT_PORT`.

3.19.3 Settings for operations in the firewall environment

To perform operations in the firewall environment, you must specify the J2EE server settings. You specify the items listed in the following table in the `<configuration>` tag of the logical J2EE server (`j2ee-server`) in the Easy Setup definition file.

Table 3-80: Settings for operations in the firewall environment in the Easy Setup definition file

Items	Specified parameters	Settings
Transaction recovery	<code>ejbserver.distributedtx.recovery.port</code>	Specifies the fixed port number to be used for transaction recovery.
Communication port used by Smart Agent	<code>vbroker.agent.port</code>	Specifies the communication port used by Smart Agent.

3.20 Notes on starting a transaction with the EJB client applications

This section describes the notes on starting transactions with the EJB client applications.

Tip

You can start a transaction with an EJB client application when Application Server is installed on the EJB client computer. You cannot start a transaction with an EJB client application when uCosminexus Client is installed on the EJB client computer.

When you start a transaction with an EJB client application, you use a global transaction to invoke the EJBs on Application Server. At this time, the transaction manager and transaction service propagate the transaction between the EJB client and Application Server and finally execute a two-phase commit.

A transaction started using an EJB client application can invoke multiple EJBs existing on Application Server. Multiple resources can be accessed with Application Server. Note that you cannot access the resources directly from an EJB client application.

This section describes the notes on starting a transaction with the EJB client applications.

3.20.1 Notes on application development

This subsection describes the notes on application development when you start a transaction with an EJB client application. For details on implementing transactions with the EJB client applications, see *3.5 Implementing transactions with the EJB client applications* in the *uCosminexus Application Server EJB Container Functionality Guide*.

- To use a global transaction with an EJB client application, immediately after starting the EJB client application, you must invoke the processing for initializing the services to be used with the EJB client application, from the user program. By invoking this initialization processing, the transaction manager and transaction service are initialized. As the initialization processing, invoke the `initialize()` method of the `EJBClientInitializer` class.
- By invoking service initialization from the user program even when the EJB client application terminates abnormally during the transaction processing, you can start the transaction recovery. Note that the recovery processing is executed in the background, so the `initialize()` method is returned without waiting for the recovery processing to finish. Design the EJB client application so that the recovery of a global transaction starts by restarting the EJB client application.
- If transactions are started in an EJB client application, specify the design so that all the transactions are concluded and then the EJB client application stops. If the EJB client application stops without waiting for the transactions to conclude, the uncompleted transactions might remain in the transaction. In this state, you might not be able to terminate Application Server normally, stop the resource adapter, or release the resource lock.

3.20.2 Notes on system setup

This subsection describes the notes on system setup and the settings.

(1) Notes on system setup

This section describes the notes on system setup when you start a transaction with an EJB client application. For details on the settings of the EJB client application transactions, see *(2) Settings for using a transaction with an EJB client application*.

- When you start a transaction with an EJB client application, to use a global transaction, the light transaction functionality must be disabled on Application Server (enabled by default).
- If the EJB to be invoked is specified in the attributes such as the `Mandatory` attribute, `Required` attribute, or `Supports` attribute in an EJB container-managed transaction (CMT), the EJB is executed within the scope of the transaction started with the EJB client application.

(2) Settings for using a transaction with an EJB client application

This section describes the settings for using a transaction with an EJB client application.

! Important note

When you use uCosminexus Client to set up the EJB client environment, the EJB client application transactions are not available.

To use transactions with the EJB client applications, you must specify the following settings:

- JAR file settings
- Property settings
- Settings for obtaining `UserTransaction`

The settings for obtaining `UserTransaction` are the settings used for developing the J2EE applications. To use transactions with the EJB client applications, you must specify settings to obtain `UserTransaction` (`javax.transaction.UserTransaction`) from the EJB client application. For details on how to specify the settings, see *3.5 Implementing transactions with the EJB client applications* in the *uCosminexus Application Server EJB Container Functionality Guide*.

This subsection describes the JAR file and property settings for using transactions with the EJB client applications.

(3) JAR file settings

To use transactions with the EJB client applications, specify the following JAR files in the class path:

In Windows

- `Cosminexus-installation-directory\TPB\lib\tpotsinproc.jar`
- `Cosminexus-installation-directory\CC\lib\ejbserver.jar#`

In UNIX

- `/opt/Cosminexus/TPB/lib/tpotsinproc.jar`
- `/opt/Cosminexus/CC/lib/ejbserver.jar#`

[#]: In the class path settings, specify `ejbserver.jar` after `HiEJBClientStatic.jar`.

For details on specifying the JAR file in the class path, see *3.7.4 Specifying the JAR file in the class path of the EJB client application* in the *uCosminexus Application Server EJB Container Functionality Guide*.

(4) Property settings

This section describes the keys that must be set up to use transactions with the EJB client applications. For details on the property settings, see *3.3.5 Setting up the EJB client application properties* in the *uCosminexus Application Server EJB Container Functionality Guide*. For details on the keys, see *14.3 usrconf.properties (User property file for Java applications)* in the *uCosminexus Application Server Definition Reference Guide*.

To use transactions with the EJB client applications, you set up the following keys:

(a) Mandatory properties

- `ejbserver.client.transaction.enabled`
Specify `true` to enable the use of transactions with the EJB client applications.
- `ejbserver.distributedtx.recovery.port`
Specifies the fixed port number to be used for transaction recovery when a global transaction is used. Specify a different port number for each EJB client process. Also, specify a port number different from the port number for the recovery of Application Server operating on the same computer.
- `ejbserver.client.transaction.clientName`

Specifies the client name to be used by the transaction service. Specify a different name for each EJB client process in this property. Also, specify a name different from the name of the J2EE server operating on the same computer.

- `ejbserver.distributedtx.ots.status.directory1`

Specifies the directory for allocating the status file to be used by the transaction service and the backup of the status file. Specify a different directory for each EJB client process. Also, specify a directory different from the status file directory specified for the J2EE server.

(b) Optional properties

- `ejbserver.jta.TransactionManager.defaultTimeout`

Specifies the time until a transaction timeout occurs.

- `ejbserver.distributedtx.ots.status.directory2`

Specify the second directory for allocating the status file to be used by the transaction service and the backup of the status file. Specify a different directory for each EJB client process. Also, specify a directory different from the status file directory specified for the J2EE server.

(5) Notes

This section describes the notes on specifying the settings for the EJB client application transactions.

To use a global transaction, you must disable the light transaction functionality on Application Server. By default, the light transaction functionality is enabled. If you specify `true` in the `ejbserver.distributedtx.XATransaction.enabled` key of `usrconf.properties` for the J2EE servers, the light transaction functionality is disabled and the global transaction can be used.

3.20.3 Notes on system operations

This subsection describes the notes on operations when transactions are started with the EJB client applications.

- After you stop the EJB client application, if you cannot stop Application Server and resource adapter, the uncompleted transactions might remain in the transaction. In this case, restart the EJB client application and recover the global transaction.

Note that the recovery processing is executed by invoking service initialization in the EJB client application after the application is restarted.

- If the EJB client computer is down due to reasons such as errors during transaction processing, you must restart the EJB client application and then recover the global transaction.
- The transactions started with the EJB client applications cannot be forcefully concluded by using the method cancellation functionality.
- If an exception occurs during service initialization, the items such as the system properties for executing the EJB client applications might be specified incorrectly. Take action according to the exception message.
- When transactions are started with the EJB client applications, the root application information and client application information is not included in the trace based performance analysis output by the JTA and OTS. To trace a request, use the hash code and the XID information of the thread.

Also, the message output when a transaction timeout occurs includes the hash code of the thread that started the transaction, instead of the root application information. You can also use this information to trace a request.

4

Invoking Application Server from OpenTP1 (TP1 Inbound Integrated Function) (INTENTIONALLY DELETED)

INTENTIONALLY DELETED

4.1 INTENTIONALLY DELETED

INTENTIONALLY DELETED

5

How to Use JPA with Application Server

This chapter gives an overview of the JPA and describes how to use JPA with Application Server.

5.1 Organization of this chapter

The JPA is a specification aimed at simplifying the designing and coding of the database-related processing and is used for mapping Java objects and relational database (O/R mapping). By using the JPA, you can operate the database information as a Java object (entity), and set up a system efficiently.

This chapter gives an overview of JPA and describes the JPA usage with Application Server. The following table describes the organization of this chapter.

Table 5-1: Organization of this chapter (Using JPA with Application Server)

Category	Title	Reference location
Explanation	Features of JPA	5.2
	JPA functionality that can be used with Application Server	5.3
	EntityManager	5.4
	Persistence context	5.5
Implementation	How to obtain a container-managed EntityManager	5.6
	How to obtain an application-managed EntityManager	5.7
	Definitions in persistence.xml	5.8
	Allocating persistence.xml	5.9
	JPA interfaces	5.10
Notes	Notes on setting up applications	5.11

Note: The functionality-specific explanation is not available for *Settings* and *Operations*.

5.2 Features of JPA

This section describes features of the applications using JPA.

5.2.1 Advantages of applications using JPA

The following contents can be realized, if you use an application using JPA:

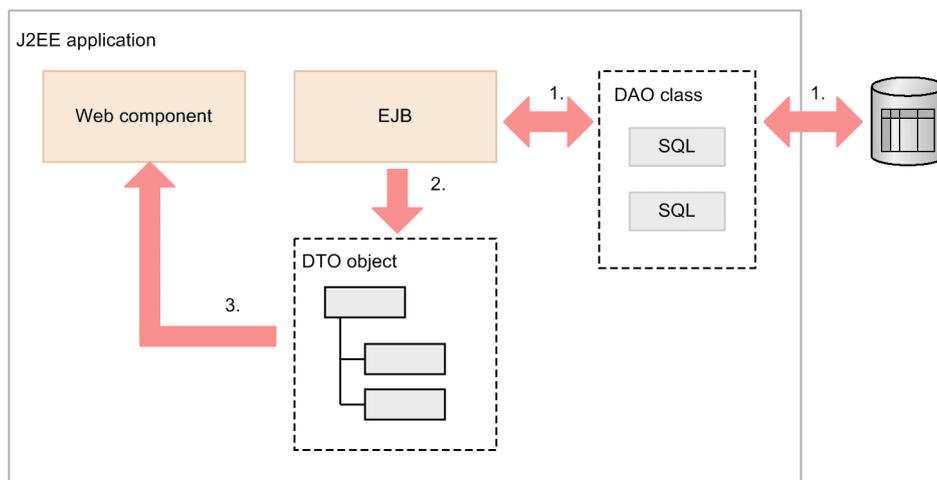
- By concealing the O/R mapping and database access processing, the user programming becomes easy.
- By using annotations, the cost of creating the definition file is reduced and coding is implemented using POJO (Plain Old Java Object).
- By setting up the default values, the amount of coding by the user can be reduced.
- Cosminexus JPA provider conforms to the JPA specifications, and therefore, you can merge the applications used with other JPA providers.

This subsection compares the data access models when the JPA is not used and when the JPA is used, and describe the advantages of applications using the JPA.

(1) Data access model when the JPA is not used

To access a database from a J2EE application when the JPA is not used, you can use the data access model shown in the following figure to create an application.

Figure 5-1: Database access model when JPA is not used



This subsection describes the above figure.

With the data access model shown in the figure, you create a class called *DAO* corresponding to the table to hide the SQL statement from the business logic. With the DAO class, you create a process to issue an SQL statement using the JDBC interface. The flow of processing shown in the figure is as follows:

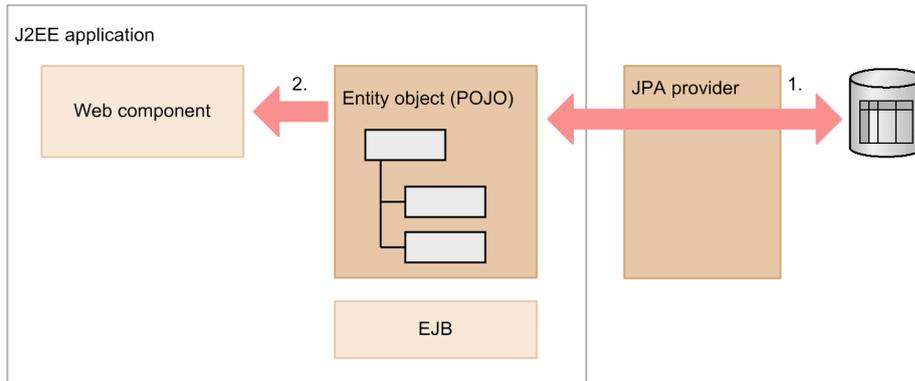
1. With an EJB where the business logic is coded, DAO is used to read the data from a database.
2. The acquired data is stored in an object called *DTO*.
3. The DTO object returns to the Web component. With the Web component, the data acquired from the database is output to a Web page.

With such a data access model, if the data model of the database becomes more complex, the number of DAO, SQL, and DTO classes that must be created also increases. The creation of DAO, SQL, and DTO involves monotonous manual work, so this decreases the productivity of an application development.

(2) Data access model when the JPA is used

The following figure shows a data access model when the JPA is used.

Figure 5-2: Database access model when the JPA is used



When the JPA is used, you create a class corresponding to the lines in the database table. This class is called an *entity class*. With an EJB where the business logic is coded, you can code the processing as if the object of this entity class is directly stored in the database.

The description of the figure is as follows:

1. The JPA engine called the JPA provider issues an SQL statement for the database. Also, the JPA provider automatically synchronizes the status of the entity object and the database table.
2. The entity object can pass the obtained data to the Web component as is.

If you use the JPA, you need not create DTO. Furthermore, the entity class can also be automatically generated from the database table schema by using a development tool such as Eclipse. Because you do not need to create the DAO, SQL, and DTO classes that were the reason of decreased productivity in the past data access models, you can further improve the productivity of the applications.

For details about the entity classes and the JPA providers, see [5.2.2 Entity class](#) and [5.2.3 JPA provider](#).

5.2.2 Entity class

When you use the JPA, you create a class that forms a data container with applications. This class is called the *entity class*. Normally, you create the entity class so that one object of the entity class corresponds to one line in the database table. You create an entity class using the normal Java class (POJO). Special interfaces need not be implemented.

You specify the mapping for the values of the entity class fields and the columns of the database table where the values are stored, using annotations in the items such as the entity class fields. However, the CoC concept has been introduced in JPA in order to improve the easy development. If you do not specify mapping explicitly, the default mapping rules are applied. For example, if the mapping of fields is not explicitly specified, the corresponding column is presumed from the field name and mapped.

5.2.3 JPA provider

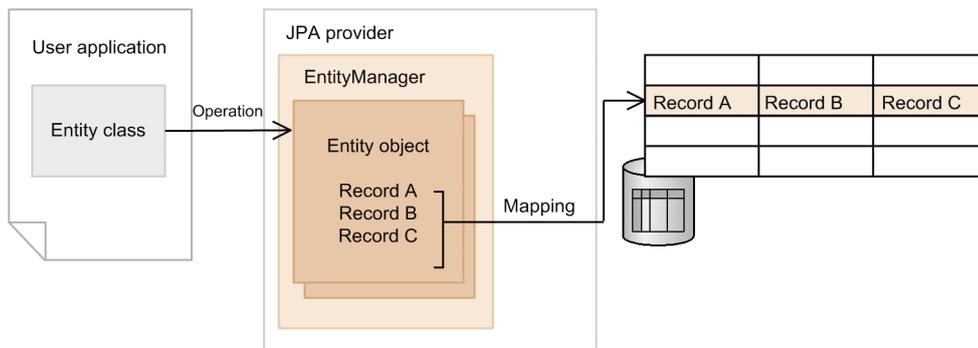
The *JPA provider* is a JPA implementation that provides the following mapping functionality, API, and query language. The following functionality is provided with the JPA provider:

- Functionality for mapping Java objects and a database
- API encapsulating the database processing
- Query language that can be commonly used in the JPA specifications

The advantage of using the JPA provider is that you can design an application without knowing the processing related to database exchanges. Also, by using the query language JPQL available with the JPA provider, you can also send a query even if you are unfamiliar with the database.

The following figure shows the mapping functionality provided by JPA providers.

Figure 5-3: Overview of the mapping functionality provided by JPA providers



This subsection describes the above figure.

An entity class is prepared in the application and an entity object is generated from the entity class. By changing the contents of the entity object, the user changes the database contents. As a result, the user can update the database contents without knowing the database processing.

The JPA provider maps the entity objects to the database records. The JPA provider also executes the search, insert, delete, or update processing implemented by the user for the database.

The generated entity object is managed by EntityManager. If the value of an entity object field is changed, EntityManager automatically detects the change and applies the change to the database table.

During the following processing, EntityManager is invoked:

- To add the data of the entity class object in the database table.
- To search the data already stored in the database and to extract the data as an entity class object.

For details on EntityManager, see *5.4 EntityManager*.

5.3 JPA functionality that can be used with Application Server

This section describes the JPA providers, components, and resource adapters that can be used with Application Server, when the JPA is used.

5.3.1 Available JPA providers

The *JPA provider* is an engine that provides the EntityManager functionality. The JPA providers that can be used with Application Server include Cosminexus JPA provider and the JPA providers provided by other vendors. The following points describe the usage of each of these JPA providers:

(1) When Cosminexus JPA provider is used

Cosminexus JPA provider is a JPA provider provided with Application Server. Cosminexus JPA provider provides the Cosminexus JPA provider-specific functionality in addition to the functionality based on the JPA 1.0 specifications. For an overview of Cosminexus JPA provider, notes on application implementation, and the usage methods of Cosminexus JPA provider, see 6. *Cosminexus JPA provider*.

(2) When the JPA providers from other vendors are used

The JPA providers are provided by other vendors. The JPA specifications clearly specify the interfaces between JPA providers and Application Server, and therefore you can also use JPA providers provided by other vendors and conforming to the JPA 1.0 specifications with Application Server.

When you use the other JPA providers from Application Server, you must specify the following settings:

- **Specifying JAR files**

You use one of the following methods to specify the JAR file containing the JPA provider implementation:

- Specify the JAR file under the `<configuration>` tag of the logical J2EE server (`j2ee-server`) in the Easy Setup definition file. To specify a JAR file, you specify `add.class.path` in the `<param-name>` tag and the JAR file in the `<param-value>` tag. For details on the Easy Setup definition file and the parameters to be specified, see 4.6 *Easy Setup definition file* in the *uCosminexus Application Server Definition Reference Guide*.
- Include JAR files in J2EE applications as a library.

- **Definitions in persistence.xml**

In the `<provider>` tag of `persistence.xml`, specify the implementation class name of `javax.persistence.PersistenceProvider` provided by the used JPA provider. For details, see 5.8.2(2) `<provider>` tag.

To use the functionality for monitoring the J2EE application execution time provided with Application Server, you must add the JPA provider classes and entity classes into the protected area list. For details on how to add a class into the protected area list, see 2.6 *criticalList.cfg (Protected area list file)* in the *uCosminexus Application Server Definition Reference Guide*.

Reference note

With the execution of applications using the JPA, you can use the trace based performance analysis functionality provided with Application Server.

- When Cosminexus JPA provider is used as the JPA provider
The trace based performance analysis can be output with both Application Server and Cosminexus JPA provider.
- When a JPA provider from other vendors is used
You can only use the trace based performance analysis output by Application Server.

Note that with Application Server, the trace based performance analysis is output by the EntityManagerFactory, EntityManager, EntityTransaction, and Query APIs of the `javax.persistence` package. Furthermore, the trace based performance analysis related to the entity life cycle callback is output with the JPA provider.

For an overview of the trace based performance analysis, see *7.2.1 Overview of the trace based performance analysis of Application Server* in the *uCosminexus Application Server Maintenance and Migration Guide*. For details on the points at which the trace based performance analysis is output, see *8. Trace Collection Points and PRF Trace Collection Levels of the Trace Based Performance Analysis* in the *uCosminexus Application Server Maintenance and Migration Guide*.

5.3.2 Available components

With Application Server, you can use the JPA with EJBs and Web applications. You can also use the JPA when user threads are used from Web applications. Note that you cannot use the JPA in the following environments or libraries:

- EJB client application environment
- J2EE application client environment
- Container extension library

The following table lists the components that can use the JPA.

Table 5-2: Components that can use the JPA

	Components	JPA usage
EJB	Stateless Session Bean (EJB 3.0 and later) ^{#1}	Y
	Stateful Session Bean (EJB 3.0 and later) ^{#1}	Y
	Stateless Session Bean (earlier than EJB 3.0)	N
	Stateful Session Bean (earlier than EJB 3.0)	N
	Interceptor	Y
	Message-driven Bean	N
	Entity Bean	N
Web application	Servlet, filter, event listener (Servlet 2.5 and later)	Y
	JSP, JSP tag handler, JSP event listener, JSP tag library event listener ^{#2} (Servlet 2.5 and later)	Y
	Servlet, filter, event listener (earlier than Servlet 2.5)	N
	JSP, JSP tag handler, JSP event listener, JSP tag library event listener (earlier than Servlet 2.5)	N

Legend:

Y: Available

N: Not available

^{#1}: With Application Server, you cannot specify the JPA definition using EJB 3.0 `ejb-jar.xml`. Therefore, you use annotations such as `@PersistenceUnit` and `@PersistenceContext` to define the references for the persistence units and persistence contexts.

^{#2} With Application Server, you cannot use annotations in the JSP tag library event listener. To use the JPA functionality in the JSP tag library event listener, you use `<persistence-unit-ref>` tag and the `<persistence-context-ref>` tag of `web.xml` to define the references for the persistence units and persistence contexts.

! Important note

If `true` is specified in the `metadata-complete` attribute of `web.xml` in Servlet 2.5 and later, the Web component annotations are not read. Therefore, you cannot use annotations to define the references for the persistence contexts or persistence units. However, you can define the references in `web.xml`.

5.3.3 Supported application formats

A J2EE application using the JPA is deployed on Application Server in one of the following formats:

- **Archive-format J2EE applications**
- **Exploded-archive format J2EE applications**

You can also replace a J2EE application that uses a deployed JPA. For the archive format, use the redeploy functionality and for the exploded archive format, use the reload functionality.

Note that when you use the reload functionality, updates are not detected for the O/R mapping file. However, the O/R mapping file is re-read when the file is reloaded. The following table describes the targets for update detection and the re-reading during reload.

Table 5-3: Targets for update detection and re-reading when reload is executed

Target classes and files	Update detection	Re-reading
Entity class	Y	Y
Mapped super class	Y	Y
Embedded class	Y	Y
<code>persistence.xml</code>	Y	Y
O/R mapping file (When <code>orm.xml</code> is allocated under <code>META-INF</code>)	N	Y
O/R mapping file (When <code>orm.xml</code> is allocated to the location specified in the <code><mapping-file></code> tag of <code>persistence.xml</code>)	Y	Y

Legend:

Y: Target

N: Not a target

For details on the J2EE applications in the archive-format and the exploded archive-format, see *13. Formats and Deployment of J2EE Applications*.

Important note

When you use an application with the JPA in the exploded archive format, do not delete the class or library JAR while the application is running. If the class or library JAR is deleted, Application Server and JPA provider might perform unexpected operations.

5.3.4 Supported class loader configuration

The J2EE applications using the JPA support the class loaders listed in the following table.

Table 5-4: Class loaders supported in a J2EE application using the JPA

Class loaders	Support
Default class loader configuration	Y
Class loader configuration used when local call optimization is performed #	Y
Class loader configuration for downward compatibility	N

Legend:

Y: Supported

N: Not supported

Note: The class loader configuration for the downward compatibility is used only in the basic mode and therefore, not supported.

#: Indicates the following specification in the `<configuration>` tag of the logical J2EE server (`j2ee-server`) in the Easy Setup definition file:

```
<param-name>ejbserver.rmi.localinvocation.scope</param-name>
<param-value>all</param-value>
```

5.3.5 available resource adapters

When you execute a J2EE application that uses the JPA, with Application Server, you can use a resource adapter with the connection factory interface `javax.sql.DataSource`. You can use DB Connector as the resource adapter provided with Application Server.

Also, you must deploy the resource adapter you want to use as the J2EE resource adapter. In the case of the J2EE applications using the JPA, you cannot include and deploy the resource adapter in the J2EE application.

The following table lists the resource adapters available with Application Server.

Table 5-5: Resource adapters available with Application Server

Connection factory interface	Deploy format of the resource adapter	Usage from the JPA
<code>javax.sql.DataSource</code>	Deployed as a J2EE resource adapter	Y
	Included and deployed in the J2EE application	N
Other than <code>javax.sql.DataSource</code>	--	N

Legend:

Y: Available

N: Not available

--: Not applicable

For details on the resource adapters available when you use Cosminexus JPA provider, see *6.2.3(3) Available DB Connectors*.

5.4 EntityManager

The *EntityManager* is an object that has an interface for registering and deleting entities for the database. This section gives an overview of *EntityManager*.

5.4.1 Methods provided with EntityManager

The *EntityManager* provides methods. The typical methods are as follows:

- **persist method (equivalent to SQL INSERT)**
The method used to add an entity object that executes `new` in the application, into the database.
- **find method (equivalent to SQL SELECT)**
The method used to search an entity object from the database.
- **remove method (equivalent to SQL DELETE)**
The method used to delete an entity object from the database.

The entity objects searched using the `find` method from *EntityManager* and the entity objects passed to *EntityManager* using the `persist` method are managed by *EntityManager*. If a field value of an entity object managed by *EntityManager* is changed, *EntityManager* automatically detects the change and applies the change to the database table.

With the JPA, an entity object managed by *EntityManager* is called a *managed entity*. By default, when a transaction is concluded, the entity is no longer managed by *EntityManager*. An entity that is no longer managed by *EntityManager* is called a *detached entity*.

5.4.2 Types of EntityManager

The types of *EntityManager* include the container-managed *EntityManager* and the application-managed *EntityManager*. The following is a description of each type:

(1) Container-managed EntityManager

The method entrusts the creation and destruction of *EntityManager* to the container. If you use a container-managed *EntityManager*, you can code an application without being aware of the generation and destruction of *EntityManager*. The following points describe how to obtain and how to destroy the container-managed *EntityManager*.

- **How to obtain the container-managed EntityManager**
To obtain the container-managed *EntityManager*, you use the DI or JNDI lookup in the application. *EntityManager* obtained using this method is *EntityManager* created by the container. You can use *EntityManager* obtained from a container as it is during the application coding.
For details on how to obtain the container-managed *EntityManager* from an application, see 5.6 *How to obtain the container-managed EntityManager*.
- **How to destroy the container-managed EntityManager**
The creation and destruction of *EntityManager* need not be coded in the application.

(2) Application-managed EntityManager

In this method, the application explicitly creates and destroys *EntityManager*. The life cycle is managed explicitly by application coding. The following points describe how to obtain and how to destroy the application-managed *EntityManager*.

- **How to obtain the application-managed EntityManager**
You use *EntityManagerFactory* to create *EntityManager* in the application. To obtain *EntityManagerFactory*, you use the DI or JNDI lookup in the application.
For details on how to obtain the application-managed *EntityManager*, see 5.7 *How to obtain the application-managed EntityManager*.

- **How to destroy the application-managed EntityManager**

You invoke the `close` method of `EntityManager` to destroy `EntityManager`.

5.4.3 Transaction control and EntityManager

The following two types of `EntityManager` are available depending on how the transactions are controlled:

- **JTA entity manager**
`EntityManager` in which the transactions are controlled by the JTA.
- **Resource local entity manager**
`EntityManager` in which the transactions are controlled by the `EntityTransaction` API.

The following table describes the relationship between the types of `EntityManager` and the transaction control methods.

Table 5-6: Relationship between types of `EntityManager` and transaction control methods

Types of <code>EntityManager</code>	Transaction control method	
	JTA	Resource local
Container-managed <code>EntityManager</code>	Y ^{#1}	N
Application-managed <code>EntityManager</code> ^{#2}	Y	Y

Legend:

- Y: Transaction can be controlled
- N: Transaction cannot be controlled
- JTA: JTA entity manager
- Resource local: Resource local entity manager

^{#1} The transaction is necessarily controlled by the JTA.

^{#2} You can select whether to control the transaction using the JTA or whether the application controls the transaction, by explicitly using the `EntityTransaction` API. When the `EntityTransaction` API is used, the transaction becomes a resource local transaction. Even if the JTA transaction exists, the transaction is controlled regardless of the JTA transaction.

You specify whether you want to use the JTA entity manager or the resource local entity manager in the definition of the persistence unit. For details on how to specify definitions in the persistence unit, see *5.8.1(2) transaction-type attribute*.

5.4.4 Persistence unit

You must define the following information, when the JPA is used from the application:

- Information about the entity classes in the application
- Information about the mapping between the entity classes and the database tables
- Information about the data source for the JPA provider to obtain the database connection

The unit that defines this information is called the *persistence unit*.

You define the persistence unit in `persistence.xml`. When the JPA is used in the Java EE environment, `persistence.xml` is allocated in the determined location in the EJB-JAR, WAR, or EAR files when the user packages the application.

You can include multiple persistence unit definitions in the `persistence.xml` file. You can also include multiple `persistence.xml` files in one application. As a result, you can define multiple persistence units in one application. When multiple persistence units are defined in an application, you specify the persistence unit to be used by the application in the `unitName` attribute of `@PersistenceContext`. Note that when the persistence unit to be used can be identified uniquely, such as when only one persistence unit is defined in the application, you can omit the `unitName` attribute.

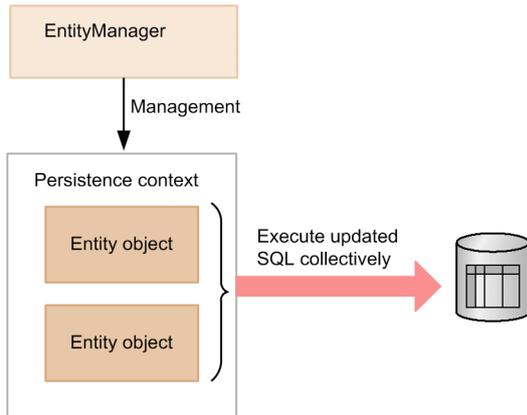
5.5 Persistence context

The EntityManager caches the entity objects to be updated and the searched entity objects. The persistence context is the cache of the entity objects cached by EntityManager.

5.5.1 EntityManager and persistence context

The following figure shows a relationship between EntityManager and persistence context.

Figure 5-4: Relationship between EntityManager and persistence context



The EntityManager inserts the managed entity object in the persistence context and manages the entity objects. When the application passes the entity object to the `persist` method or updates the field value of the managed entity object, the status of the entity object in the persistence context is changed. EntityManager synchronizes the statuses of the entity objects in the persistence context and the database table just before the transaction is committed. In order to apply the statuses of the entity objects within the persistence context to the database table, the update SQL statements are issued collectively at this time. As a result, the database locking time is shortened, and therefore you can improve the concurrent executability and update the data efficiently.

(1) Types of EntityManager

You decide whether to use the container-managed EntityManager or the application-managed EntityManager depending on the relationship between the persistence context and transaction.

- When the persistence context needs to be automatically propagated along with the JTA transaction**
 You use the container-managed EntityManager. If you use the container-managed EntityManager, the persistence context is automatically propagated along with the JTA transaction. Therefore, when multiple components are invoked in one JTA transaction, EntityManager used in the same JTA transaction can be associated with the same persistence context.
 As a result, the application need not pass the EntityManager references to the arguments used when the references are invoked from one component in another component.
- When the application needs to use the persistence context independently from the JTA transaction**
 You use the application-managed EntityManager. If you use the application-managed EntityManager, when you use another EntityManager in the same JTA transaction also, this EntityManager does not share the persistence context and has an independent persistence context.

(2) Types of persistence context

The persistence contexts are of two types depending on the lifetime:

- **Transaction scope persistence context**
- **Extended persistence context**

For the container-managed EntityManager, you can choose the type of persistence context. You specify the type of persistence context in the `type` attribute of `@PersistenceContext`. The default type is the transaction scope persistence context.

Note that the extended persistence context is always used for the application-managed EntityManager. You cannot choose the type of persistence context.

5.5.2 Persistence context when the container-managed EntityManager is used

When you use the container-managed EntityManager, the life cycle of the persistence context is managed by the container and the persistence context is automatically propagated along with the JTA transaction. You can choose the transaction scope persistence context or the extended persistence context as the type of the persistence context life cycle.

The following subsections describe the respective persistence contexts:

(1) Transaction scope persistence context

With the JPA specifications, the persistence context having the same life cycle as the transaction is called the *transaction scope persistence context*.

The EntityManager has the same life cycle as that of the transaction by default. Therefore, the updates cached in the persistence context of EntityManager are applied to the database when the transaction is committed.

(a) Life cycle of persistence context

The life cycle of the transaction scope persistence context is as follows:

- **Creating the persistence context**

The transaction scope persistence context is created when the container-managed EntityManager is first invoked in a JTA transaction.

The created persistence context is associated with the JTA transaction.

After that, when the container-managed EntityManager is used in the same JTA transaction, this persistence context is used.

- **Destroying the persistence context**

When the JTA transaction is committed or rolled back, the transaction scope persistence context is destroyed.

If the container-managed EntityManager is invoked outside the transaction, all the entities loaded from the database are immediately detached when the invocation of the EntityManager method ends.

(2) Extended persistence context

With the Java EE environment, if EntityManager is used from the Stateful Session Bean, you can set the same persistence context lifetime as the Stateful Session Bean lifetime. In this case, the updates are applied to the database every time the transaction is committed, but the entity objects managed in the persistence context are stored in the managed status as are across multiple transactions. The persistence context with the same life cycle as the Stateful Session Bean is called the *extended persistence context* in the JPA.

The extended persistence context is created simultaneously when the Stateful Session Bean is created and is associated with that Stateful Session Bean. Thereafter, the extended persistence context is destroyed simultaneously when the Stateful Session Bean is destroyed.

When the Stateful Session Bean creates another Stateful Session Bean and when the definition is such that the creating Stateful Session Bean and the created Stateful Session Bean both use the extended persistence context, the persistence context on the creating side is inherited on the created machine. The persistence context is inherited regardless of whether the transaction is active when the Stateful Session Bean is created. If the persistence context is inherited when the Stateful Session Bean is created, the persistence context is destroyed when all the Stateful Session Beans sharing that persistence context are destroyed.

(3) Extended persistence context and transactions

The extended persistence context exists from the time the EntityManager instance is created until the instance is closed. The extended persistence context supports multiple transactions and the invocation outside the EntityManager transaction.

The relationship with transactions is as follows:

- If EntityManager is invoked within the transaction scope or if the stateful session beans bound by the persistence context are invoked in the transaction scope, the entities managed by EntityManager participate in the transaction.
- Regardless of whether the transaction is running, the `persist`, `remove`, `merge`, and `refresh` operations might be performed. In this case, EntityManager participates in the transaction and the updates are applied in the database when the transaction is committed.
- Even after the transaction is committed, the references to the entity object are stored. The entity object is managed by EntityManager and is updated as an object managed between transactions (managed entity).

(4) Propagation of persistence context

When the container-managed EntityManager is used, the persistence context is propagated using the JTA transaction and might be associated with multiple EntityManager. However, the persistence context is only propagated with the same Application Server. The persistence context is not propagated in a remote Application Server.

The following points separately describe the propagation of the persistence context for the states when a component is invoked:

If the JTA transaction does not exist or the persistence context is not associated with the JTA transaction when the component is invoked

The persistence context is not propagated. The operations when EntityManager is invoked from this component are as follows:

- When EntityManager using the transaction scope persistence context is invoked, a new persistence context is created.
- When EntityManager using the extended persistence context is invoked, the extended persistence context associated with the invoked Stateful Session Bean is used.
- If the JTA transaction exists when EntityManager is invoked, the persistence context is associated with the JTA transaction.

If the JTA transaction is propagated and the persistence context is associated with the JTA transaction when the component is invoked

The operations when EntityManager is invoked from this component are as follows:

- If another persistence context is associated with the JTA transaction although the component is the Stateful Session Bean that already has the extended persistence context, the container throws `EJBException`.
- When EntityManager using the transaction scope persistence context is invoked, the persistence context associated with the propagated JTA transaction is used.

5.5.3 Persistence context when the application-managed EntityManager is used

When you use the application-managed EntityManager, the application directly invokes EntityManagerFactory of the JPA provider and manages the EntityManager life cycle and the creation and destruction of the persistence context. The life cycle of the application-managed persistence context can manage life cycles across multiple transactions.

(1) Managing the EntityManager life cycle

With the application, you use the `close` and `isOpen` methods of EntityManager to manage the life cycle of the application-managed EntityManager. If you invoke the `close` method of EntityManager, EntityManager, persistence context associated with EntityManager, and the other resources are released. After the `close` method is invoked, do not invoke methods other than EntityManager `getTransaction` method and `isOpen` method with the application. If other methods are invoked, `IllegalStateException` is thrown. If the `close` method is invoked when the

transaction is active, the persistence context remains stored until the transaction is concluded. The `isOpen` method of `EntityManager` returns `true` until `EntityManager` is closed and `false` after `EntityManager` is closed.

(2) Life cycle of persistence context

The life cycle of the application-managed persistence context is as follows:

- **Creating the persistence context**

The persistence context is created when the `createEntityManager` method of `EntityManagerFactory` is invoked.

- **Destroying the persistence context**

The persistence context is destroyed when the `close` method of `EntityManager` is invoked.

The application-managed persistence context is independent of the transaction. Therefore, this persistence context is not propagated along with the JTA transaction.

(3) Notes on using the JTA entity manager

When the JTA entity manager is used in the application-managed `EntityManager`, the invocation of `joinTransaction` of `EntityManager` when the application creates `EntityManager` outside the JTA transaction scope is the responsibility of the application. The application must report `EntityManager` that the transaction has started, so invoke the `joinTransaction` method of `EntityManager` after the transaction starts.

5.6 How to obtain the container-managed EntityManager

To obtain the container-managed EntityManager from the J2EE application, use one of the following methods:

- **Method of injecting EntityManager in the application field and setter method by using the DI**
- **Method of looking up EntityManager by using the JNDI from the application**

The following subsections describe each of the methods.

5.6.1 Method of injecting EntityManager in the application

The following two more methods are available for injecting EntityManager in the application fields and in the setter method:

1. Method of adding `@PersistenceContext` in the field or method at the inject destination
2. Method of specifying definitions in the `<persistence-context-ref>` tag of a DD (`web.xml`)

However, you cannot specify the JPA definition using EJB 3.0 `ejb-jar.xml` with Application Server. Therefore, if you want to use the JPA in an EJB, use the method specified in point 1.

The following is a description of the methods:

(1) Method of using `@PersistenceContext`

When you use `@PersistenceContext` to inject EntityManager, you add `@PersistenceContext` in the field and setter method where EntityManager is to be injected. The attributes that can be specified in `@PersistenceContext` are as follows:

(a) `unitName` attribute

In the `unitName` attribute, you specify the name of the persistence unit defined in `persistence.xml`. However, when the persistence unit to be used can be uniquely identified, such as when only one persistence unit is defined in the EJB-JAR and WAR or EAR file, you can omit the `unitName` attribute. For details on the persistence unit used when the `unitName` attribute is omitted, see *5.11.2 Reference scope of the persistence unit name*.

(b) `type` attribute

In the `type` attribute, you specify the life cycle type of the persistence context. You can specify `PersistenceContextType.TRANSACTION` or `PersistenceContextType.EXTENDED`.

- **When `PersistenceContextType.TRANSACTION` is specified**

The transaction scope persistence context is used and the transaction lifetime and the persistence context lifetime becomes the same.

- **When `PersistenceContextType.EXTENDED` is specified**

The extended persistence context is used and the Stateful Session Bean lifetime and the persistence context lifetime becomes the same.

Note that `@PersistenceContext` in which `PersistenceContextType.EXTENDED` is specified in the `type` attribute can only be added in the field or method of the Stateful Session Bean.

When the `type` attribute is omitted, the default value is `PersistenceContextType.TRANSACTION`.

(c) `properties` attribute

In the `properties` attribute, you can specify the properties for the JPA provider used for setting up a persistence unit. The properties specified here are passed to the JPA provider when EntityManager is obtained from the JPA provider.

(d) name attribute

When you use injection, normally you need not specify the name attribute, but if specified, EntityManager is registered in the JNDI Namespace (`java:comp/env`) with the name specified in the name attribute. An example of using `@PersistenceContext` to inject EntityManager is as follows:

```
@Stateless
public class InventoryManagerBean implements InventoryManager {
    @PersistenceContext(unitName="myUnit")
    private EntityManager em;
    ...
}
```

(2) Method of using the <persistence-context-ref> tag of the DD

When you use a DD to inject EntityManager, you define the following tags in the <persistence-context-ref> tag of the DD:

(a) <description> tag

In the <description> tag, the user can freely code the explanation for the EntityManager references to be defined. Even if this tag is specified, the specified contents do not affect the operations of the application. You can also omit this tag.

(b) <persistence-context-ref-name> tag

In the <persistence-context-ref-name> tag, specify the name with which EntityManager is registered in the JNDI Namespace. The specified name is the relative path from `java:comp/env`. The JNDI registration name of EntityManager is not mandatory, but JPA specifications recommend that the name be set under `java:comp/env/persistence`.

(c) <persistence-unit-name> tag

In the <persistence-unit-name> tag, you specify the name of the persistence unit defined in `persistence.xml`. However, when the persistence unit to be used can be uniquely identified, such as when only one persistence unit is defined in the EJB-JAR and WAR or EAR, you can omit the <persistence-unit-name> tag. For details on the persistence unit used when the <persistence-unit-name> tag is omitted, see 5.11.2 *Reference scope of the persistence unit name*.

(d) <persistence-context-type> tag

In the <persistence-context-type> tag, you specify the life cycle type of the persistence context. You specify `Transaction` or `Extended`.

- **When Transaction is specified**

The transaction scope persistence context is used and the transaction lifetime and the persistence context lifetime becomes the same.

- **When Extended is specified**

The extended persistence context is used and the Stateful Session Bean lifetime and the persistence context lifetime becomes the same.

Note that the <persistence-context-ref> tag in which `Extended` is specified in the <persistence-context-type> tag can only be defined for the Stateful Session Bean.

When the <persistence-context-type> tag is omitted, the default value is `Transaction`.

(e) <persistence-property> tag

In the <persistence-property> tag, you can specify the properties for the JPA provider used for setting up the persistence unit. The properties specified here are passed to the JPA provider when EntityManager factory is obtained from the JPA provider. You can omit this tag.

(f) `<injection-target>` tag

In the `<injection-target-class>` tag of the `<injection-target>` tag, you specify the inject destination class. In the `<injection-target-name>` tag of the `<injection-target>` tag, you specify the field name or setter method name at the inject destination. An example of defining the `<persistence-context-ref>` tag in `web.xml` and injecting `EntityManager` is as follows:

```

...
<web-app>
  ...
  <servlet>
    <display-name>InventoryManagerServlet</display-name>
    <servlet-name>InventoryManagerServlet</servlet-name>
    <servlet-class>com.hitachi.InventoryManagerServlet</servlet-class>
  </servlet>
  ...
  <persistence-context-ref>
    <description>
      Persistence context for the inventory management application.
    </description>
    <persistence-context-ref-name>persistence/InventoryAppMgr
    </persistence-context-ref-name>
    <persistence-unit-name>InventoryManagement</persistence-unit-name>
    <persistence-context-type>Transaction</persistence-context-type>
    <injection-target>
      <injection-target-class>
        com.hitachi.InventoryManagerServlet
      </injection-target-class>
      <injection-target-name>em</injection-target-name>
    </injection-target>
  </persistence-context-ref>
  ...
</web-app>
...

```

5.6.2 Method of looking up `EntityManager` from the application

The following two more methods are available for using the JNDI to look up `EntityManager` from the application:

1. Method of adding `@PersistenceContext` in the class for looking up `EntityManager` and defining the `EntityManager` references
2. Method of defining the `<persistence-context-ref>` tag in the DD (`web.xml`) and defining the `EntityManager` references

However, you cannot specify the JPA definition using EJB 3.0 `ejb-jar.xml` with Application Server. Therefore, if you want to use the JPA in an EJB, use the method specified in point 1.

The following subsections describe the methods:

(1) Method of using `@PersistenceContext`

When you use `@PersistenceContext` to define the `EntityManager` references, add `@PersistenceContext` in the class that executes lookup.

The attributes that can be specified in `@PersistenceContext` are as follows:

(a) name attribute

In the name attribute, you specify the lookup name with which the application code will look up `EntityManager`. The specified lookup name is the relative path from `java:comp/env`. The lookup name of `EntityManager` is not mandatory, but the JPA specifications recommend that the name be set up under `java:comp/env/persistence`.

For `@PersistenceContext` the same attributes are used that are used as other attributes in *5.6.1 Method of injecting `EntityManager` in the application*. Note that only the Stateful Session Bean can look up `EntityManager` in the extended scope. Also, for adding multiple `@PersistenceContext` in one class, you add `@PersistenceContexts` in the class and specify the array of `@PersistenceContext` as the value

attribute. An example of using `@PersistenceContext` to look up `EntityManager` from `SessionContext` is as follows:

```

@Stateless
@PersistenceContext(name="persistence/OrderEM")
public class MySessionBean implements MyInterface {
    @Resource SessionContext ctx;
    public void doSomething() {
        ...
        EntityManager em = (EntityManager)ctx.lookup("persistence/OrderEM");
        ...
    }
}

```

An example of using `@PersistenceContext` to look up `EntityManager` from `InitialContext` is as follows:

```

@Stateless
@PersistenceContext(name="persistence/InventoryAppMgr")
public class InventoryManagerBean implements InventoryManager {
    public void updateInventory(...) {
        ...
        Context initCtx = new InitialContext();
        EntityManager em = (EntityManager)
            initCtx.lookup("java:comp/env/persistence/InventoryAppMgr");
        ...
    }
}

```

(2) Method of using the `<persistence-context-ref>` tag of the DD

When you use the DD to define the `EntityManager` references, define the following tags in the `<persistence-context-ref>` tag of the DD:

(a) `<persistence-context-ref-name>` tag

In the `<persistence-context-ref-name>` tag, specify the lookup name with which the application code will look up `EntityManager`. The specified lookup name is the relative path from `java:comp/env`. The lookup name of `EntityManager` is not mandatory, but the JPA specifications recommend that the name be set under `java:comp/env/persistence`.

For the `<persistence-context-ref>` tag, the same tags are used that are used as other tags in *5.6.1 Method of injecting `EntityManager` in the application*. However, when `EntityManager` is obtained with the JNDI lookup, you cannot specify `<injection-target>`. Note that only the Stateful Session Bean can look up `EntityManager` in the extended scope.

An example of defining `<persistence-context-ref>` in `web.xml` is as follows:

```

...
<web-app>
    ...
    <servlet>
        <display-name>InventoryManagerServlet</display-name>
        <servlet-name>InventoryManagerServlet</servlet-name>
        <servlet-class>com.hitachi.InventoryManagerServlet</servlet-class>
    </servlet>
    ...
    <persistence-context-ref>
        <description>
            Persistence context for the inventory management application.
        </description>
        <persistence-context-ref-name>
            persistence/InventoryAppMgr
        </persistence-context-ref-name>
        <persistence-unit-name>InventoryManagement</persistence-unit-name>
        <persistence-context-type>Transaction</persistence-context-type>
    </persistence-context-ref>
    ...
</web-app>
...

```

5.6.3 Overriding the `@PersistenceContext` definition using the DD

If the `<persistence-context-ref>` tag is defined in the DD when `@PersistenceContext` is mentioned in the application, the contents defined in the annotation are overwritten by the contents defined in the DD. In this case, the mapping between the annotations and DD is determined with the mapping between the `name` attribute of `@PersistenceContext` and the `<persistence-context-ref-name>` tag present under the `<persistence-context-ref>` tag of the DD. Note that even if the `name` attribute is not explicitly specified in `@PersistenceContext`, the `name` attribute contains the default value.

The precautions to be taken when the attributes specified in `@PersistenceContext` are overridden with the DD tags are as follows:

(1) `<persistence-unit-name>` tag and `unitName` attribute

The `<persistence-unit-name>` tag of the DD overrides the `unitName` attribute of `@PersistenceContext`. Normally, if you change the persistence unit name, the application stops operating, so take care when you define the DD and annotations.

(2) `<persistence-context-type>` tag and `type` attribute

The `<persistence-context-type>` tag of the DD overrides the `type` attribute of `@PersistenceContext`. Normally, if you change the life cycle type of the persistence context, the application stops operating, so take care when you define the DD and annotations.

(3) `<persistence-property>` tag and `properties` attribute

The properties specified in the `<persistence-property>` tag of the DD are added to the properties specified in the `properties` attribute of `@PersistenceContext`. However, if the property name is the same, the property value is overridden.

(4) `<injection-target>` tag

The injection target cannot be overridden. Note that when the `<injection-target>` tag is coded in the DD, accurately specify the fields and methods in which `@PersistenceContext` is added.

5.7 How to obtain the application-managed EntityManager

When the application-managed EntityManager is used, the application uses EntityManagerFactory to create EntityManager. There are two methods by which the application obtains EntityManagerFactory:

- Method of using the DI to inject EntityManagerFactory in the application field and setter method
- Method of using the JNDI from the application to look up EntityManagerFactory

The following is a description of the methods:

5.7.1 Method of injecting EntityManagerFactory in the application

The following two more methods are available for injecting EntityManagerFactory in the application fields and setter method:

1. Method of adding @PersistenceUnit in the field or method at the inject destination
2. Method of specifying a definition in the <persistence-unit-ref> tag of the DD (web.xml)

However, you cannot specify the JPA definition using EJB 3.0 ejb-jar.xml with Application Server. Therefore, if you want to use the JPA in an EJB, use the method specified in point 1.

The following subsections describe the methods:

(1) Method of using @PersistenceUnit

When you use @PersistenceUnit to inject EntityManagerFactory, add @PersistenceUnit in the field and setter method where EntityManagerFactory is to be injected. The attributes that can be specified in @PersistenceUnit are as follows:

(a) unitName attribute

In the unitName attribute, you specify the name of the persistence unit defined in persistence.xml. However, when the persistence unit to be used can be uniquely identified, such as when only one persistence unit is defined in the EJB-JAR and WAR or EAR, you can omit the unitName attribute. For details on the persistence unit used when the unitName attribute is omitted, see 5.11.2 *Reference scope of the persistence unit name*.

(b) name attribute

When you use injection, normally you need not specify the name attribute, but if you specify, EntityManager is registered in the JNDI Namespace (java:comp/env) with the name specified in the name attribute. An example of using @PersistenceUnit to inject EntityManagerFactory is as follows:

```
@Stateless
public class InventoryManagerBean implements InventoryManager {
    @PersistenceUnit(unitName="myUnit")
    private EntityManagerFactory emf;
    ...
}
```

(2) Method of using the <persistence-unit-ref> tag of the DD

When you use the DD to inject EntityManagerFactory, define the following tags in the <persistence-unit-ref> tag of the DD:

(a) <persistence-unit-ref-name> tag

In the <persistence-unit-ref-name> tag, you specify the name with which EntityManagerFactory is registered in the JNDI Namespace. The specified name is the relative path from java:comp/env.

The JNDI registration name of `EntityManagerFactory` is not mandatory, but the JPA specifications recommend that the name be set under `java:comp/env/persistence`.

(b) `<description>` tag

In the `<description>` tag, you can freely code the `EntityManagerFactory` references to be defined. Even if this element is specified, the specified contents do not affect the operations of the application. You can also omit this tag.

(c) `<persistence-unit-name>` tag

In the `<persistence-unit-name>` tag, you specify the name of the persistence unit defined in `persistence.xml`. However, when the persistence unit to be used can be uniquely identified, such as when only one persistence unit is defined in the EJB-JAR and WAR or EAR, you can omit the `<persistence-unit-name>` tag. For details on the persistence unit used when the `<persistence-unit-name>` tag is omitted, see 5.11.2 *Reference scope of the persistence unit name*.

(d) `<injection-target>` tag

In the `<injection-target-class>` tag of the `<injection-target>` tag, you specify the inject destination class. In the `<injection-target-name>` tag of the `<injection-target>` tag, you specify the field name or setter method name at the inject destination. An example of defining the `<persistence-unit-ref>` tag in `web.xml` and injecting `EntityManagerFactory` is as follows:

```

...
<web-app>
  ...
  <servlet>
    <display-name>InventoryManagerServlet</display-name>
    <servlet-name>InventoryManagerServlet</servlet-name>
    <servlet-class>com.hitachi.InventoryManagerServlet</servlet-class>
  </servlet>
  ...
  <persistence-unit-ref>
    <description>
      Persistence unit for the inventory management application.
    </description>
    <persistence-unit-ref-name>persistence/InventoryAppDB</persistence-unit-ref-name>
    <persistence-unit-name>InventoryManagement</persistence-unit-name>
    <injection-target>
      <injection-target-class>
        com.hitachi.InventoryManagerServlet
      </injection-target-class>
      <injection-target-name>emf</injection-target-name>
    </injection-target>
  </persistence-unit-ref>
  ...
</web-app>
...

```

5.7.2 Method of looking up `EntityManagerFactory` from the application

The following two more methods are available for using the JNDI to look up `EntityManagerFactory` from the application:

1. Method of adding `@PersistenceUnit` in the class for looking up `EntityManagerFactory` and defining the `EntityManagerFactory` references
2. Method of defining the `<persistence-unit-ref>` tag in the DD (`web.xml`) and defining the `EntityManagerFactory` references

However, you cannot specify the JPA definition using EJB 3.0 `ejb-jar.xml` with Application Server. Therefore, if you want to use the JPA in an EJB, use the method specified in 1.

The following subsections describe the methods:

(1) Method of using @PersistenceUnit

When you use @PersistenceUnit to define the EntityManagerFactory references, you add @PersistenceUnit in the class that executes lookup. The attributes that can be specified in @PersistenceUnit are as follows:

(a) name attribute

In the name attribute, you specify the lookup name with which the application code will look up EntityManagerFactory. The specified lookup name is the relative path from java:comp/env. The lookup name of EntityManagerFactory is not required, but the JPA specifications recommend that the name be set up under java:comp/env/persistence.

For @PersistenceUnit, the same attributes are used that are used as the other attributes in *5.7.1 Method of injecting EntityManagerFactory in the application*. Note that to add multiple @PersistenceUnit in one class, you add @PersistenceUnits in the class and specify the array of @PersistenceUnit as the value attribute. An example of using @PersistenceUnit to look up EntityManagerFactory from SessionContext is as follows:

```
@Stateless
@PersistenceUnit(name="persistence/InventoryAppDB")
public class InventoryManagerBean implements InventoryManager {
    @Resource SessionContext ctx;
    public void updateInventory(...) {
        ...
        EntityManagerFactory emf = (EntityManagerFactory)
        ctx.lookup("persistence/InventoryAppDB");
        EntityManager em = emf.createEntityManager();
        ...
    }
}
```

An example of using @PersistenceUnit to look up EntityManagerFactory from InitialContext is as follows:

```
@Stateless
@PersistenceUnit(name="persistence/InventoryAppDB")
public class InventoryManagerBean implements InventoryManager {
    public void updateInventory(...) {
        Context initCtx = new InitialContext();
        EntityManagerFactory emf = (EntityManagerFactory)
        initCtx.lookup("java:comp/env/persistence/InventoryAppDB");
        EntityManager em = emf.createEntityManager();
        ...
    }
}
```

(2) Method of using the <persistence-unit-ref> tag of the DD

When you use the DD to define the EntityManagerFactory references, define the <persistence-unit-ref> tag of the DD:

(a) <persistence-unit-ref-name> tag

In the <persistence-unit-ref-name> tag, specify the lookup name with which the application code will look up EntityManagerFactory. The specified lookup name is the relative path from java:comp/env. The lookup name of EntityManagerFactory is not mandatory, but the JPA specifications recommend that the name be set under java:comp/env/persistence.

For the <persistence-unit-ref> tag, the same tags are used that are used as the other tags in *5.7.1 Method of injecting EntityManagerFactory in the application*. However, when EntityManagerFactory is obtained with the JNDI lookup, you cannot specify <injection-target> tag. An example of defining <persistence-unit-ref> in web.xml is as follows:

```
...
<web-app>
    ...
    <servlet>
        <display-name>InventoryManagerServlet</display-name>
```

```

<servlet-name>InventoryManagerServlet</servlet-name>
<servlet-class>com.hitachi.InventoryManagerServlet</servlet-class>
</servlet>
...
<persistence-unit-ref>
<description>
  Persistence unit for the inventory management application.
</description>
<persistence-unit-ref-name>
  persistence/InventoryAppDB
</persistence-unit-ref-name>
<persistence-unit-name>InventoryManagement</persistence-unit-name>
</persistence-unit-ref>
...
</web-app>
...

```

5.7.3 Overriding the @PersistenceUnit definition using the DD

If the `<persistence-unit-ref>` tag is defined in the DD when `@PersistenceUnit` is mentioned in the application, the contents defined in the annotation are overwritten by the contents defined in the DD. In this case, the mapping between the annotations and DD is determined with the mapping between the name attribute of `@PersistenceUnit` and the `<persistence-unit-ref-name>` tag present under the `<persistence-unit-ref>` tag of the DD. Note that even if the name attribute is not explicitly specified in `@PersistenceUnit`, the name attribute contains the default value.

The precautions to be taken when the attributes specified in `@PersistenceUnit` are overridden with the DD tags are as follows:

(1) `<persistence-unit-name>` and `unitName` attribute

The `<persistence-unit-name>` tag of the DD overrides the `unitName` attribute of `@PersistenceUnit`. Normally, if you change the persistence unit name, the application stops operating, so take care when you make changes.

(2) `<injection-target>` tag

The injection target cannot be overridden with the DD. Note that when the `<injection-target>` tag is coded in the DD, you must accurately specify the fields and methods in which `@PersistenceContext` is added.

5.8 Definitions in persistence.xml

You define the persistence unit information by using the `<persistence-unit>` tag of `persistence.xml`. This section describes the attributes of the `<persistence-unit>` tag and the tags specified under the `<persistence-unit>` tag.

Note that the space and linefeed characters added at the beginning and end of values specified in the attributes of the `<persistence-unit>` tag and in the tags specified under the `<persistence-unit>` tag are ignored.

For details on the tags in `persistence.xml`, see *6.2 persistence.xml* in the *uCosminexus Application Server Definition Reference Guide*.

5.8.1 Attributes specified in the `<persistence-unit>` tag

In the `<persistence-unit>` tag, you specify the `name` attribute and `transaction-type` attribute.

(1) `name` attribute

You specify the name of the persistence unit to be defined. The name specified here is referenced from the `unitName` attribute of `@PersistenceUnit` or `@PersistenceContext` in the case of annotations. Also, in the case of the DD, the name specified here is referenced from the `<persistence-unit-name>` tag under the `<persistence-context-ref>` tag or under the `<persistence-unit-ref>` tag.

You cannot omit the `name` attribute. Also, when the JPA is used with Application Server, you cannot specify null in the `name` attribute. Specify a string of at least 1 character.

(2) `transaction-type` attribute

In the persistence unit to be defined, you specify whether the JTA will control the transaction or whether the application will control the transaction by using `javax.persistence.EntityTransaction`.

- **When the transaction is controlled by the JTA**

Specify JTA in the `transaction-type` attribute. When you specify JTA, you must also specify the `<jta-data-source>` tag at the same time.

- **When the application controls the transaction by using `EntityTransaction`**

Specify `RESOURCE_LOCAL` in the `transaction-type` attribute. When you specify `RESOURCE_LOCAL`, you must also specify the `<non-jta-data-source>` tag at the same time.

Note that if the `transaction-type` attribute is omitted, the default value is JTA.

5.8.2 Tags specified under the `<persistence-unit>` tag

The tags listed in the following table are specified under the `<persistence-unit>` tag.

Table 5-7: Tags specified under the `<persistence-unit>` tag

Specified tags	Settings
<code><description></code> tag	Describes the persistence unit.
<code><provider></code> tag	Specifies the JPA provider used.
<code><jta-data-source></code> tag <code><non-jta-data-source></code> tag	Specifies the JTA data source or non-JTA data source used in the JPA provider.
<code><mapping-file></code> tag	Specifies the O/R mapping file used.
<code><jar-file></code> tag	Specifies the name of the JAR file containing the <code>entity</code> class, <code>embeddable</code> class, and <code>mappedsuper</code> class.

Specified tags	Settings
<class> tag	Specifies the entity class, embeddable class, and mappedsuper class.
<exclude-unlisted-classes> tag	Specifies whether to handle the class as a Persistence class.
<properties> tag	Specifies the JPA provider-specific properties.

The following points describe respective tags:

(1) <description> tag

The user can freely code the explanation about the persistence unit. The contents specified here do not affect the operations of the application. Note that you can omit this tag.

(2) <provider> tag

Specifies the JPA provider used in the persistence unit. You specify the implementation class name of the `javax.persistence.spi.PersistenceProvider` interface of the JPA provider with the fully qualified name containing the package name. You can omit this tag.

If this tag is omitted, the default JPA provider specified in the Easy Setup definition file is used. Also, when this tag is omitted and the default JPA provider is not specified in the Easy Setup definition file, Cosminexus JPA provider is used as the JPA provider.

If the application depends on a specific JPA provider functionality and behavior, make sure to specify the <provider> tag.

Tip

To specify the default JPA provider in the Easy Setup definition file, specify `ejbserver.jpa.defaultProviderClassName` in the <param-name> tag of the logical J2EE server and specify the name of the default JPA provider class in the <param-value> tag.

Note that if the `ejbserver.jpa.overrideProvider` parameter is specified in the <param-name> tag of the logical J2EE server in the Easy Setup definition file, the name of the JPA provider class specified in the <param-value> tag of the `ejbserver.jpa.overrideProvider` parameter is used on a higher priority than the value specified in the <provider> tag and the `ejbserver.jpa.defaultProviderClassName` parameter.

For details on the parameters to be specified in the Easy Setup definition file, see *4.6.2 Contents specified in the Easy Setup definition file* in the *uCosminexus Application Server Definition Reference Guide*.

The following table describes the priority for determining the JPA provider used in the persistence unit.

Table 5-8: Priority for determining the JPA provider used in the persistence unit

Priority	JPA provider to be used
1	Value specified in <code>ejbserver.jpa.overrideProvider</code> property of the Easy Setup definition file
2	Value specified in the <provider> tag of <code>persistence.xml</code>
3	Cosminexus JPA provider (The JPA provider can also be changed by using the <code>ejbserver.jpa.defaultProviderClassName</code> parameter of the Easy Setup definition file)

(3) <jta-data-source> tag and <non-jta-data-source> tag

Specifies the JTA data source or non-JTA data source used by the JPA provider. The value specified here is product-dependent as per the JPA specifications, but define the data source references with Application Server as follows:

- **When referencing a resource adapter conforming to Connector 1.0**
Specify the '*Display-name-of-resource-adapter*' or '*Optional-name-of-resource-adapter*'.
- **When referencing a resource adapter conforming to Connector 1.5**

Specify the '*Display-name-of-resource-adapter!Connection-definition-identifier*' or '*Optional-name-of-resource-adapter*'.

The specified value is interpreted as '*display-name-of-resource-adapter*' or '*display-name-of-resource-adapter ! Connection-definition-identifier*' and the relevant resource adapter is searched. If the relevant resource adapter does not exist, the specified value is interpreted as '*Optional-name-of-resource-adapter*' and the relevant resource adapter is searched.

The resource adapter to be referenced must be deployed as a J2EE resource adapter (method of deploying the resource adapter as a standalone module). Start the resource adapter before starting the application containing the persistence unit.

You can omit the `<jta-data-source>` tag and `<non-jta-data-source>` tag. If the tags are omitted, the value specified in the `ejbserver.jpa.defaultJtaDsName` parameter or `ejbserver.jpa.defaultNonJtaDsName` parameter of the Easy Setup definition file is used. However, these properties do not have default values.

If a value is specified in the `ejbserver.jpa.overrideJtaDsName` parameter or `ejbserver.jpa.overrideNonJtaDsName` parameter, this value is used on higher priority than the value specified in the `<jta-data-source>` tag and `<non-jta-data-source>` tag and the value specified in the `ejbserver.jpa.defaultJtaDsName` parameter and `ejbserver.jpa.defaultNonJtaDsName` parameter.

You must specify `LocalTransaction` or `XATransaction` in the transaction support level of the resource adapter specified in the `<jta-data-source>` tag. You must also specify `NoTransaction` in the transaction support level of the resource adapter specified in `<non-jta-data-source>`.

The following table lists the priority for determining the JTA data source and non-JTA data source used in the persistence unit.

Table 5-9: Priority for determining the JTA data source and non-JTA data source used in the persistence unit

Priority	JPA provider to be used
1	Value specified in the <code>ejbserver.jpa.overrideJtaDsName</code> property or <code>ejbserver.jpa.overrideNonJtaDsName</code> property of the Easy Setup definition file
2	Value specified in the <code><jta-data-source></code> or <code><non-jta-data-source></code> element of <code>persistence.xml</code>
3	Value specified in the <code>ejbserver.jpa.defaultJtaDsName</code> property or <code>ejbserver.jpa.defaultNonJtaDsName</code> property of the Easy Setup definition file

Important note

If a resource adapter display name containing characters other than one-byte alphanumeric characters and underscore (`_`) is specified in the `<jta-data-source>` tag or `<non-jta-data-source>` tag of `persistence.xml`, replace that character with an underscore (`_`). However, if the replaced display name is duplicated with another resource adapter display name or optional name, note that the persistence unit might perform operations using an unintended data source.

(4) `<mapping-file>`, `<jar-file>`, `<class>`, and `<exclude-unlisted-classes>` tag

The following two methods are available for specifying the entity class, embeddable class, mapped superclass included in the persistence unit:

1. Method specified explicitly by using the O/R mapping file and the `<class>` tag
2. Method that is not specified explicitly and uses auto-search by JPA provider

A description of method 1 is as follows:

(a) Specifying the classes using the O/R mapping file

If an XML file named `orm.xml` is allocated under `META-INF` at the persistence unit root or under `META-INF` in another JAR file referenced with the `<jar-file>` tag from the persistence unit, the file is automatically handled as

an O/R mapping file even if the file is not specified in the `<mapping-file>` tag. Furthermore, if the name of the XML file that can be loaded on the class path is specified in the `<mapping-file>` tag, that XML file is also handled as an O/R mapping file. If one persistence unit contains multiple O/R mapping files, the mapping information is read from all the O/R mapping files. However, the operations for duplicate mapping defined between multiple O/R mapping files are not provided.

(b) Specifying the JAR file that searches the persistence class

In the `<jar-file>` tag, you can specify the JAR file containing the persistence class and O/R mapping file. From the JAR file specified in the `<jar-file>` tag, the class in which `@Entity`, `@Embeddable`, and `@MappedSuperclass` are added is searched and the mapping information is obtained automatically. If `META-INF/orm.xml` exists in the specified JAR file, the mapping information is also obtained from `orm.xml`. Note that you can also use the `<jar-file>` tag and the `<mapping-file>` tag together.

The JAR files that can be specified in the `<jar-file>` tag must be included in the class path. The following JAR files can be specified:

- JAR file placed in the EAR root
- JAR file placed in the library directory of the EAR file
- EJB-JAR
- JAR file placed in `WEB-INF/lib` in the WAR

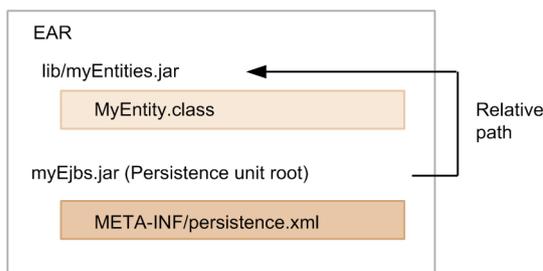
However, you cannot specify the JAR file placed in `WEB-INF/lib` in the WAR in the `<jar-file>` tag of the persistence unit defined in the EAR level or EJB-JAR level. This is because the JAR file placed in `WEB-INF/lib` in the WAR is loaded using the WEB application class loader and, therefore, can only be referenced from the components in the WAR.

Also, you can only specify the JAR file included in the same WAR from the `<jar-file>` tag of the persistence unit defined in the WAR level. This is because the classes included in the JAR file allocated to a location other than the same WAR might be loaded using the class loader before the deployment of the persistence unit defined in the WAR level. In such cases, the conversion of the byte code by the JPA provider might not be performed correctly.

In the `<jar-file>` tag, specify the relative path from the persistence unit root to the JAR file. The following is an example of specification:

Example 1

The specification of the relative path shown in the following figure is described below:

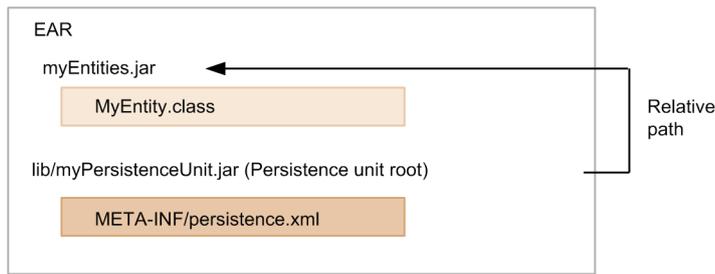


- The EAR lib contains `myEntities.jar` that stores the entity class.
- Specify `myEntities.jar` in the `<jar-file>` tag of `META-INF/persistence.xml` of EJB-JAR placed in the EAR root.

In this figure, the persistence unit root becomes EJB-JAR, so specify the relative path from EJB-JAR to `myEntities.jar`. The relative path is `lib/myEntities.jar`. Therefore, specify `lib/myEntities.jar` in the `<jar-file>` tag.

Example 2

The specification of the relative path shown in the following figure is described below:

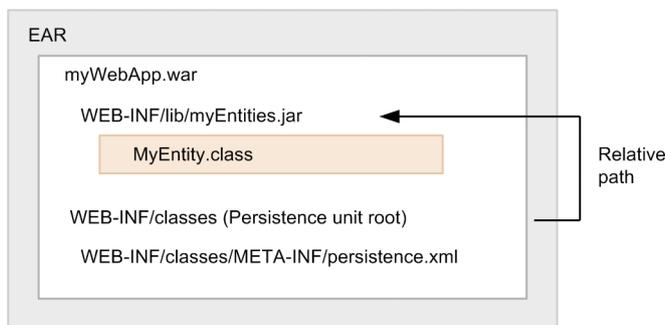


- The EAR root contains `myEntities.jar` that stores the entity class.
- Specify `myEntities.jar` in the `<jar-file>` tag of `META-INF/persistence.xml` of `lib/myPersistenceUnit.jar` in EAR.

In this figure, the persistence unit root becomes `myPersistenceUnit.jar`, so specify the relative path from `myPersistenceUnit.jar` to `myEntities.jar`. The relative path is `../myEntities.jar`. Therefore, specify `../myEntities.jar` in the `<jar-file>` tag.

Example 3

The specification of the relative path shown in the following figure is described below:



- `WEB-INF/lib` of the WAR contains `myEntities.jar` that stores the entity class.
- Specify `myEntities.jar` in the `<jar-file>` tag of `WEB-INF/classes/META-INF/persistence.xml` of the WAR.

In this figure, the persistence unit root becomes `WEB-INF/classes` of the WAR, so specify the relative path from `WEB-INF/classes` to `myEntities.jar`. The relative path is `../lib/myEntities.jar`. Therefore, specify `../lib/myEntities.jar` in the `<jar-file>` tag.

(c) Specifying the persistence class list explicitly

If you use the `<class>` tag, you can explicitly specify the persistence class list. The mapping information is obtained from the annotation added in the specified class. Note that you can also use the `<class>` tag in combination with the `<mapping-file>` tag and `<jar-file>` tag.

(d) Allocating the persistence class with the added annotation to the persistence unit root

From the persistence unit root, the persistence class in which `@Entity`, `@Embeddable`, and `@MappedSuperclass` are added is automatically searched. The mapping information is obtained from the annotation added in the class. If you do not want to add the annotation-added class, which is allocated to the persistence unit root, in the persistence unit, you must specify the `<exclude-unlisted-classes>` tag beforehand.

(5) `<properties>` tag

You can specify the vendor-specific properties of the JPA provider. If you specify properties that cannot be understood by the JPA provider, the properties are ignored. Note that you cannot specify properties beginning with `javax.persistence` in `<properties>`.

If you specify properties with property names beginning with the prefix `ejbserver.jpa.emfprop.` as the system properties, the properties with the prefix removed are added in the persistence unit properties.

5.9 Allocating persistence.xml

You allocate `persistence.xml` in the EJB-JAR, WAR, or EAR. Make sure that you place `persistence.xml` under `META-INF`. The path with `persistence.xml` under `META-INF` is called the *persistence unit root*.

The locations in which `persistence.xml` can be allocated are as follows:

- `META-INF/persistence.xml` of EJB-JAR
- `WEB-INF/classes/META-INF/persistence.xml` of the WAR
- `META-INF/persistence.xml` in the jar file placed under `WEB-INF/lib` of the WAR
- `META-INF/persistence.xml` in the jar file placed in the EAR root directory
- `META-INF/persistence.xml` in the jar file placed in the EAR library directory

You can define multiple persistence units in one `persistence.xml`. You must name the persistence unit. However, you cannot define multiple persistence units with duplicated names in one EJB-JAR, WAR, or EAR.

The class managed with the persistence unit defined in EAR is loaded using the application class loader and can be referenced from all the components in the application.

Furthermore, when the same entity class is referenced from the components in different EJB-JAR and WAR, the referenced class is the same unique class even if the persistence units are different.

5.10 JPA interfaces

This section introduces the JPA interfaces and describes the `javax.persistence.EntityManager` interface and `javax.persistence.EntityManagerFactory` interface.

5.10.1 `javax.persistence.EntityManager` interface

This subsection describes the interface definition and the notes on the `javax.persistence.EntityManager` interface.

(1) Definition of the interface

```
package javax.persistence;

/**
 * Interface for operating the persistence context.
 *
 * * EntityManager instance is associated with the persistence context.
 * * The persistence context is a set of entity instances and
 * * the entity instance is unique for each
 * * perpetuated entity.
 * * The entity instances and their life cycles
 * * are managed in the persistence context.
 * * The EntityManager interface defines the methods for operating the *
 * persistence context and is used for
 * * creating or deleting the perpetuated entity instances,
 * * searching the entity based on the primary key, and for executing
 * * the entity query.
 *
 * * Define the set of entities that can be managed by EntityManager
 * * using the persistence unit.
 * * The persistence unit defines the group of entity classes used
 * * by the application and also defines the mapping between the entity
 * * classes and the database.
 */
public interface EntityManager {

    /**
     * The instance is set to managed and is perpetuated.
     * @param entity instance of the persistent entity
     * @throws EntityExistsException When the entity already exists
     * (When persist method is invoked, EntityExistsException is thrown
     * or EntityExistsException or another PersistenceException
     * is thrown during flush or commit)
     * @throws IllegalArgumentException When the argument is not an entity
     * @throws TransactionRequiredException When the container-managed
     * EntityManager specifying PersistenceContextType.TRANSACTION
     * is invoked when the transaction does not exist
     */
    public void persist(Object entity) ;

    /**
     * The entity status is merged with the current persistence context
     * @param entity Entity
     * @return instance where the status is merged with the persistence context
     * @throws IllegalArgumentException When the instance is not an entity
     * or is a removed entity
     * @throws TransactionRequiredException When the container-managed
     * EntityManager specifying PersistenceContextType.TRANSACTION
     * is invoked when the transaction does not exist.
     */
    public <T> T merge(T entity) ;

    /**
     * entity instance is deleted.
     * @param entity Entity
     * @throws IllegalArgumentException Instance is not an entity
     * or is a detached entity
     */
}
```

5. How to Use JPA with Application Server

```
* @throws TransactionRequiredException When the container-managed
* EntityManager specifying PersistenceContextType.TRANSACTION
* is invoked when the transaction does not exist.
*/
public void remove(Object entity) ;

/**
 * Searches the primary key.
 * @param entityClass Entity class
 * @param primaryKey Primary key
 * @return Instance of the searched entity
 * null when the entity does not exist
 * @throws IllegalArgumentException When the entityClass argument
 * is not an entity type or if the primaryKey argument is not the
 * valid type as the primary key of that entity
 */
public <T> T find(Class<T> entityClass, Object primaryKey) ;

/**
 * The status obtains the delayed fetch instance.
 * When the requested entity does not exist in the database,
 * When the instance status is accessed for the first time
 * EntityNotFoundException is thrown.
 * (when getReference is invoked, the JPA provider is also allowed
 * to throw EntityNotFoundException)
 * If the application does not access the instance while
 * Entity Manager is open, do not expect that the instance status
 * can be accessed during detach
 * @param entityClass Entity class
 * @param primaryKey Primary key
 * @return Instance of the searched entity
 * @throws IllegalArgumentException When the entityClass argument
 * is not an entity type or the primaryKey argument is not a
 * valid type as the primary key of that entity
 * @throws EntityNotFoundException When the entity status cannot
 * be accessed
 */
public <T> T getReference(Class<T> entityClass, Object primaryKey) ;

/**
 * The persistence context status is flushed in the database.
 * @throws TransactionRequiredException When the transaction does
 * not exist
 * @throws PersistenceException When flush fails
 */
public void flush() ;

/**
 * Specifies the flush mode applied to all the objects
 * included in the persistence context.
 * @param flushMode Flush mode
 */
public void setFlushMode(FlushModeType flushMode) ;

/**
 * Obtains the flush mode applied to all the objects
 * included in the persistence context.
 * @return flushMode Flush mode
 */
public FlushModeType getFlushMode() ;

/**
 * Sets the lock mode of the entity objects
 * included in the persistence context.
 * @param entity Entity
 * @param lockMode Lock mode
 * @throws PersistenceException When an unsupported
 * lock invocation is performed
 * @throws IllegalArgumentException When the instance is not an entity
 * or is a detached entity
 * @throws TransactionRequiredException When the transaction
 * does not exist
 */
public void lock(Object entity, LockModeType lockMode) ;
```

```

/**
 * Instance status is refreshed to the database status.
 * If the instance status is changed, the status is
 * overwritten by the database status.
 * @param entity Entity
 * @throws IllegalArgumentException When the argument is not an entity
 * or does not have the managed status
 * @throws TransactionRequiredException When the container-managed
 * EntityManager specifying PersistenceContextType.TRANSACTION
 * is invoked when the transaction does not exist
 * @throws EntityNotFoundException When the entity already
 * does not exist in the database
 */
public void refresh(Object entity) ;

/**
 * Clears the persistence context and all the managed entities
 * are detached.
 * If the entity has changes that were not flushed in the database
 * the entity is not perpetuated.
 */
public void clear() ;

/**
 * Checks if the instance is included in the current
 * persistence context.
 * @param entity Entity
 * @return true if included
 * @throws IllegalArgumentException When the argument is not an entity
 */
public boolean contains(Object entity) ;

/**
 * Creates a Query instance for executing
 * JPQL(Java Persistence Query language) statement
 * @param qlString Java Persistence Query statement
 * @return New Query instance
 * @throws IllegalArgumentException When the query statement is not valid
 */
public Query createQuery(String qlString) ;

/**
 * Creates a Query instance for executing the named query
 * (JPQL or native SQL).
 * @param name Name of the query defined in the Meta data
 * @return New Query instance
 * @throws IllegalArgumentException If the query with the specified
 * name is not defined.
 */
public Query createNamedQuery(String name) ;

/**
 * Creates a Query instance for executing the native SQL statement
 * (update statement and delete statement)
 * @param sqlString Native SQL statement
 * @return New Query instance
 */
public Query createNativeQuery(String sqlString) ;

/**
 * Creates a Query instance for executing the native SQL query.
 * @param sqlString Native SQL statement
 * @param resultClass Class of the instance that forms the
 * return value
 * @return New Query instance
 */
public Query createNativeQuery(String sqlString, Class result-
Class) ;

/**
 * Creates a Query instance for executing the native SQL query.
 * @param sqlString Native SQL statement
 * @param resultSetMapping Result set mapping name

```

5. How to Use JPA with Application Server

```
* @return New Query instance
*/
public Query createNativeQuery(String sqlString, String result-
SetMapping) ;

/**
 * Reports EntityManager that the JTA transaction is active.
 * This method is invoked by the application
 * to associate the application-managed JTA entity manager that was
 * created outside the transaction scope
 * with the current JTA transaction.
 * @throws TransactionRequiredException When the transaction
 * does not exist
 */
public void joinTransaction() ;

/**
 * Returned when the lower provider objects exist.
 * The return value of this method depends on the implementation,
 * but when the container-managed EntityManager is used in
 * Cosminexus Component Container,
 * the EntityManager object of the JPA provider is returned.
 * @return EntityManager object of the JPA provider
 */
public Object getDelegate() ;

/**
 * Closes the application-managed EntityManager.
 * After the close method is invoked, all the methods other than
 * getTransaction and isOpen (returning false) of the Query object
 * obtained from EntityManager instance and EntityManager
 * throw IllegalStateException.
 * If this method is invoked when EntityManager
 * is associated with an active transaction, the persistence context
 * continues to exist until the transaction is concluded.
 * @throws IllegalStateException For the container-managed
 * EntityManager
 */
public void close() ;

/**
 * Returns whether EntityManager is open.
 * @return Returns true until EntityManager closes.
 */
public boolean isOpen() ;

/**
 * Returns the resource level transaction object.
 * Used by the EntityTransaction instance to start and commit
 * multiple transactions serially.
 * @return EntityTransaction instance
 * @throws IllegalStateException When this method is invoked with
 * the JTA entity manager
 */
public EntityTransaction getTransaction() ;
}
```

(2) Notes

- When the transaction scope persistence context is used, the `persist`, `merge`, `remove`, and `refresh` methods must be invoked in the transaction context. If the transaction context does not exist, `javax.persistence.TransactionRequiredException` is thrown.
- The `find` and `getReference` methods can also be invoked outside the transaction context. When the transaction scope persistence context is used, the resulting entity has a detached status. When the extended persistence context is used, the resulting entity has a managed status.
- The `Query` and `EntityTransaction` objects obtained from `EntityManager` can be used while `EntityManager` is open.

- If the argument of the `createQuery` method is not a valid JPQL (Java Persistence Query Language), `IllegalArgumentException` is thrown or the execution of the query fails. If the native query is incorrect and if the definition of the result set is not compatible with the query result, `PersistenceException` is thrown when the query is executed and the execution of the query fails. When possible, `PersistenceException` wraps the database exceptions.
- If a runtime exception is thrown in the method of the `EntityManager` interface, the transaction is rolled back.
- The `close`, `isOpen`, `joinTransaction`, and `getTransaction` methods are used for managing the application-managed `EntityManager`.
- With the EJB specifications, the following methods can invoke the `EntityManager` methods in the Stateless Session Bean:
 - Business method of the business interface or component interface
 - Business method
 - Interceptor method
 - Timeout callback method

The `EntityManager` methods cannot be invoked with the constructor, the `setter` method (including `setSessionContext` method) of the DI, and the life cycle callback method (`PostConstruct` and `PreDestroy`). If the `EntityManager` methods are invoked in the unpermitted locations, the KDJE56538-E message is displayed and `java.lang.IllegalStateException` is thrown.

- With the EJB specifications, the following methods can invoke the `EntityManager` methods in the Stateful Session Bean:
 - Life cycle callback method (`PostConstruct` and `PreDestroy`)
 - Business method of the business interface or component interface
 - Business method
 - Interceptor method
 - `afterBegin` and `beforeCompletion` methods of `SessionSynchronization`

The `EntityManager` methods cannot be invoked with the constructor, the `setter` method of the DI, and `afterCompletion` method of `SessionSynchronization`. If the `EntityManager` methods are invoked in unpermitted locations, the KDJE56538-E message is displayed and `java.lang.IllegalStateException` is thrown.

Note the following points as well when you use the JPA with Application Server:

- `EntityManager` that the application obtains by using `inject`, JNDI lookup, and `EntityManagerFactory` becomes the proxy class of `EntityManager` provided by Application Server and not the `EntityManager` object provided by the JPA provider. If you use the `getDelegate()` method of this proxy class, you can obtain the `EntityManager` object provided by the JPA provider. However, when the container-managed `EntityManager` is used, the container manages the `EntityManager` life cycle, so do not invoke the `close()` method of the `EntityManager` object obtained by using the `getDelegate()` method, from the application.
- If the `getDelegate()` method of `EntityManager` is invoked when the transaction scope persistence context is used and when transaction does not exist, the container cannot close the returned `EntityManager`. In this case, you must invoke `EntityManager.close()` with the application.

5.10.2 javax.persistence.EntityManagerFactory interface

This subsection describes the interface definition and the notes on the `javax.persistence.EntityManagerFactory` interface.

(1) Definition of the interface

```
package javax.persistence;

/**
 * The EntityManagerFactory interface is used
```

```

* to obtain the application-managed EntityManager by the application.
* When the application ends the use of EntityManagerFactory,
* the application must close EntityManagerFactory.
* After EntityManagerFactory is closed,
* all EntityManagers created from EntityManagerFactory are
* treated as closed.
*/
public interface EntityManagerFactory {
/**
 * Creates a new EntityManager.
 * Whenever this method is invoked, a new EntityManager instance
 * is returned.
 * The isOpen method of EntityManager returned by this method returns
 * true.
 * @return New EntityManager
 */
public EntityManager createEntityManager() ;

/**
 * A new EntityManager is created by using the specified property
 * map.
 * Whenever this method is invoked, a new EntityManager instance
 * is returned.
 * The isOpen method of EntityManager returned by this method returns
 * true.
 * @param map Map storing the EntityManager properties
 * @return New EntityManager
 */
public EntityManager createEntityManager(Map map) ;

/**
 * Closes factory and releases all the resources
 * stored in factory.
 * After factory is closed, if method other than isOpen is invoked,
 * IllegalStateException is thrown. The isOpen method returns
 * false.
 * If EntityManagerFactory is closed, all EntityManagers created
 * from factory are also treated as closed.
 */
public void close() ;

/**
 * Returns whether factory is open.
 * @return True until factory is closed
 */
public boolean isOpen() ;
}

```

(2) Notes

- You can include vendor-specific properties of the JPA provider in the map passed to `createEntityManager`. The properties that cannot be recognized by the JPA provider are ignored.
- With the EJB specifications, the following methods can invoke the `EntityManagerFactory` methods in the Stateless Session Bean:
 - Life cycle callback method (`PostConstruct` and `PreDestroy`)
 - Business method of the business interface or component interface
 - Business method
 - Interceptor method
 - Timeout callback method

The `EntityManagerFactory` methods cannot be invoked with the constructor and the setter method (including the `setSessionContext` method) of the DI.

- With the EJB specifications, the following methods can invoke the `EntityManagerFactory` methods in the Stateful Session Bean:
 - Life cycle callback method (`PostConstruct` and `PreDestroy`)

- Business method of business interface or component interface
- Business method
- Interceptor method
- The `afterBegin` and `beforeCompletion` methods of `SessionSynchronization`.

The `EntityManagerFactory` methods cannot be invoked with the constructor, the setter method of the dependency injection, and the `afterCompletion` method of `SessionSynchronization`.

5.11 Notes on setting up applications

This section describes the notes on setting up the applications running on Application Server and using the JPA.

5.11.1 Notes on allocating the entity classes

With Application Server, package the entity classes in the EARs, EJB-JARs, or WARs, at the locations decided in the JPA specifications. Do not add the entity classes in the system class path.

When an entity class is loaded with the application class loader or the Web application class loader, the byte code of the class is converted using the JPA provider in order to implement operations such as Lazy fetch. If the entity class is included in the system class path, the byte code conversion does not work because the entity class is loaded with the system class loader. Therefore, the JPA provider does not operate properly.

5.11.2 Reference scope of the persistence unit name

The components, such as the EJBs and servlets, included in the application, reference the used persistence unit by specifying the persistence unit name in the `unitName` attribute of `@PersistenceUnit` and `@PersistenceContext` and in the `<persistence-unit-name>` tag under the `<persistence-context-ref>` tag or under the `<persistence-unit-ref>` tag defined in the DD. However, the persistence unit scope that can be referenced from each component is as follows:

- The persistence units defined in the EJB-JARs or WARs can be referenced from the components included in the EJB-JARs or WARs.
- The persistence unit defined in the EAR file can be referenced from all the components included in the EAR file.

If the name of the persistence unit defined in the EAR file and the name of the persistence unit defined in the EJB-JARs or WARs are duplicated, from the components in the EJB-JAR or WAR, the persistence units defined with a narrower scope are given priority. For example, if persistence units with the same name are defined in the EAR and WAR files, from the components included in the WAR file, the persistence unit defined in the WAR file is visible on priority. The persistence unit existing in the EAR file is not visible.

(1) Persistence unit used when the persistence unit name is omitted

If you omit the persistence unit name referenced by the components when the JPA is used with Application Server, the persistence unit used is determined with the following rules:

- If only one persistence unit is defined in the EJB-JAR or WAR that contains components, that persistence unit is used.
- If persistence unit is not defined in the EJB-JAR or WAR that contains components and only one persistence unit is defined in the EAR, the persistence unit in the EAR is used.

Note that if the following conditions are fulfilled, one persistence unit cannot be identified, so the persistence unit name cannot be omitted:

- When two or more persistence units are defined in the EJB-JAR or WAR that contains components
- If persistence unit is not defined in the EJB-JAR or WAR that contains components and two or more persistence units are defined in the EAR

(2) Explicitly referencing the EAR-level persistence unit using '#' syntax

If the name of the persistence unit defined in the EAR and the name of the persistence unit defined in the EJB-JAR or WAR are duplicated, from the components in the EJB-JAR or WAR, the persistence units defined with a narrower scope are visible on priority. However, by using '#' syntax in the reference name of the persistence unit, you can explicitly reference the persistence unit defined in the EAR. When you use '#' syntax, specify the persistence unit name as follows:

Relative-path-from-the-EJB-JAR-or-WAR-containing-components-to-the-persistence-unit-root # *Name-of-the-persistence-unit*

For example, when the persistence unit `myPersistenceUnit` included in `lib/persistenceUnitRoot.jar` of the EAR file is referenced from the EJB included in `ejbs/myEjbs.jar` of the EAR file, the referenced persistence unit name is `../lib/persistenceUnitRoot.jar#myPersistenceUnit`.

5.11.3 Items checked when the application is deployed

The following items are checked when an application using the JPA is deployed on the J2EE server:

- Checking the persistence unit definitions
- Checking the EntityManager and EntityManagerFactory references

The following is a description of the checks:

(1) Checking of the persistence unit definition

The following points describe the contents checked in the persistence unit definition:

(a) Validation of persistence.xml

Validates whether `persistence.xml` included in the application is in accordance with the `persistence_1_0.xsd` schema. If an error is found during the validation, the error message KDJE56526-E is output and the deployment is cancelled.

(b) Checking of the persistence unit name

Checks whether the persistence unit name is null. If the persistence unit name is null, the error message KDJE56505-E is output and the deployment is cancelled.

Also, checks whether persistence units with duplicated names are defined in the application EAR files or in one EJB-JAR or WAR. If persistence units with duplicated names are defined, a warning message KDJE56500-W is output and the deployment continues. Note that in this case, only one persistence unit is actually deployed.

(c) Checking whether the data source referenced by the persistence unit exists

The contents checked differ depending on the transaction type of the persistence unit.

• When the transaction type of the persistence unit is the JTA

The following table lists the contents checked and the operations to be performed when an error occurs during the check.

Table 5-10: Checked contents and operations to be performed when an error occurs during the check (for the JTA transaction type)

Checked contents	Operations when an error occurs during the check
The JTA data source (the data source specified in the <code><jta-data-source></code> tag of <code>persistence.xml</code> or the default value specified in the system properties) used in the persistence unit is specified.	The error message KDJE56527-E is output and the deployment is cancelled.
The resource adapter that provides the JTA data source exists.	The error message KDJE56529-E is output and the deployment is cancelled.
The resource adapter that provides the JTA data source is running.	
The transaction support level of the resource adapter is <code>LocalTransaction</code> or <code>XATransaction</code> .	The error message KDJE56531-E is output and the deployment is cancelled.
The connection factory interface of the resource adapter is <code>javax.sql.DataSource</code> .	The error message KDJE56533-E is output and the deployment is cancelled.

• When the transaction type of the persistence unit is RESOURCE_LOCAL

The following table lists the contents checked and the operations to be performed when an error occurs during the check.

Table 5-11: Checked contents and operations to be performed when an error occurs during the check (for the RESOURCE_LOCAL transaction type)

Checked contents	Operations when an error occurs during the check
The non-JTA data source (the data source specified in the <code><non-jta-data-source></code> tag of <code>persistence.xml</code> or the default value specified in the system properties) used in the persistence unit is specified.	The error message KDJE56528-E is output and the deployment is cancelled.
The resource adapter that provides the non-JTA data source exists.	The error message KDJE56530-E is output and the deployment is cancelled.
The resource adapter that provides the non-JTA data source is running.	
The transaction support level of the resource adapter is <code>NoTransaction</code> .	The error message KDJE56532-E is output and the deployment is cancelled.
The connection factory interface of the resource adapter is <code>javax.sql.DataSource</code> .	The error message KDJE56533-E is output and the deployment is cancelled.

(d) Checking the provider class specified in the persistence unit

Checks whether the JPA provider class (class specified in the `<provider>` tag of `persistence.xml`) used by the persistence unit can be loaded from the application. If the loading of the class fails, the error message KDJE56503-E is output and the deployment is cancelled.

(e) Checking whether the JAR file referenced by the persistence unit exists

Checks whether the JAR file (JAR file specified in the `<jar-file>` tag of `persistence.xml`) referenced by the persistence unit exists. If the JAR file does not exist, the error message KDJE56506-E is output and the deployment is cancelled.

(f) Checking the contents defined in the persistence unit using the JPA provider

The JPA provider generates the persistence unit from the contents of `persistence.xml` parsed by the JPA container. If the persistence unit definition has a problem and if the JPA provider returns an error, the error message (KDJE56539-E) is displayed and the deployment is cancelled.

(2) Checking the EntityManager and EntityManagerFactory references

The following points describe the contents checked in the EntityManager and EntityManagerFactory references:

(a) Checking the existence of the persistence unit

In the EntityManager and EntityManagerFactory references defined in the EJB and Web components, check whether the specified persistence unit name is an actually referable persistence unit name. Also, if the persistence unit name is omitted, check whether the persistence unit to be used can be identified. If an error is found in this check, the error message KDJE56501-E is output and the deployment is cancelled.

(b) Checking the transaction type for the container-managed EntityManager

Check whether the transaction type of the persistence unit is the JTA when the container-managed EntityManager is used. If an error occurs in this check, the error message KDJE56534-E is output and the deployment is cancelled.

(c) Checking if the extended persistence context is used from outside the Stateful Session Bean

When `EXTENDED` is specified as the persistence context type with the EntityManager references, check whether the location where the references are defined is the Stateful Session Bean. If the references are defined in a location other than the Stateful Session Beans, the error message KDJE56535-E is output and the deployment is cancelled.

5.11.4 Notes on using the JPA with Application Server

The `EntityManager` type that the application can obtain is the proxy class of `EntityManager` provided by Application Server and not the `EntityManager` object provided by the JPA provider.

The `EntityManager` object can be obtained by using injection, JNDI lookup, or `EntityManagerFactory`.

Note that to use injection for obtaining `EntityManager`, you set up `javax.persistence.EntityManager` as the field or method argument type that injects `EntityManager`.

Also, you cannot cast `EntityManager` obtained with injection, JNDI lookup, or `EntityManagerFactory` in the `EntityManager` implementation class of the JPA provider.

If you need to obtain the `EntityManager` object of the JPA provider, use the `getDelegate` method of the `EntityManager` proxy object obtained with injection, JNDI lookup, or `EntityManagerFactory`.

5.11.5 Notes when the Cosminexus JPA functionality is not used

With the Cosminexus JPA functionality, when `persistence.xml` is included in an application, `persistence.xml` is read by default regardless of whether the Cosminexus JPA functionality is used. If `persistence.xml` cannot be interpreted with Application Server, the application fails to start.

To avoid this situation, when `persistence.xml` is included in the application and you do not want to use the Cosminexus JPA functionality, specify `true` in the `ejbserver.jpa.disable` property of the J2EE server.

As a result, `persistence.xml` is no longer read with Application Server.

Also, regardless of the value specified in the `ejbserver.jpa.disable` property, the JPA version that can be used with Application Server is JPA 1.0.

6

Cosminexus JPA Provider

This chapter gives an overview of the Cosminexus JPA Provider functionality, describes the application implementation methods, and the execution environment settings.

6.1 Organization of this chapter

Cosminexus JPA Provider is a JPA provider that implements API functions and the internal processing determined in the JPA specifications and is provided by Application Server provided to the user in the library format.

This chapter describes the Cosminexus JPA Provider functionality. The following table describes the organization of this chapter.

Table 6-1: Organization of this chapter (Cosminexus JPA Provider functionality)

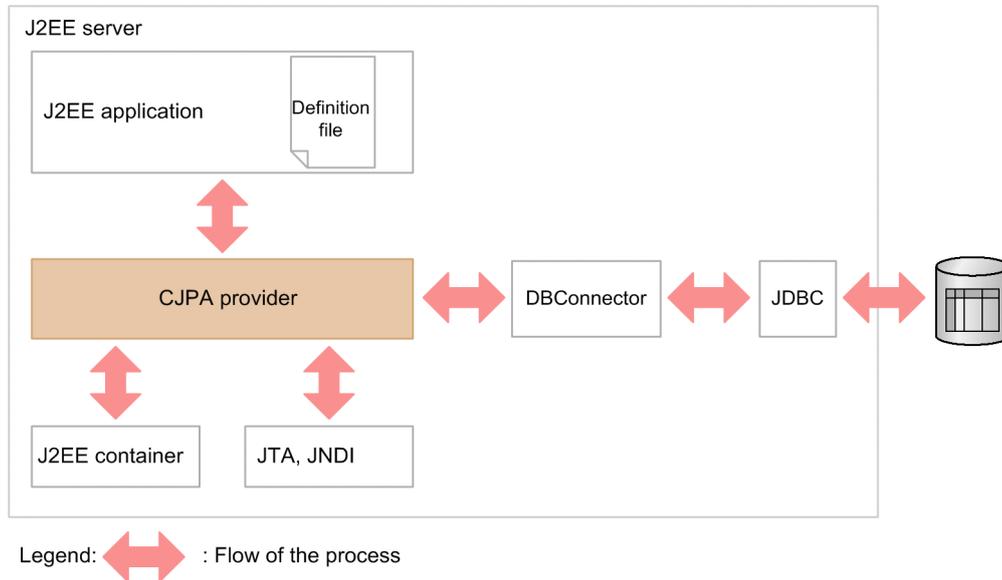
Category	Title	Reference location
Explanation	Cosminexus JPA Provider	6.2
	Updating a database using entities	6.3
	Entity operations by <code>EntityManager</code>	6.4
	Defining the mapping information between the database and Java objects	6.5
	Entity relationships	6.6
	Cache functionality of the entity objects	6.7
	Auto-numbering of the primary key values	6.8
	Database operations based on the query language	6.9
	Optimistic lock	6.10
	Pessimistic lock in JPQL	6.11
Implementation	Creating an entity class	6.12
	Procedure for inheriting an entity class	6.13
	Procedure for using <code>EntityManager</code> and <code>EntityManagerFactory</code>	6.14
	Procedure for specifying the callback method	6.15
	Procedure for referencing and updating the database with the query language	6.16
	JPQL coding method	6.17
	Defining persistence.xml	6.18
Settings	Settings in the execution environment	6.19

Note: The functionality-specific explanation is not available for *Operations*.

6.2 Cosminexus JPA Provider

Cosminexus JPA Provider is the JPA provider provided by Application Server. The following figure shows the positioning of Cosminexus JPA Provider.

Figure 6-1: Positioning of Cosminexus JPA Provider



Cosminexus JPA Provider is the J2EE server functionality and uses JTA and JNDI to operate the database. The following subsections give an overview of the Cosminexus JPA Provider functionality.

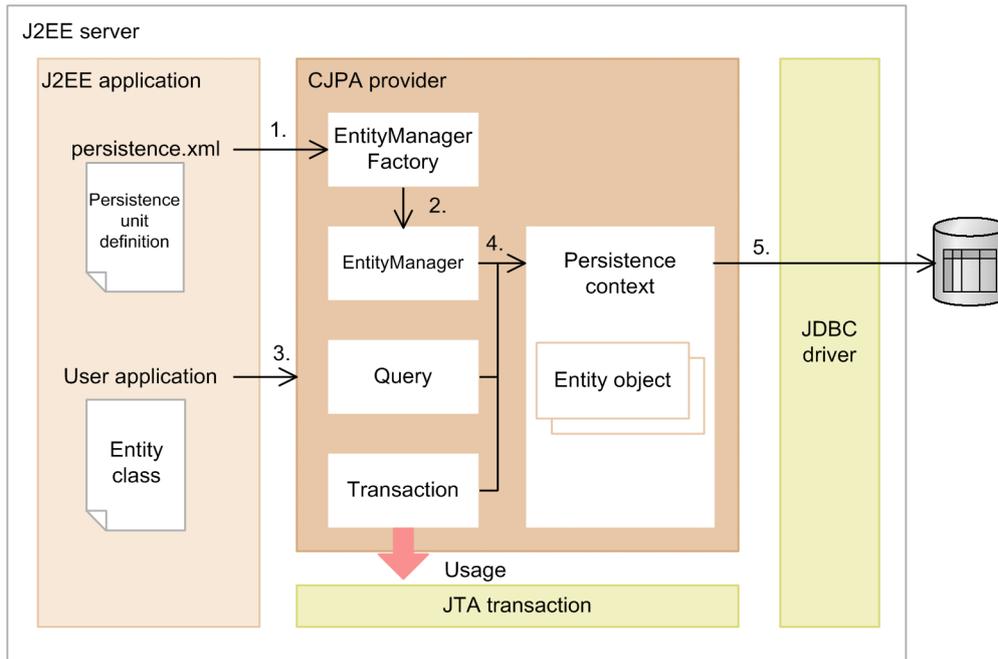
6.2.1 Processing in Cosminexus JPA Provider

Cosminexus JPA Provider provides the following functionality:

- Mapping of the entity objects and database
- Managing the entity objects
- Data operations using JPQL
- Accessing the database through JDBC

To use JPA, you operate the interfaces provided by Cosminexus JPA Provider by using the user-created applications. The following figure shows the processing executed in Cosminexus JPA Provider.

Figure 6-2: Processing executed in Cosminexus JPA Provider



A description of the figure is as follows:

1. Cosminexus JPA Provider generates `EntityManagerFactory` from the persistence unit information defined in `persistence.xml`.
A persistence unit is a logical group that collects information such as the classes to be mapped, O/R mapping information, and the data sources. For details on a persistence unit, see [5.4.4 Persistence unit](#).
2. `EntityManager` is generated from the generated `EntityManagerFactory`.
3. `EntityManager`, query, or transaction is operated from the user application.
4. The entity object is operated through the persistence context from `EntityManager`, query, or transaction.
5. The changes in the entity object are applied to the database through the JDBC driver.

Note that there are two types of `EntityManager`:

- Container-managed `EntityManager`
`EntityManager` managed by the J2EE container
- Application-managed `EntityManager`
`EntityManager` managed by an application

For details on the types of `EntityManager`, see [5.5.1 EntityManager and persistence context](#).

Furthermore, a transaction manages operations such as the transaction commit and rollback. With Cosminexus JPA Provider, you can use the resource local transaction as the transaction. To integrate with JTA transactions, use the functionality provided by the J2EE container.

6.2.2 Functionality provided by Cosminexus JPA Provider

The following functionality is provided by Cosminexus JPA Provider.

Table 6-2: Functionality provided by Cosminexus JPA Provider

Functionality	Overview of functionality	Standard/Extended #	Reference location
Updating a database using entities	Updates the data in the database using the entities generated from the user applications.	Standard	6.3 6.4
Defining the mapping information between the database and Java objects	The mapping information between the database and Java objects can be defined with annotations or the O/R mapping file.	Standard	6.5
Entity relationship settings	Sets up the relationship between the database tables by using entities.	Standard	6.6
Entity object cache	This functionality stores the content of the entity object of <code>EntityManager</code> in the memory. When an entity object with the same content is invoked, the entity object in the memory is used.	Extended	6.7
Auto-numbering of primary key	Cosminexus JPA Provider automatically stores the primary key when the entity object is used to insert the data in the database. Even if the user does not specify the primary key, a unique value is stored.	Standard	6.8
Database operations based on the query language	The database can be operated using the query language. With Cosminexus JPA Provider, you can use JPQL or SQL as the query language.	Standard	6.9
Optimistic lock	This is one of the database locking methods. The data is updated after confirming that the data in the database is not being updated by another transaction. An optimistic lock does not lock the data, so a deadlock does not occur.	Standard	6.10
Pessimistic lock in JPQL	This is one of the database locking methods. A dedicated lock is set for the record so that the record is not updated by another transaction. A pessimistic lock is executed only when JPQL is used.	Extended	6.11

Indicates whether the functionality is based on the JPA 1.0 specifications.

Standard: Indicates that the functionality is based on the JPA 1.0 specifications.

Extended: Indicates that the functionality is unique to Cosminexus JPA Provider.

For details on the functionality, see the sections mentioned in the *Reference location* column in the above table.

Also, you can collect the PRF trace in Cosminexus JPA Provider. For details on the PRF trace collection points, see 8. *Trace Collection Points and PRF Trace Collection Levels of the Trace Based Performance Analysis* in the *uCosminexus Application Server Maintenance and Migration Guide*.

6.2.3 Preconditions for using Cosminexus JPA Provider

This subsection describes the preconditions for using Cosminexus JPA Provider.

(1) Available components

The components that use Cosminexus JPA Provider are EJB 3.0 and later in the case of EJBs and Servlet 2.5 and later in the case of Web components. For details on the available components, see 5.3.2 *Available components*.

(2) Connectable databases

You can connect to the following databases when you use Cosminexus JPA Provider:

- Oracle10g
- Oracle11g

- HiRDB Version 8
- HiRDB Version 9

(3) Available DB Connectors

Cosminexus JPA Provider uses DB Connector to update the database. You can use DB Connectors listed in the following table with Cosminexus JPA Provider.

Table 6-3: DB Connectors available with Cosminexus JPA Provider

Database used	Transaction type	Available DB Connector
Oracle	LocalTransaction	DBConnector_Oracle_CP.rar
	NoTransaction	
	XATransaction	DBConnector_Oracle_XA.rar
Oracle RAC	LocalTransaction	DBConnector_Oracle_CP.rar
	NoTransaction	DBConnector_CP_ClusterPool_Root.rar DBConnector_Oracle_CP_ClusterPool_Member.rar
	XATransaction	
HiRDB	LocalTransaction	DBConnector_HiRDB_Type4_CP.rar
	NoTransaction	
	XATransaction	DBConnector_HiRDB_Type4_XA.rar

(4) Environment that cannot be used

You cannot use Cosminexus JPA Provider in the following environment:

- Execution environment for the batch applications
- Execution environment for the EJB client applications

(5) Using the method cancellation functionality

With Cosminexus JPA Provider, a unique binary code is embedded in the accessor method for OneToOne and ManyToOne LAZY fetch. As a result, the accessor method is subject to method cancellation. Therefore, when LAZY fetch is specified for the OneToOne relationship or ManyToOne relationship, the entity class, embeddable class, and mapped super-class must be registered in the protected area.

Note that if the classes are not registered in the protected area, method cancellation might occur on the embedded binary code. The operations when the classes are not registered in the protected area might not function properly.

Whether the classes are registered or not registered in the protected area, the following stack trace is output due to method timeout. However, if the classes are registered in the protected area, method cancellation does not occur.

In the following example, the embedded `EntityClass1._toplink_getmappingClass2` is invoked by extending the `EntityClass1.getMappingClass2` method invoked by the user. Consequently, a method timeout occurs, but method cancellation does not occur.

```

message-id      message(LANG=ja)
KDJE52703-W      A timeout occurred while the user program was executing. (threadID =
23794987, rootAPInfo = 10.209.11.124/5964/0x4828eb62000128e0, application = JPA_JavaEE_TP,
bean = TestBean, method = doTest)

jpa.test.annotation.onetoone.entity.EntityClass1._toplink_getmappingClass2(EntityClass1.java
)
    locals:
        (jpa.test.annotation.onetoone.entity.EntityClass1) this =
<0x11e35878> (jpa.test.annotation.onetoone.entity.EntityClass1)
        at
jpa.test.annotation.onetoone.entity.EntityClass1.getMappingClass2(EntityClass1.java:34)
    locals:
        (jpa.test.annotation.onetoone.entity.EntityClass1) this =
<0x11e35878> (jpa.test.annotation.onetoone.entity.EntityClass1)

```

For details on registering the classes in the protected area, see *2.6 criticalList.cfg (Protected area list file)* in the *uCosminexus Application Server Definition Reference Guide*.

(6) Using the annotation reference control functionality

When you use Cosminexus JPA Provider, you cannot use the annotation reference control functionality. If the annotation reference control functionality is enabled, you cannot define the persistence context and persistence unit references.

6.2.4 Estimating the number of DB Connector connections

This subsection describes the estimation of the DB Connector resources required for using Cosminexus JPA Provider.

With Cosminexus JPA Provider, you obtain the DB Connector connections. The formula for estimating the number of connections used by Cosminexus JPA Provider is as follows:

Formula for estimating the number of connections

Number of connections used by the Cosminexus JPA Provider applications

= Number of concurrent executions of applications using the JPA functionality #

In the applications using the JPA functionality, when `EntityManager` of multiple persistence units is used or when the transactions used by `EntityManager` for invoking the business methods are different, a connection is required for each `EntityManager`.

! Important note

When the user acquires a connection separately from the JPA functionality, the number of connections can be reduced by using the connection sharing functionality. For details on connection sharing, see *3.14.3 Connection sharing or association*.

6.3 Updating a database using entities

With Cosminexus JPA Provider, you can update a database using entities.

To use the JPA, the user must create an entity class for handling the database tables as Java objects. If you use the entity class, the data in the database is operated through Cosminexus JPA Provider. At this time, you operate the database through the `EntityManager` interface provided by Cosminexus JPA Provider.

The database operations using the entity class are as follows:

1. Generating the entity class instance (entity object).
2. Passing the entity object to the argument of the `EntityManager` interface to operate the database.

To perform these operations, an entity class must be created.

`EntityManager` is an object used for operating the entities and controlling the states. The entity operations include the `persist` operation, `remove` operation, `merge` operation, `flush` operation, and `refresh` operation. For details on these operations, see *6.4.1 Transition of entity states*.

6.4 Entity operations by EntityManager

The state of the entity is controlled by `EntityManager` provided with Cosminexus JPA Provider. This section describes the entity operations in `EntityManager`.

6.4.1 Transition of entity states

An entity has a state. When you perform operations for an entity, the state of the entity changes. This subsection describes the types of entity states, the operations executed for the entity, and the entity operations and state transition.

(1) Types of entity states

An entity has four types of states, namely `new`, `managed`, `detached`, and `removed`. The entity state is changed by using the `EntityManager` method to operate an entity.

The following table lists and describes each entity state.

Table 6-4: Entity state

State of entity instances	Explanation
<code>new</code>	A state in which the entity is newly generated. A newly generated entity instance does not have a persistence identity, and is, therefore, not associated with the persistence context.
<code>managed</code>	A state in which the entity has a persistence identity associated with the persistence context and is managed with the persistence context.
<code>detached</code>	A state in which the entity has a persistence identity that is not associated with the persistence context.
<code>removed</code>	A state in which the entity has a persistence identity and is associated with the persistence context. Also, includes the state in which the entity instance is scheduled for deletion from the database.

(2) Operations for the entities

If the user searches the database records, Cosminexus JPA Provider stores the obtained data in the entity fields. Also, when the user updates the database contents, the entity state is applied to the database by changing the state of the entity registered in the persistence context and by committing the transaction. Thus, by executing operations for the entity, the database information is updated.

The following table lists and describes the types of operations for the entities.

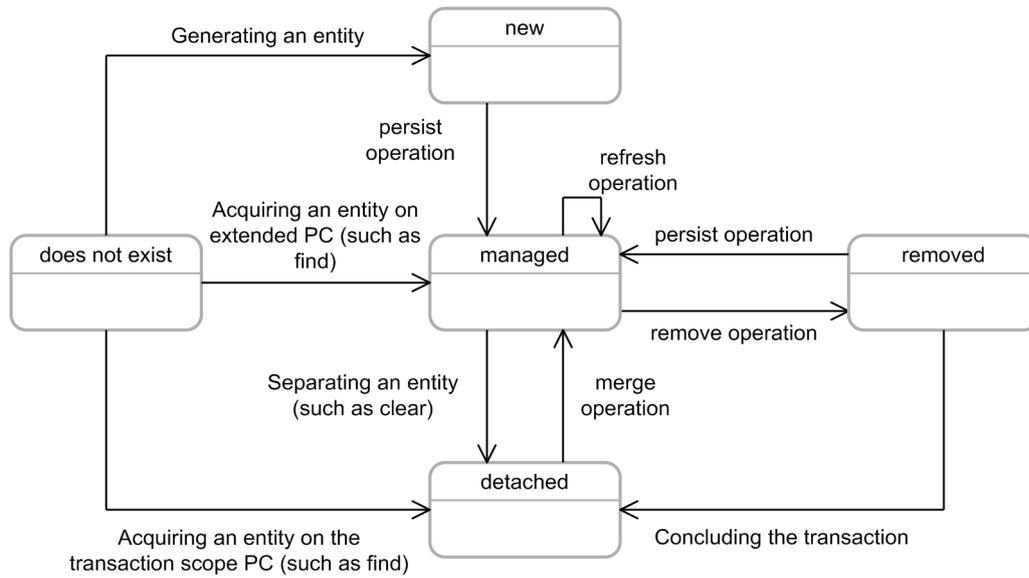
Table 6-5: Types of operations for the entities

Operations	Explanation
<code>flush</code>	This operation applies the content of the entity object to the database.
<code>merge</code>	In this operation, the entity object that was not managed by <code>EntityManager</code> is managed by <code>EntityManager</code> .
<code>persist</code>	This operation manages and perpetuates the entity object.
<code>refresh</code>	This operation applies the database content to the entity object.
<code>remove</code>	This operation sets the entity object to the state of scheduled deletion.

(3) Operations and state transition for the entities

The following figure shows the operations and the state transition for the entity instances.

Figure 6-3: Operations and state transition for the entity instances



The following table describes the operations and state transition for the entities.

Table 6-6: Transition of entity states

Operations	State			
	new	managed	detached	removed
persist	managed ^{#1}	managed	managed ^{#1}	managed
remove	new	removed	Exception ^{#2, #3}	removed
merge	<ul style="list-style-type: none"> Copy source new Copy destination managed 	<ul style="list-style-type: none"> Copy source managed Copy destination managed 	<ul style="list-style-type: none"> Copy source detached Copy destination managed 	Exception ^{#3, #4}
refresh	Exception ^{#2, #4}	managed ^{#5}	Exception ^{#2, #4}	Exception ^{#2, #4}
commit	--	#6	--	detached
rollback	--	detached	--	detached
flush	--	managed	--	detached
clear	--	detached	--	detached

Legend:

--: Not applicable

#1 If an exception occurs in the `persist` operation, the state enters the original state without being changed.

#2 The exception that occurs is `IllegalArgumentException`.

#3 If the corresponding line does not exist in the database, the operation is ignored.

#4 If an exception occurs, the state does not change and remains as the original.

#5 If the corresponding line does not exist in the database, `EntityNotFoundException` occurs.

#6 For a persistence context in the transaction scope, the state is `detached`. For an extended persistence context, the state is `managed`.

Furthermore, the operations when `persist`, `remove`, `merge`, and `refresh` are executed outside the transaction vary according to the type of the persistence context.

- For a persistence context in the transaction scope
TransactionRequiredException occurs.
- For an extended persistence context
The state changes and the states are applied to the database when the next transaction is concluded.

(4) Propagation of operations to the entities

When the entities have a relationship and if you specify the `cascade` attribute of the annotation indicating the relationship, the operations for the entity are propagated to the related entities. You can specify the values listed in the following table in the `cascade` attribute.

Table 6-7: Types of cascade attributes

Types of cascade attributes	Explanation
<code>CascadeType.ALL</code>	The <code>persist</code> , <code>remove</code> , <code>merge</code> , and <code>refresh</code> operations are propagated to the relation destination.
<code>CascadeType.PERSIST</code>	The <code>persist</code> operation is propagated to the relation destination.
<code>CascadeType.REMOVE</code>	The <code>remove</code> operation is propagated to the relation destination.
<code>CascadeType.MERGE</code>	The <code>merge</code> operation is propagated to the relation destination.
<code>CascadeType.REFRESH</code>	The <code>refresh</code> operation is propagated to the relation destination.

6.4.2 persist operation for the entities

To execute the `persist` operation for the entities, invoke the `persist` method of `EntityManager`. If the `persist` method of `EntityManager` is invoked and the `persist` processing is cascaded, the entity is managed in the persistence and persistence context of the database.

The following table describes the state of the entity after the `persist` operation, for each entity state.

Table 6-8: State of the entities in the persist operation

State of the entity	State of the entity after the persist operation
<code>new</code> [#]	The state changes to <code>managed</code> . The entity changed to <code>managed</code> is added into the database during or before transaction commit or is added into the database as a result of the execution of the <code>flush</code> operation.
<code>managed</code>	The <code>persist</code> operation is ignored. However, if <code>PERSIST</code> or <code>ALL</code> is specified in the <code>cascade</code> attribute for the relationship from one entity to another entity, the <code>persist</code> operation is propagated to the entities referenced by this entity.
<code>detached</code> [#]	If the line corresponding to the entity does not exist in the database, the state changes to <code>managed</code> . If the corresponding line exists, <code>EntityExistsException</code> occurs.
<code>removed</code>	The state changes to <code>managed</code> .

[#] For Cosminexus JPA Provider, when the state of an entity is `new` or `detached`, the results of the transition of entity states differ according to whether the data corresponding to the entity exists on the database. Note the following for Cosminexus JPA Provider:

- If an entity that is not perpetuated in the database is specified in the argument, the state changes to `managed`.
- If a line duplicating with the entity passed in an argument exists on the database and if that entity is managed with the persistence context, `EntityExistsException` occurs during the `persist` processing.
- If a line duplicating with the entity passed in an argument exists on the database, but if the entity is not managed with the persistence context, `EntityExistsException` or another `PersistenceException` occurs during `flush` and `commit`.

! Important note

The entity is registered in the persistence context when the entity is perpetuated in the database and when the entity information is read from the database. Note that after the entity state is changed to `managed`, if the persistence processing is rolled back, the entity is not managed with the persistence context.

6.4.3 remove operation for the entities

To execute the `remove` operation for the entities, invoke the `remove` method of `EntityManager`. If the `remove` method of `EntityManager` is invoked and the `remove` processing is cascaded, the state of the entity becomes `removed`. A `removed` entity is deleted from the database by the transaction commit processing.

The following table describes the state of the entity after the `remove` operation, for each entity state.

Table 6-9: State of entity in the `remove` operation

State of the entity	Result of state transition
<code>new</code>	The <code>remove</code> operation is ignored. However, if <code>REMOVE</code> or <code>ALL</code> is specified in the <code>cascade</code> attribute for the relationship from one entity to another entity, the <code>remove</code> operation is propagated to the entities referenced by this entity.
<code>managed</code>	The state changes to <code>removed</code> . If <code>REMOVE</code> or <code>ALL</code> is specified in the <code>cascade</code> attribute for the relationship from one entity to another entity, the <code>remove</code> operation is propagated to the entities referenced by this entity.
<code>detached</code>	The state changes as follows: <ul style="list-style-type: none"> • If the line corresponding to the <code>detached</code> entity passed in the argument exists in the database, <code>IllegalArgumentException</code> is thrown during the <code>remove</code> operation. • For Cosminexus JPA Provider, if the corresponding line does not exist in the database, the <code>remove</code> operation is ignored. However, if <code>REMOVE</code> or <code>ALL</code> is specified in the <code>cascade</code> attribute for the relationship with other entities, the <code>remove</code> operation is propagated to the entities referenced by this entity.
<code>removed</code>	The <code>remove</code> operation is ignored. The operation is also not propagated to other entities.

Note 1: A `removed` entity is deleted from the database as a result of the execution of operations during transaction commit, before transaction commit, or during `flush` operations.

Note 2: After the entity is deleted, the entity contents change to the contents immediately after the `remove` processing was invoked, however, excluding the content immediately after the entity was generated.

6.4.4 Obtaining the entities from the database

By invoking the `find` method or `getReference` method of `EntityManager`, you can obtain the entities corresponding to the primary key specified in the argument, from the database.

The entities returned by the `find` method or `getReference` method of `EntityManager` use the persistence context of the transaction scope. Therefore, when the method is invoked within the transaction, the state changes to `managed`. Also, if the `find` method or `getReference` method is invoked outside the transaction, the state changes to `detached`.

The following table describes the state of entities obtained with the `find` operation or `getReference` operation.

Table 6-10: State of entities obtained with the `find` operation or `getReference` operation

Persistence context	State of the obtained entity
Persistence context of the transaction scope (outside the transaction)	<code>detached</code>
Persistence context of the transaction scope (within the transaction)	<code>managed</code>
Extended persistence context (outside the transaction)	<code>managed</code>
Extended persistence context (within the transaction)	<code>managed</code>

Note that for Cosminexus JPA Provider, the following points differ from the JPA specifications in the `find` operation or `getReference` operation. Note that apart from these differences, there are no other functionality differences with the JPA specifications.

- With the JPA specifications, in the `getReference` method of `EntityManager`, you can return `Proxy` to the entity corresponding to the primary key given in the argument and not in the actual instance. However, with

Cosminexus JPA Provider, Lazy loading is not supported for `@Basic`. Therefore, with Cosminexus JPA Provider, entity instance is returned as the return value of `getReference` instead of `Proxy`.

For details on the supported range of Lazy loading for a relationship, see 6.4.5(2) *Reading the entity information from the database*.

- With Cosminexus JPA Provider, if an entity that does not exist between the `find` method and `getReference` method is specified in the argument, a null value is returned with the `find` method. Also, `EntityNotFoundException` is thrown with the `getReference` method.

6.4.5 Synchronization with the database

When the transaction commit or `flush` method is executed, the state of the entity is written to the database. On the other hand, unless `refresh` is invoked explicitly, the state of the entity loaded in the memory is not refreshed.

This subsection describes writing of the entity information to the database and reading of the entity information from the database.

(1) Writing the entity information to the database

This section describes the transition of entity states in the `flush` operation or transaction commit. The following table describes the state transition results for each entity A state.

Table 6-11: State transition of entity instances in the flush operation or transaction commit

State of entity A	Result of state transition
new	The <code>flush</code> operation is ignored.
managed	Entity A is synchronized with the database.
removed	Entity A is deleted from the database.
detached	The <code>flush</code> operation is ignored.

Also, if entity A with the managed state has a relationship to entity B, the `persist` operation is cascaded according to the conditions described in the following table by extending the `flush` processing.

Table 6-12: Cascading of the flush processing and persist operation in commit for the related entity B

Specification of the cascade attribute of the relationship to entity B	State of entity B	Results
PERSIST or ALL is specified	--	The <code>persist</code> operation is cascaded to entity B.
PERSIST or ALL is not specified	new	<ul style="list-style-type: none"> • In the <code>flush</code> operation <code>IllegalStateException</code> occurs and the transaction is marked for rollback. • In the commit processing <code>IllegalStateException</code> occurs and the transaction commit operation fails.
	managed	Entity B is synchronized with the database.
	removed	<ul style="list-style-type: none"> • In the <code>flush</code> operation <code>IllegalStateException</code> occurs and the transaction is marked for rollback. • In the commit processing <code>IllegalStateException</code> occurs and the transaction commit operation fails.
	detached	<ul style="list-style-type: none"> • When entity A is the owner of the relationship The change in the relationship is synchronized with the database.

Specification of the cascade attribute of the relationship to entity B	State of entity B	Results
PERSIST or ALL is not specified	detached	<ul style="list-style-type: none"> When entity B is the owner of the relationship An exception occurs. With Cosminexus JPA Provider, the operations in this case are not supported.

Legend:

--: Not applicable

Note that if the `flush` method is invoked outside the transaction, `TransactionRequiredException` occurs.

Tip

Persistence relation for relationship and database

- A managed entity having a bi-directional relationship is perpetuated on the basis of the reference stored in the owner side of the relationship. The application developer must create an application so that when changes occur, the entity is stored in the owner side and the non-owner side respectively without conflicting with the state of the entity on the memory.
- For unidirectional `OneToMany` and `OneToOne`, it is the developer's responsibility to guarantee that the relation of the relationships defined in the entity and the relation between the database tables is matching.

(2) Reading the entity information from the database

If the `refresh` method of `EntityManager` is invoked, the changes performed in the entity until then are destroyed and the state of the entity is overwritten by the database contents. At this time, if the corresponding line does not exist in the database, `EntityNotFoundException` occurs.

If you invoke the `refresh` method of `EntityManager` when the state of entity is not managed, `IllegalArgumentException` occurs.

(a) Timing when the entity information is read from the database

The entity is read from the database when the `refresh` method or `find` method is executed or when a query is issued. The related entities can also be read at this time. This is called the *fetch strategy*. Specify the fetch strategy in the `fetch` attribute of each relationship. Specify one of the following types in the `fetch` attribute:

- **FetchType.EAGER**

When the entity information is read from the database, the related field and entity information is read.

- **FetchType.LAZY**

When the field or relation destination is accessed for the first time, the entity is read from the database. This is called *Lazy loading*.

If you specify `FetchType.EAGER`, the related field and entity information is read every time the entity information is read from the database. Therefore, specify `FetchType.LAZY` to prevent the obtaining of unnecessary relation destination entities.

The following table describes the supported range of the `fetch` attribute in Cosminexus JPA Provider.

Table 6-13: Supported range of the `fetch` attribute for each relationship

Relationship annotation	Supported range
@ManyToOne	The default value is <code>FetchType.LAZY</code> .
@OneToMany	The default value is <code>FetchType.LAZY</code> .
@OneToOne	The default value is <code>FetchType.EAGER</code> . Note that if <code>LAZY</code> is specified, see (b).
@ManyToOne	The default value is <code>FetchType.EAGER</code> . Note that if <code>LAZY</code> is specified, see (b).
@Basic	The <code>fetch</code> attribute is ignored. The default <code>FetchType.EAGER</code> is always applied.

(b) LAZY fetch in @OneToOne and @ManyToOne

If you specify *LAZY* in the fetch strategy for a `@OneToOne` and `@ManyToOne` relationship, when the entity class is loaded, the binary code is embedded in the `getter` method for the field specifying *LAZY*.

If you invoke the `getter` method, the relation destination entity is obtained from the database through the processing of the embedded binary code. Because the binary code is embedded in the `getter` method, the `getter` method used for accessing the field that forms the target of relation must be set up. Furthermore, `@OneToOne` or `@ManyToOne` must be specified in that `getter` method.

! Important note

The `getter` method determines the type of the relation destination entity from the `targetEntity` type of the relationship. The type of class specified in `targetEntity` must be castable in the field or property type.

6.4.6 Separate and merge operations of an entity from the persistence context

An entity separated from the persistence context is called a *detached* entity. The entity state becomes *detached* at the following times:

- When a transaction in the persistence context of the transaction scope is committed
- When a transaction is rolled back
- When the persistence context is cleared (when `EntityManager.clear()` is invoked)
- When `EntityManager` is closed
- When the entity is serialized and the entity value is passed

The instance of the separated entity continues to exist outside the persistence context that is perpetuated and obtained. The states of the entity and the database are not synchronized.

(1) Accessing a detached entity

An application can access the *detached* entity even after the persistence context ends. In this case, the entity fields and relation destinations must already be fetched. The fields and the relation destination entities that are not fetched by the *detached* entity cannot be accessed. Note that `FetchType.EAGER` is always applied to `@Basic` specified in the field, so the relation destination entity and field information are already obtained.

To access a relationship from the *detached* entity, one of the following conditions must be satisfied:

- The entity instance is obtained using `find()`
- The entity is obtained by using a query or the entity is explicitly requested using the `FETCH JOIN` clause
- The persistence instance that is not a primary key is already accessed with the application
- The entity is already obtained from another valid entity by tracing the relation specified as `fetch=EAGER`

If unavailable instances are accessed and if available instances are accessed in a disabled state, an exception occurs. However, when `FetchType.LAZY` is specified with Cosminexus JPA Provider, if you access un-fetched fields and relation destination entities even if the entity is *detached* after `EntityManager` terminates, sometimes the value is obtained from the database and the contents can be referenced.

(2) merge processing of an entity

By invoking the `merge` method of `EntityManager` or by cascading the `merge` processing, you can merge the *detached* entity with the persistence context managed by `EntityManager`.

The following table describes the transition of entity states in the `merge` processing for each state of entity A.

Table 6–14: Results of transition of entity states in the merge processing

State of entity A	Results of state transition
new	A new entity A' is created with managed state and the state of entity A is copied to entity A'. Note that the entity of the merge argument remains new.
managed	The merge operation is ignored. However, if MERGE or ALL is specified in the cascade attribute for the relationship from entity A to other entities, the merge operation is propagated to the entities referenced by entity A.
detached	The state of entity A is copied to entity A' that has the same ID and already exists and is being managed, or a new managed entity is created that is a copy of entity A. Note that the entity of the merge argument remains detached.
removed	IllegalArgumentException is thrown by the merge operation, or transaction commit fails. Note that the state of entity A remains removed.

Note 1 If the relation from entity A to entity B is specified with `cascade=MERGE` or `cascade=ALL`, all the entity B are recursively merged as entity B'. Entity B' is set in entity A'. Note that if entity A is in the managed state, entity A and entity A' are the same.

Note 2 If entity B is referenced when `cascade=MERGE` or `cascade=ALL` is not specified in the relation of entity A, when entity A is merged with entity A' and if you trace the relation from entity A', you will finally reach the reference of the managed entity B' with the same persistence identity as entity B.

With the JPA specifications, the fields marked as LAZY to imply that the field is not fetched, are ignored during merge. However, with Cosminexus JPA Provider, @Basic operates as EAGER, therefore, all the fields that are not relationships are subject to the merge processing.

Also, if the Version string is used in the entity, the version of the entity is checked during the merge operation and during the flush and commit processing invoked after the merge operation. If the Version string does not exist, the version of the entity is not checked with the merge operation. For details, see *6.10.1 Optimistic lock processing*.

Note that Cosminexus JPA Provider does not support the processing to return to the persistence context using the entity merge processing between vendors.

(3) Notes

- In Cosminexus JPA Provider, the managed entity becomes a detached entity due to transaction commit with the persistence context of the transaction scope. On the other hand, with the extended persistence context, the managed entity remains managed.
- If the transaction is rolled back in the transaction scope and in the extended persistence context, all the existing managed instances and removed instances become detached. The state of the instance is the state existing when the transaction is rolled back.
- If the transaction is rolled back, the state of the persistence context becomes the state existing at rollback, so the state conflicts with the database state. Note that in Cosminexus JPA Provider, the version attribute state and the generated state form conflicting states, therefore, if the merge operation is performed, an exception might occur.

6.4.7 managed entity

You can use the `contains()` method of `EntityManager` to obtain information about whether the entity instance is being managed with the current persistence context.

This subsection describes the conditions for the return value of the `contains()` method.

- **Conditions when the contains() method returns true**
 - When the entity is acquired from the database and is not deleted from `EntityManager`, or is not separated
 - When the entity instance is generated and the `persist` method is executed for that entity or the `persist` operation is propagated to that entity
- **Conditions when the contains() method returns false**
 - When the entity instance is separated

- When the `remove` method is executed for the entity or the `remove` operation is propagated to the entity
- When the entity instance is generated and the `persist` method is not executed for that entity or the `persist` operation is not propagated to that entity

The actual `insert` and `delete` processing in the database is delayed until the conclusion of the transaction. At the same time, note that the propagation of `persist` and `remove` is applied immediately with the `contains` method.

Make sure that the entity instance is only managed using a single persistence context in the application. With Cosminexus JPA Provider, the operations do not function properly when the same Java instance is managed with multiple persistence contexts.

6.5 Defining the mapping information between the database and Java objects

With Cosminexus JPA Provider, you can define the information for mapping the database and the Java objects. You define the mapping information with an annotation or the O/R mapping file.

- **Definition using an annotation**

You define the mapping information directly in the entity class of the application.

- **Definition using an O/R mapping file**

An O/R mapping file is an XML file used for describing the mapping information. You use tags to define the mapping information.

If the mapping information is defined in both annotations and O/R mapping file, the definition in the O/R mapping file is given priority. Therefore, if you use the O/R mapping file to change the mapping information defined by using annotations, you can change the mapping information without changing the application.

Determine whether you want to use an annotation or an O/R mapping file or you want to use the annotation and O/R mapping file together, in accordance with the application creation policy.

6.6 Entity relationships

With an entity, the relation between the database tables can be expressed using a relationship. With Cosminexus JPA Provider, you can set an entity relationship.

6.6.1 Relationship types

A relationship consists of a relation and a direction. The possible relations are `OneToOne`, `ManyToOne`, `OneToMany`, and `ManyToMany`. Furthermore, the possible directions of each relation are unidirectional and bi-directional. The relations and directions are combined to form the following seven types of relationship:

- Unidirectional `OneToOne` relationship
- Unidirectional `ManyToOne` relationship
- Unidirectional `OneToMany` relationship
- Unidirectional `ManyToMany` relationship
- Bi-directional `OneToOne` relationship
- Bi-directional `ManyToOne/OneToMany` relationship
- Bi-directional `ManyToMany` relationship

This subsection describes relationships using the employees and departments of a company as an example. The relation between the employees and departments is as follows:

- The entity expresses the employees and departments respectively.
- The employees are assumed to belong to some department.
- A department maintains multiple employees.
- The departments are referenced from employees and the employees are referenced from departments.

This example expresses the bi-directional `ManyToOne/OneToMany` relationship of entities. In this case, `Many` stands for the employees and `One` stands for the department. The following figure shows the relation between the employee entity and the department entity.

Figure 6-4: Relation between the employee entity and the department entity



Note that in an entity, you can specify settings to propagate the operations to the relation destination entity. These settings are called *cascade*. When you perform an operation for an entity and if *cascade* is specified, similar operations are automatically executed even in the entities that have a relationship with the operated entity. If *cascade* is used, the user can save the effort of performing operations for the relation destination entity.

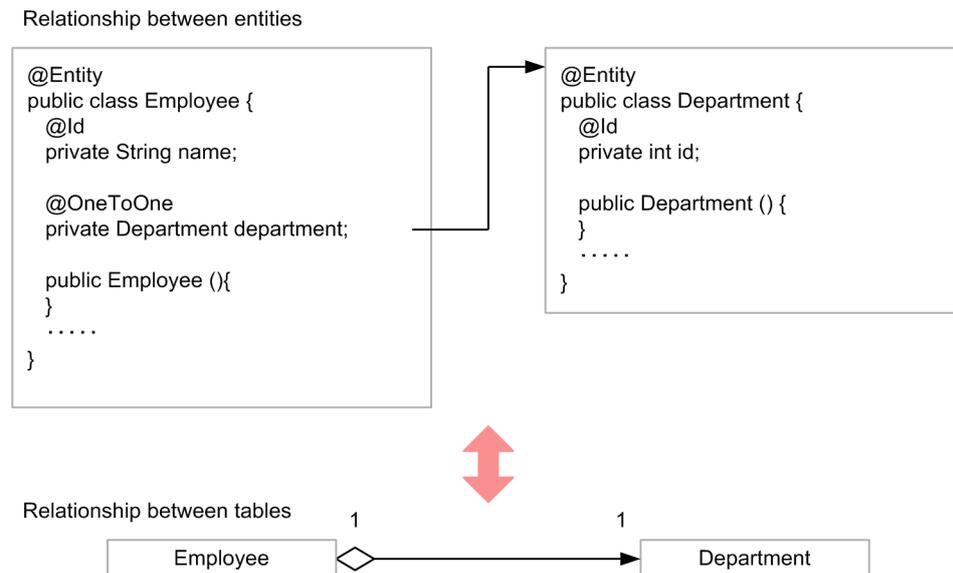
6.6.2 Annotations for relationships

The entities handle the relation between tables as a relationship. When you handle a relationship with an entity, the reference to another entity is stored as a field in the entity. In addition, set the following relationship annotations in the persistence property or instance variable that references an entity:

- `OneToOne` (One-to-one)
- `OneToMany` (One-to-many)
- `ManyToOne` (Many-To-One)
- `ManyToMany` (Many-To-Many)

The following figure shows the relationship between entities and the relation between tables.

Figure 6–5: Relationship between entities and relation between tables



If the relationship annotation is not specified for referencing the entity, the operations will not function properly. Note that if the `generic` type is not used in the reference where collection is used, the target entity must be specified as the relationship target.

Instead of using an annotation, you can also define a relationship using the O/R mapping file. However, note that when a relationship is defined in the O/R mapping file and if the same definition is specified in the annotation, the definition in the annotation is overwritten.

Default mapping in the relationship annotations

When the annotations for the `OneToOne`, `OneToMany`, `ManyToOne`, and `ManyToMany` relationships are applied, the default mapping rules are applied. Similarly when a relationship is specified in the O/R mapping file, the default mapping is applied.

When you use the default mapping of the relationship annotations, if the database tables and relations differ from the default values specified in *6.6.4 Default mapping (bi-directional relationship)* or *6.6.5 Default mapping (unidirectional relationship)*, the SQL statement cannot be created correctly in the database query during execution. An exception occurs.

6.6.3 Direction of relationships

A relationship includes a bi-directional relationship and a unidirectional relationship. When a bi-directional relationship is handled, a relationship has an owner and a non-owner. A unidirectional relationship only has an owner. The owner of a relationship can take decisions on updating a database relationship.

The following rules are applied to a bi-directional relationship:

- The non-owner of a bi-directional relationship specifies the owner based on the `mappedBy` element of `@OneToOne`, `@OneToMany`, and `@ManyToOne`. With the `mappedBy` element, you specify the properties or field names that reference the non-owner side using the owner entity.
- With the `ManyToOne/ OneToMany` bi-directional relationship, set `many` as the owner. Therefore, the `mappedBy` element cannot be specified in `@ManyToOne`.
- With the `OneToOne` bi-directional relationship, the entity on the side containing the external key becomes the owner.
- In the `ManyToMany` bi-directional relationship, any entity might be set as the owner.

A relationship annotation has a `cascade` attribute. If you specify the `cascade` attribute, you can propagate the operations for the entities even for the reference destination entities. However, you can specify `REMOVE` in the

`cascade` attribute of the relationship annotation only for `OneToOne` or `OneToMany`. If `cascade=REMOVE` is applied for other relations, the operations might not function properly. For details on the `cascade` attribute when the entities have relationships, see *6.4.1(4) Propagation of operations to the entities*.

! Important note

Cosminexus JPA Provider does not implement the check for maintaining the consistency of relationships during execution. Therefore, when you update the relationships during the execution of an application, even if the update causes an inconsistency in the relationships, no warning or exception occurs.

Note that when a value is fetched from the database with collection relationships such as `OneToMany` or `ManyToMany`, an empty collection is returned as the relationship value if the related entity does not exist.

6.6.4 Default mapping (bi-directional relationship)

This subsection describes the default mapping of a bi-directional relationship.

(1) Bi-directional `OneToOne` relationship

This section describes the default mapping of a bi-directional `OneToOne` relationship applied in the following conditions:

Conditions

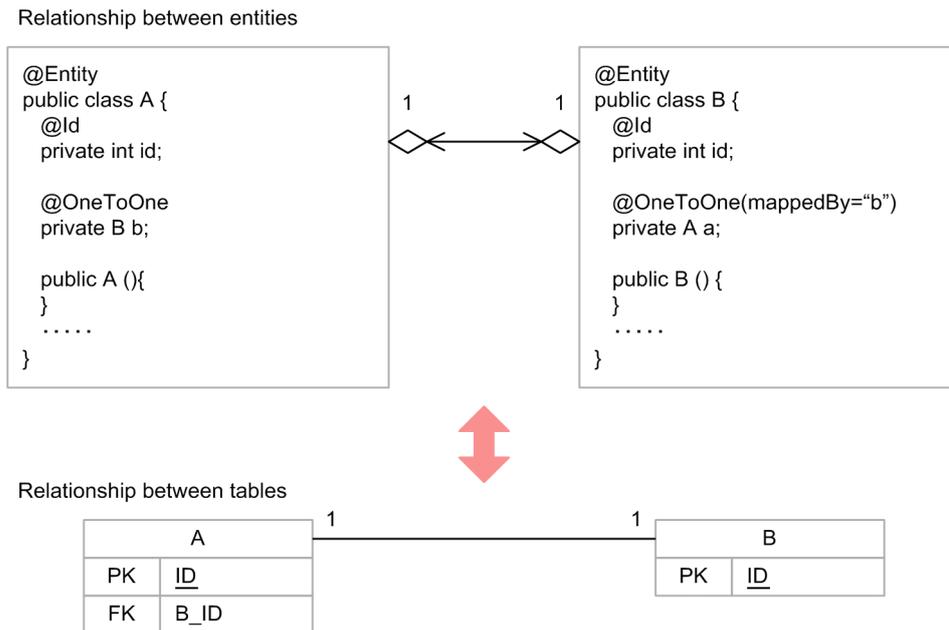
- Entity A references the stand-alone instance of entity B and sets `@OneToOne`.
- Entity B references the stand-alone instance of entity A and sets `@OneToOne`. The persistence property (or field) name that references entity B using entity A is specified in the `mappedBy` attribute of `@OneToOne`.
- Entity A is the relationship owner.

Applied default mapping

- Entity A is mapped to table A.
- Entity B is mapped to table B.
- Table A must have the external key for table B. The name of the external key string is as follows:
Name of the external key string
Name of the relationship property (or field) of entity A_ Name of the primary key string of table B
 Note Italics indicate a variable value.

Also, the external key string has the same type as the primary key of table B and is a unique key constraint.

Figure 6-6: Default mapping in a bi-directional OneToOne relationship



(2) Bi-directional ManyToOne/ OneToMany relationship

This section describes the default mapping of a bi-directional ManyToOne/ OneToMany relationship applied in the following conditions:

Conditions

- Entity A references the stand-alone instance of entity B and sets @ManyToOne (or the corresponding XML tags in the O/R mapping file).
- Entity B references the entity A collection and sets @OneToMany (or the corresponding XML tags in the O/R mapping file). The mappedBy attribute is specified in @OneToMany. The mappedBy attribute specifies the persistent property (or field) name set for referencing entity B with entity A.
- Entity A is the relationship owner.

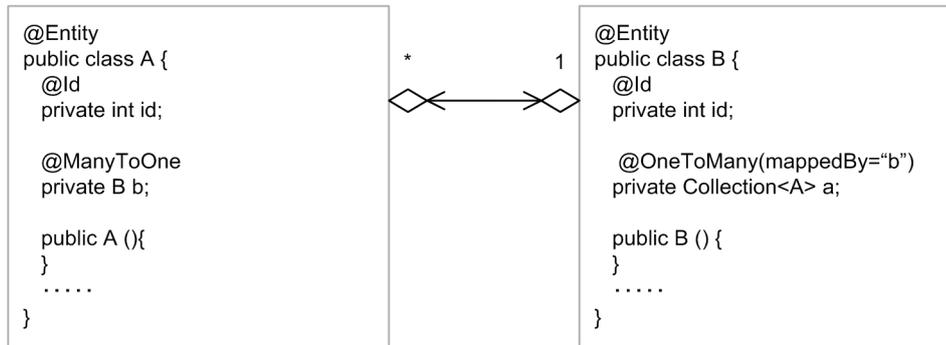
Applied default mapping

- Entity A is mapped to table A.
- Entity B is mapped to table B.
- Table A must have the external key for table B. The name of the external key string is as follows:
Name of the external key string
Name of the relationship property (or field) of entity A_ Name of the primary key string of table B
 Note Italics indicate a variable value.

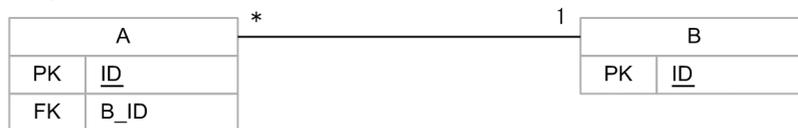
The external key string has the same type as the primary key of table B.

Figure 6-7: Default mapping in a bi-directional ManyToOne/OneToMany relationship

Relationship between the entities



Relationship between the tables



(3) Bi-directional ManyToMany relationship

This section describes the default mapping of a bi-directional ManyToMany relationship applied in the following conditions:

Conditions

- Entity A references the entity B collection. `@ManyToMany` (or the corresponding XML element in the O/R mapping file) is set in the collection.
- Entity B references the entity A collection. `@ManyToMany` (or the corresponding XML tags in the O/R mapping file) is set in the collection and the `mappedBy` attribute is specified. The `mappedBy` attribute specifies the persistent property (or field) name set for referencing entity B with entity A.
- Entity A is the relationship owner.

Applied default mapping

- Entity A is mapped to table A.
- Entity B is mapped to table B.
- Apart from table A and B, a junction table named A_B where the name of the owner table appears at the beginning, is necessary. This junction table has two external key strings.

The first external key string references table A and has the same type as the primary key of table A. The name of this external key string is as follows:

Name of the external key string

Name of the relationship property (or field) of entity B_ Name of the primary key of table A

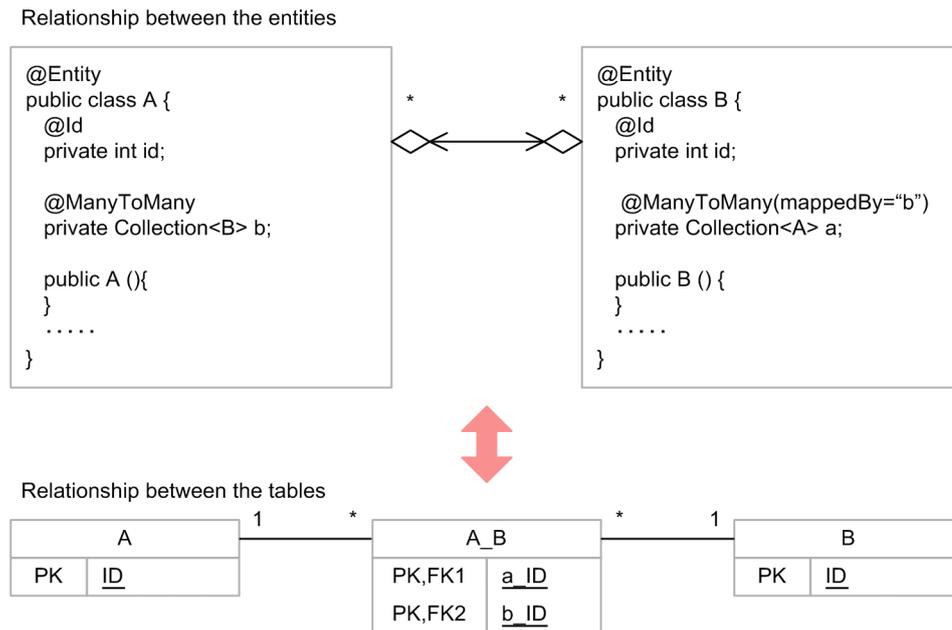
The other external key string references table B and has the same type as the primary key of table B. The name of this external key string is as follows:

Name of the external key string

Name of the relationship property (or field) of entity A_ Name of the primary key of table B

Note Italics indicate a variable value.

Figure 6–8: Default mapping in a bi-directional ManyToMany relationship



6.6.5 Default mapping (unidirectional relationship)

A unidirectional relationship includes a Single-Valued relationship and a Multi-Valued relationship. The following points describe each relationship:

- **Unidirectional Single-Valued relationship**

A unidirectional Single-Valued relationship is a relationship that references the stand-alone instance and where only the owner exists.

Possible unidirectional Single-Valued relationships are unidirectional OneToOne relationships and unidirectional ManyToOne relationships.

- **Unidirectional Multi-Valued relationship**

A unidirectional Multi-Valued relationship is a relationship that references the entity in the collection format and where only the owner exists.

Possible unidirectional Multi-Valued relationships are unidirectional OneToMany relationships and unidirectional ManyToMany relationships.

This subsection describes the default mapping of a unidirectional relationship.

(1) Unidirectional Single-Valued relationship

This section describes the default mapping of a unidirectional OneToOne relationship and a unidirectional ManyToOne relationship.

(a) Unidirectional OneToOne relationship

This section describes the default mapping of a unidirectional OneToOne relationship applied in the following conditions:

Conditions

- Entity A references the stand-alone instance of entity B and sets @OneToOne (or the corresponding XML tags in the O/R mapping file).
- Entity A is not referenced from entity B.
- Entity A is the relationship owner.

Applied default mapping

- Entity A is mapped to table A.
- Entity B is mapped to table B.
- Table A must have the external key for table B. The name of the external key string is as follows:

Name of the external key string

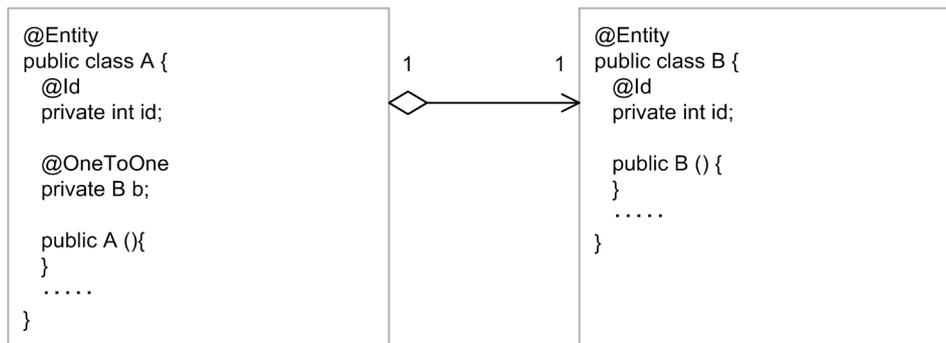
Name of the relationship property (or field) of entity A_ Name of the primary key string of table B

Note Italics indicate a variable value.

The external key string has the same type as the primary key of table B and is a unique key constraint.

Figure 6–9: Default mapping in a unidirectional OneToOne relationship

Relationship between the entities



Relationship between the tables



(b) Unidirectional ManyToOne relationship

This section describes the default mapping of a unidirectional ManyToOne relationship applied in the following conditions:

Conditions

- Entity A references the stand-alone instance of entity B and sets @ManyToOne (or the corresponding XML tags in the O/R mapping file).
- Entity A is not referenced from entity B.

Applied default mapping

- Entity A is mapped to table A.
- Entity B is mapped to table B.
- Table A must have the external key for table B. The name of the external key string is as follows:

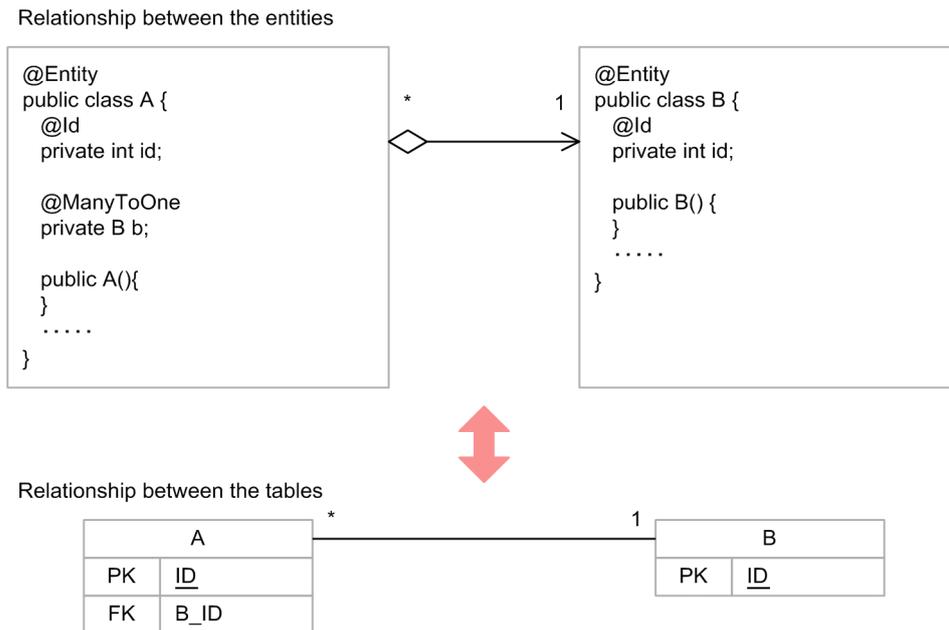
Name of the external key string

Name of the relationship property (or field) of entity A_ Name of the primary key string of table B

Note Italics indicate a variable value.

The external key string must have the same type as the primary key of table B.

Figure 6–10: Default mapping in a unidirectional ManyToOne relationship



(2) Unidirectional Multi-Valued relationship

This section describes the default mapping of a unidirectional OneToMany relationship and a unidirectional ManyToMany relationship.

(a) Unidirectional OneToMany relationship

This section describes the default mapping of a unidirectional OneToMany relationship applied in the following conditions:

Conditions

- Entity A references the entity B collection. `@OneToMany` (or the corresponding XML tags in the O/R mapping file) is set in the collection.
- Entity A is not referenced from entity B.
- Entity A is the relationship owner.

Applied default mapping

- Entity A is mapped to table A.
- Entity B is mapped to table B.
- Apart from table A and B, a junction table named A_B where the name of the owner table appears at the beginning, is necessary. This junction table has two external key strings.

The first external key string references table A and has the same type as the primary key of table A. The name of this external key string is as follows:

Name of the external key string

Name of entity A_ Name of the primary key string of table A

The other external key string references table B, has the same type as the external key of table B, and is a unique key constraint. The name of this external key string is as follows:

Name of the external key string

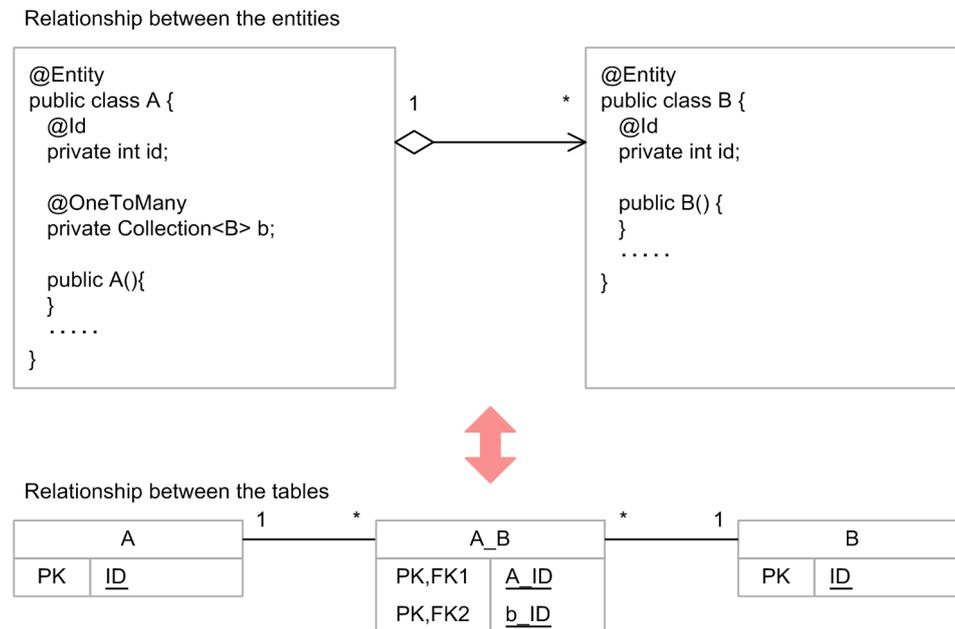
Name of the relationship property (or field) of entity A_ Name of the primary key string of table B

Note Italics indicate a variable value.

! Important note

A unidirectional OneToMany relationship that uses a junction table as described in this example might not be supported with JPA Providers other than Cosminexus JPA Provider. Note this when you operate applications created with a unidirectional OneToMany relationship using JPA Providers other than Cosminexus JPA Provider.

Figure 6–11: Default mapping in a unidirectional OneToMany relationship

**(b) Unidirectional ManyToMany relationship**

This section describes the default mapping of a unidirectional ManyToMany relationship applied in the following conditions:

Conditions

- Entity A references the entity B collection. `@ManyToMany` (or the corresponding XML tags in the O/R mapping file) is set in the collection.
- Entity B does not reference entity A.
- The owner is entity A.

Applied default mapping

- Entity A is mapped to table A.
- Entity B is mapped to table B.
- Apart from table A and B, a junction table named A_B where the name of the owner table appears at the beginning, is necessary. This junction table has two external key strings.

One of the external key strings references table A and has the same type as the external key of table A. The name of this external key string is as follows:

Name of the external key string

Name of entity A *Name of the primary key of table A*

The other external key string references table B and has the same type as the primary key of table B. The name of this external key string is as follows:

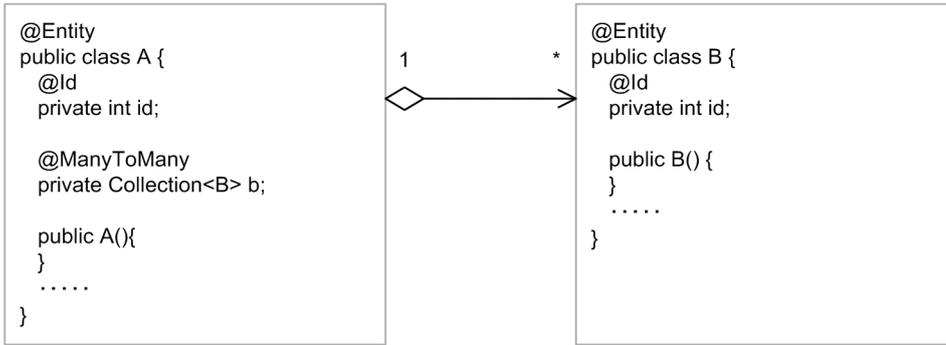
Name of the external key string

Name of the relationship property (or field) of entity A *Name of the primary key of table B*

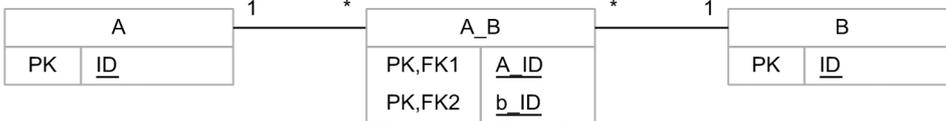
Note Italics indicate a variable value.

Figure 6-12: Default mapping in a unidirectional ManyToMany relationship

Relationship between the entities



Relationship between the tables



6.7 Cache functionality of the entity objects

The cache functionality of the entity objects is a functionality that stores the entity objects used by an application in the memory. When you use the cache functionality of the entity objects, the entity objects cached in Cosminexus JPA Provider are used if the same entity objects are operated. The data is not read from the database again, so the access to the database is minimized and the load on the processing performance can be reduced. Note that this functionality is unique to Cosminexus JPA Provider.

This section describes the cache functionality of the entity objects.

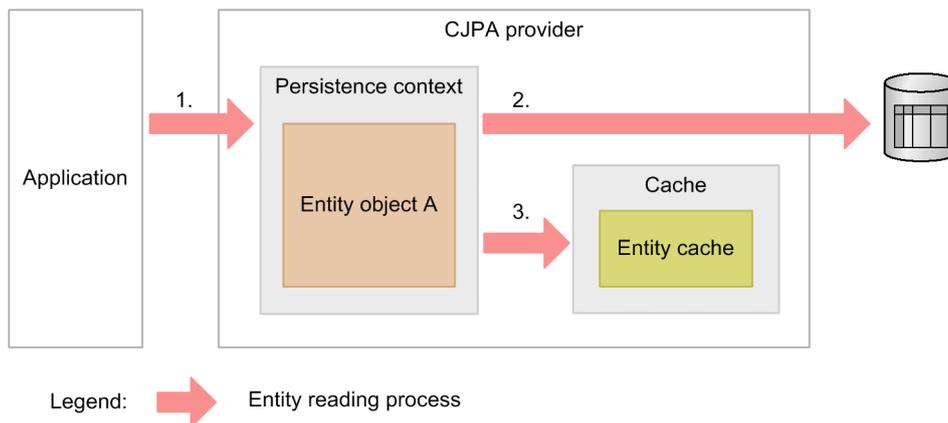
6.7.1 Processing of the cache functionality

If you use the cache functionality, when the same entity is read, the data is obtained from the cache instead of the database. This subsection describes the procedure for processing the cache functionality, the cache registration and update timing, and the procedure for updating the cache.

(1) Procedure for processing the cache functionality

The following figure shows the flow of processing of the cache functionality.

Figure 6-13: Flow of processing of the cache functionality



A description of the above figure is as follows:

- **Reading of the entity**

When an entity is first read with methods such as `find`, the flow of processing is as follows:

1. The entity is read.
2. The data is obtained from the database.
3. The entity object of the obtained data is registered in the cache.

When the entity has a relationship and when you obtain the relationship destination entity, if the target entity exists in the cache, the entity in the cache is referenced without accessing the database.

A cache exists for each persistence context.

(2) Cache registration and update timing

Registration and update in a cache is implemented at the following times:

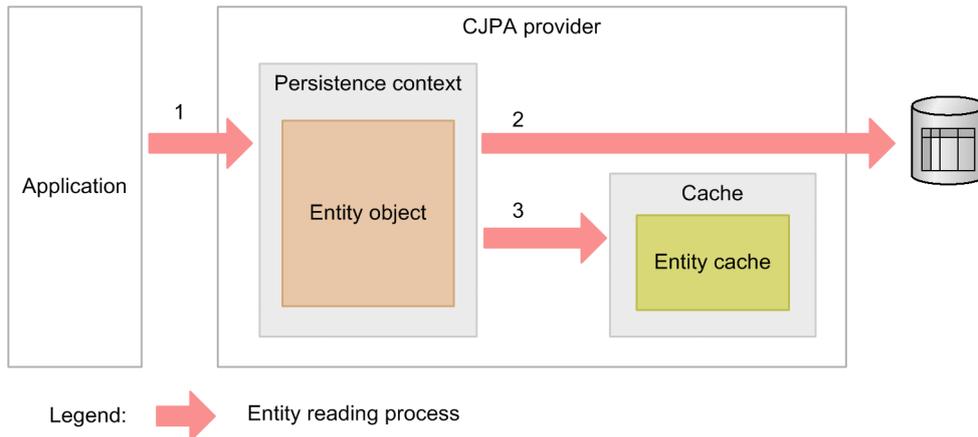
- Cache registration timing
 - During the reading of an entity object (`find` operation) when the target cache does not exist
- Cache update timing
 - When the entity is refreshed (`refresh` operation)

- When a transaction is committed
 - When an exception occurs in the optimistic lock processing
- Note that when the `OptimisticLockException` exception occurs in the optimistic lock processing, the entity objects registered in the cache are deleted.

(3) Procedure for updating the cache

The following figure shows the flow of updating the cache.

Figure 6-14: Updating the cache



A description of the figure is as follows:

1. The following operations are implemented for the entity object:
 - `refresh` operation
 - Transaction commit
2. During the `refresh` operation, the database is accessed.
3. The data existing in the persistence context is registered in the cache and the cache data is updated.

Note that the cache is registered even when JPQL is executed. The cache is registered if JPQL is executed when the target cache does not exist. The cache data is used in JPQL as well, but the database is accessed irrespective of the presence or absence of the cache data. Therefore, the cache-based processing performance cannot be expected to improve. For details, see point (4).

The cache stores information for the entity objects; therefore, if the object returned during the execution of the query is the entity itself, the cache is updated. The cache is not updated when other fields are specified. The JPQL examples when the cache update is valid and when the update is invalid are as follows:

- JPQL example when the cache is updated

```
SELECT emp FROM Employee AS emp
```

- JPQL example when the cache is not updated

```
SELECT emp.id, emp.name, emp.address FROM Employee AS emp
```

(4) Relation between JPQL and cache

When all the entity objects are obtained with JPQL, if there is information registered in the cache, the cache information is obtained. The cache functionality of Cosminexus JPA Provider requires a primary key that is the ID for identifying the target entity. To obtain the primary key, the JPQL result must be obtained and at this time, the database is accessed. The primary key is extracted from the database access results and the entity object is obtained from the cache. Also, if the target data does not exist in the cache, the entity is created from the database information.

With JPQL, the database is accessed regardless of the presence or absence of the cache data. Therefore, for JPQL, cache-based performance improvement cannot be expected.

6.7.2 Cache reference forms and cache types

There are three types of cache reference forms:

- **Hard reference**

This reference is not collected by garbage collection.

- **Weak reference (`java.lang.ref.WeakReference`)**

If the reference is weakly reachable, the reference is collected by garbage collection. Note that whether the cache reference is weakly reachable depends on the specifications in `java.lang.ref.WeakReference`.

The examples of weak reference cache that are not collected with Cosminexus JPA Provider are as follows:

- Cache of an entity object registered in the persistence context
- Cache that references a non-weakly reachable cache using a relationship
- Cache wherein the cache of another entity object with an inheritance relationship is not weakly reachable, when the entity inheritance strategy is used

- **Soft reference (`java.lang.ref.SoftReference`)**

This is a reference form that caches out when the remaining amount of memory decreases.

A soft reference is collected by garbage collection depending on the consumption rate and survival time of a resource. The specifications such as the collection timing and the selection of objects for collection depend on JavaVM.

The cache types differ depending on which form is used to reference the cache. The following table describes the mapping of the cache reference forms and cache types.

Table 6-15: Mapping of the cache reference forms and cache types

Cache reference forms	Cache types
Hard reference	Full
Hard reference + Weak reference ^{#1}	HardWeak
Weak reference + Soft reference ^{#1}	SoftWeak
Weak reference	Weak
None ^{#2}	None

^{#1} The reference forms are combined.

^{#2} The entity object is not cached.

With Cosminexus JPA Provider, you can choose the cache type. Choose the type based on the application design and environment. Specify the cache type in `persistence.xml`. For details on `persistence.xml`, see 6.2 *persistence.xml* in the *uCosminexus Application Server Definition Reference Guide*.

The following points describe the cache types.

(1) Full

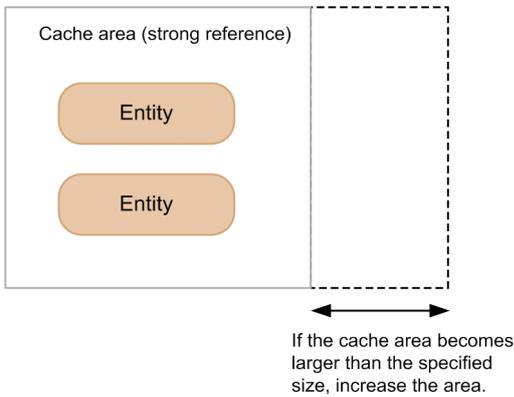
All the entities are cached with a hard reference.

If you specify `Full` in the cache type, the access to the database decreases, so the processing load decreases. However, the memory continues to be occupied, therefore, the load on the memory increases.

Specify `Full` when the duration of the entity object is long and when the reference is created for a few entity objects that require frequent access. Furthermore, when several entity objects are read, the memory load increases, so we do not recommend using `Full` to update multiple records in a batch.

When Full is specified, the hard reference area is allocated with the specified cache size. If the hard reference area exceeds the defined size, the area is increased based on the Hashtable specifications. The following figure shows the image in the cache when Full is specified.

Figure 6-15: Cache image for Full



(2) HardWeak

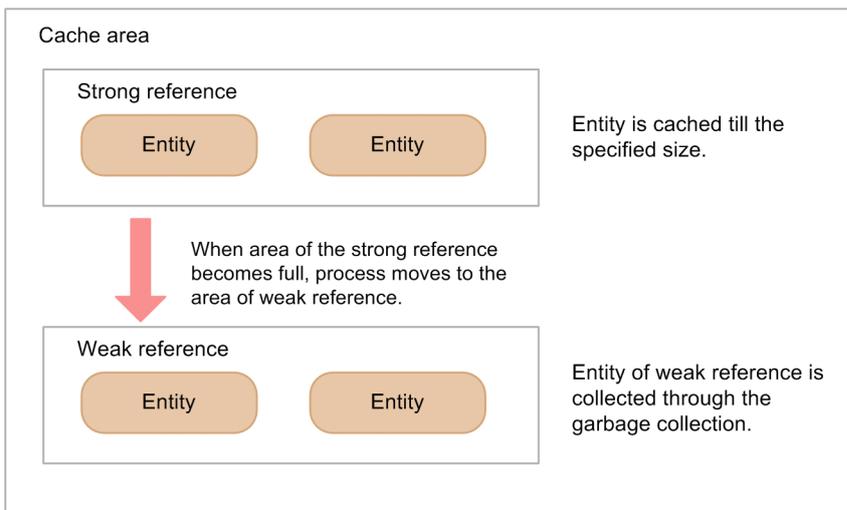
The entities are cached with a hard reference and weak reference.

When you want to store the entity object in a list, use a hard reference. Create the hard reference area with a fixed length only up to the value specified in the cache size. If the cache size reaches the specified value, the old entity objects are moved to the weak reference. At this time, the entity objects for which cache registration has not been used for the longest time are moved sequentially to the weak reference. If the entity objects moved to the weak reference are used, the entity objects are once again stored in the hard reference area.

If you specify `HardWeak`, you can use the entity objects with a long duration to efficiently control the memory used in the cache.

If the state of insufficient memory occurs frequently in a system where `SoftWeak` is used, you cannot take advantage of the soft reference; therefore, use `HardWeak`. The following figure shows the image in the cache for `HardWeak`.

Figure 6-16: Cache image for HardWeak



In the case of `HardWeak`, the cache is stored with a hard reference in the hard reference cache area. Also, the cache is stored with a weak reference in the weak reference cache area.

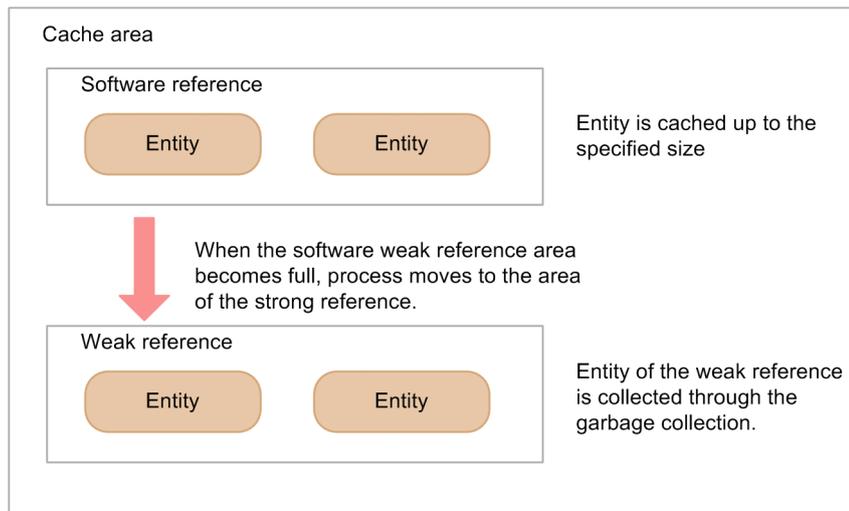
(3) SoftWeak

The entities are cached with a soft reference and weak reference.

You use the soft reference to store the entity objects in a list and create a soft reference area of a fixed length only up to the value specified in the cache size. If the cache size reaches the specified value, the old entity objects are moved to the weak reference area. At this time, the entity objects for which cache registration has not been used for the longest time are moved sequentially to the weak reference. If the entity objects moved to the weak reference are used, the entity objects are once again stored in the hard reference area.

If you specify `SoftWeak`, you can use the entity objects with a long duration to efficiently control the memory used in the cache. Therefore, when you use the cache functionality, we recommend that you specify `SoftWeak`. The following figure shows the image in the cache for `SoftWeak`.

Figure 6-17: Cache image for `SoftWeak`



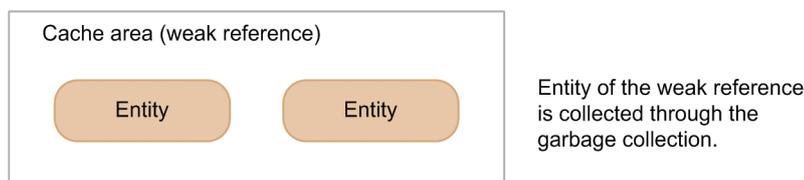
In the case of `SoftWeak`, the cache is stored with a soft reference in the soft reference cache area. Also, the cache is stored with a weak reference in the weak reference cache area.

(4) Weak

All the entities are cached with a weak reference.

Therefore, all the entity objects are subject to garbage collection. If you specify `weak`, the memory load decreases, but the cache might be deleted due to garbage collection. Use the `Weak` cache type in systems that do not place significance on the cache functionality of the entity objects. The following figure shows the image in the cache for `weak`.

Figure 6-18: Cache image for `Weak`



In the weak cache type, the entities are cached with a weak reference.

(5) NONE

No entity objects are cached. Use the `None` cache type if you want to destroy an entity object immediately after the entity object is read from the database.

6.7.3 Scope of the cache functionality

The cache data is stored for each persistence context. Therefore, the data cannot be obtained from the cache even if the entity object is invoked in another persistence context.

Note that the cache data is stored for the duration from the generation of the persistence context until the persistence context is destroyed.

6.7.4 Notes on using the cache functionality

This subsection describes the notes on using the cache functionality of the entity objects.

(1) Notes on updating or deleting the data in a query

When JPQL and native query are used in the application to update and delete the data, the cache contents are not updated. Execute operations such as the `refresh` operation to obtain the database contents again.

The cache data is read in the following operations; therefore, the data is not updated in the database:

1. The data is read into the entity object.
The entity data is registered in the cache as well.
2. The delete query containing the data read in 1 is executed.
The data is deleted by the execution of the delete query, but the cache is not deleted.
3. The same data as in 1 is read into the entity object.
Because the data is the same as 1, the data existing in the cache is read.
4. The data in 3 is flushed.
The applicable line does not exist in the database, so the add or update processing cannot be performed.

(2) Notes on multiple persistence contexts using a cache

By using the cache, you can reduce the database access frequency. However, on the other hand, a time lag occurs in the data due to the cache and the frequency of optimistic lock exception might increase.

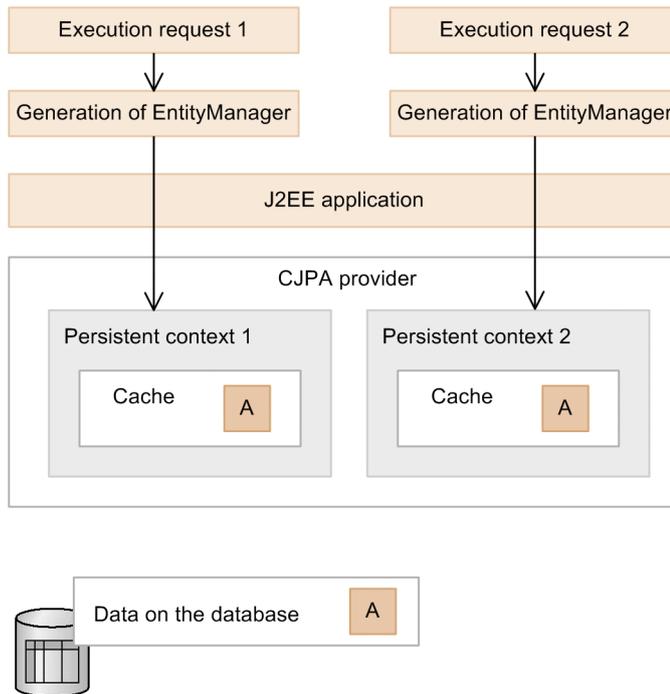
A cache exists for each persistence context. Therefore, multiple `EntityManagers` existing in pairs with the persistence contexts are generated simultaneously, and if the entities with the same primary keys are operated at the same time, even if the data is updated, the user might not be able to reference the updated data in a timely manner. Due to this, the optimistic lock exception occurs easily.

The following points describe the mechanism and action for the occurrence of the optimistic lock exception.

(a) Example of optimistic lock exception

This section describes an example of cache for each persistence context in the environment shown in the following figure.

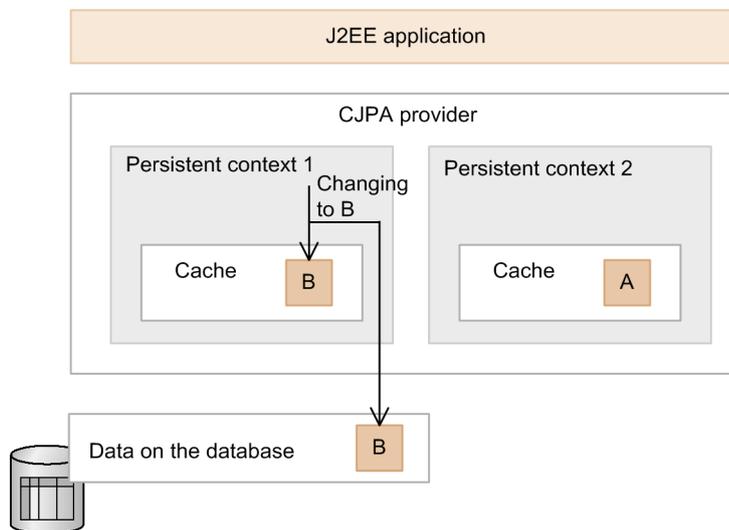
Figure 6-19: Environment described in this example



In the figure, the cache and the database are consistent and data A is already stored in the cache.

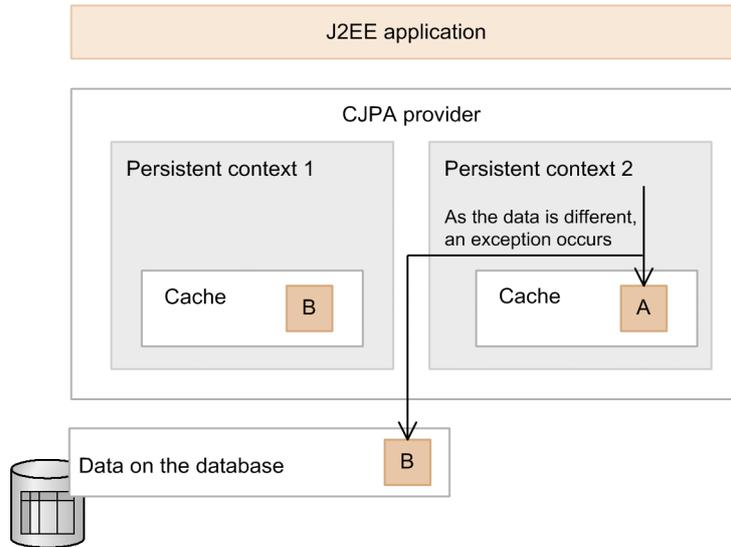
1. In persistence context 1, the data is changed from A to B.
At this time, the contents of the cache and the database are identical, so an exception does not occur.

Figure 6-20: Changes in data in persistence context 1



2. After the processing of 1 ends, A data is changed in persistence context 2.
The cache data is not changed; so the database data and the cache are inconsistent. Therefore, an exception is thrown due to the optimistic lock.

Figure 6-21: Changes in data in persistence context 2



In such an environment, if you use the cache, the non-updated cache is left behind and an exception might occur due to the optimistic lock.

(b) Action

When an optimistic lock exception occurs, the relevant objects in the cache are deleted. Therefore, execute the `find` method or the `refresh` method to obtain all the related data from the database once again. With this, you can synchronize the cache data and the database.

(3) Notes on the cache registration and update timing

- When you use JPQL to obtain the pessimistic lock, the entity object returned when the query is executed is not registered in the cache.
- When a native query is used to execute the query that sets the entity as the return value in `@SqlResultSetMapping`, the entity object of the return value is stored in the cache.
- During the `refresh` operation of an entity, if `OptimisticLockException` occurs in another thread after the entity object is read and the cache is deleted, the entity object is not registered in the cache even if the refresh processing is executed.

6.8 Auto-numbering of the primary key values

The primary key numbering functionality automatically generates the primary key value when the entity object is used to insert a record. Due to this functionality, even if the user does not specify the primary key value, a unique value is stored. Cosminexus JPA Provider provides the primary key numbering functionality.

How to generate the primary key value

There are four methods for generating the primary key values:

- TABLE

In this method, you generate the primary key values by using a table to store the primary key values.

- SEQUENCE

In this method, you use the database sequence objects to generate the primary key values. However, if HiRDB is used as the database, implement the same processing as that of TABLE with Cosminexus JPA Provider.

- IDENTITY

In this method, you use the `identity` column of the database to generate the primary key values. However, with Cosminexus JPA Provider, the operations differ depending on the database type used.

In HiRDB, you implement the same processing as that of TABLE.

In Oracle, you implement the same processing as that of SEQUENCE.

- AUTO

You choose the generation method suitable for the database used. With Cosminexus JPA Provider, choose TABLE for both HiRDB and Oracle.

Timing when the primary key values are numbered

In Cosminexus JPA Provider, the primary key values are numbered during the `flush` operation or when a transaction is committed.

Example of SEQUENCE as the primary key value generation method

The following is an example of using SEQUENCE as the generation method of the primary key values. In this example, a sequence object named EMP_SEQ is assumed to have been created beforehand.

```

@Entity
public class Employee {
    ...
    @SequenceGenerator(
        name="EMPLOYEE_GENERATOR",
        sequenceName="EMP_SEQ"
    )
    @Id
    @GeneratedValue(strategy=GenerationType.SEQUENCE,
generator="EMPLOYEE_GENERATOR")
    @Column(name="EMPLOYEE_ID")
    public Integer getId(){
        return id;
    }
    ...
}

```

6.9 Database operations based on the query language

To use the JPA for operating the database data, implement the operation through the `javax.persistence.Query` interface. Using the `javax.persistence.Query` interface enables you to execute operations, such as search, update, and delete, for multiple records at the same time. To use the `javax.persistence.Query` interface, the user uses the query language to operate the database.

With Cosminexus JPA Provider, you can use JPQL and SQL as the query language. A description of JPQL and SQL is as follows:

- **JPQL**
JPQL is a query language defined in the JPA specifications. This language does not depend on the database and operates for the entity class.
- **SQL**
SQL is a database-dependent query language. SQL is also called a native query. This language operates for the database data.

For details on the database operations based on JPQL and SQL, see *6.16 Procedure for referencing and updating the database with the query language*. Furthermore, for details on the JPQL syntax, see *6.17 JPQL coding method*.

6.10 Optimistic lock

With Cosminexus JPA Provider, you use `EntityManager` and persistence context to manage the entities to be persisted. The changed entity information is applied to the database when the `flush` method or transaction commit processing is executed. When the same line of the database might be updated at the same time in multiple transactions, the data integrity must be ensured. Cosminexus JPA Provider provides the optimistic lock to ensure data integrity.

The optimistic lock is a lock functionality for making sure that the data is not updated by other applications from the beginning of the data update processing until the update processing is complete. The optimistic lock does not lock the database, and therefore, has the advantage that deadlocks do not occur.

This section describes the optimistic lock.

6.10.1 Optimistic lock processing

If you use the optimistic lock, Cosminexus JPA Provider checks whether the database data is being updated by other applications, instead of the user. If the database data is being updated by other applications, Cosminexus JPA Provider throws an exception and notifies the user that the data is being updated. Furthermore, Cosminexus JPA Provider marks the transaction for rollback.

(1) Procedure for checking whether the data is updated

Whether the data is updated is checked by the presence or absence of update of the version column prepared in the database table. If the data in the database is updated, the version number of the version column is updated. As a result, the user understands that the database was updated by another application. The following table describes the states and operations of the version column when the database is updated.

Table 6-16: States and operations of the version column when the database is updated

State of the version column in the table	Operation
When the value of the version column is not updated	Cosminexus JPA Provider applies the entity information to the database. At this time, the value of the version column in the database is updated.
When the value of the version column is updated	Indicates that the database data is being updated by another application. Therefore, Cosminexus JPA Provider throws <code>OptimisticLockException</code> and marks the transaction for rollback.

In this way, using the state of the version column, you can ensure that another transaction does not update the data during the period from the reading of the entity until the database is updated.

(2) Checking the versions of the persistence fields and relationships

To use the optimistic lock, you check the versions of both persistence fields and relationships of the entities. To check the versions, set the `Version` field (property) corresponding to the version column for the entity. Set the `Version` field by using `@Version` or the `version` tag of the O/R mapping file.

The version of the entity is checked at one of the following timings:

- When the entity state is changed and that change is written to the database
- When the entity state is changed to managed by the `merge` processing[#]

#

The version is not only checked during the execution of `merge`, but also during `flush` or when the transaction is committed.

When the version of the entity is determined to be old by the version check, `OptimisticLockException` occurs. The transaction is also marked for rollback.

(3) Checking the versions during the flush operation or transaction conclusion

You can check the version of the entity during the `flush` operation or transaction conclusion. To check the version, specify the entity in the `lock` method of `EntityManager`. By using the `lock` method of `EntityManager`, you can add the entity as a target for version check in the transaction and change the update policy of the version column.

Cosminexus JPA Provider supports `LockModeType.READ` and `LockModeType.WRITE` as the timing for updating the version column (`LockModeType` of the `lock` method). Regardless of the content specified in the update timing for the version column, Cosminexus JPA Provider ensures that the following two events do not occur during transaction conclusion:

- Dirty Read

Transaction T1 changes a line. Then, before T1 executes commit and rollback, another transaction T2 reads the same line and obtains the changed value. Finally, T2 succeeds in commit.

Whether T1 executes commit or rollback is no longer important, but whether T1 performs commit or rollback before or after commit by T2 becomes important.

- Un-Repeatable Read

Transaction T1 reads a line. Then, before T1 executes commit, another transaction T2 changes or deletes that line. Finally, both the transactions succeed in commit.

If you specify `LockModeType.WRITE` as the `LockModeType`, the version column is forcefully updated even when there is no change in the entity state. The version column is updated when `flush` or transaction commit is invoked. Note that if the entity is deleted before the version column is updated, the update of the version column might be omitted.

6.10.2 Exception processing when optimistic lock fails

If you want to check the execution of an optimistic lock explicitly, invoke the `flush()` method. If an optimistic lock exception occurs during the invocation of the `flush()` method, you can catch the exception processing and perform the recovery processing. If the optimistic lock has no problems, the entity version is checked and the `Version` column is updated by the `flush()` method.

A coding example of the exception processing is as follows:

```
try{
    em.flush();
} catch (OptimisticLockException e){
    // Exception processing
}
```

As shown in this example, by wrapping `OptimisticLockException` during the exception processing, you can show the optimistic lock exception as an application exception. However, note that the transaction in this case is marked for rollback and cannot be committed.

Note that with Cosminexus JPA Provider, the entity that causes the exception is not stored in `OptimisticLockException`. The `getEntity()` method of `OptimisticLockException` always returns a null value.

6.10.3 Notes on using the optimistic lock

This subsection describes the notes on using the optimistic lock.

(1) Notes on the Version field settings

The notes on the `Version` field settings are as follows:

- If the `Version` field is not set, the version of that entity is not checked. In this case, the user shall create an application that maintains the consistency between the entity and the database.

- If the transaction contains entities for which the `Version` field is set and entities for which the `Version` field is not set, the version is only checked for the entities where the `Version` field is set. Note that the non-inclusion of a version in the entity does not affect the transaction conclusion processing.
- The user can reference the `Version` field value, but must not update the value. However, the user can update the `Version` field value during bulk update processing.

(2) Notes on using the lock method

The notes on using the `lock` method are as follows:

- The `lock` method of `EntityManager` is not supported for entities that do not have the `Version` field (property). If an entity that does not have a `Version` field (property) is specified and the `lock(entity, LockModeType.READ)` or `lock(entity, LockModeType.WRITE)` is invoked, `PersistenceException` occurs.
- When the state of an entity that has a `Version` field (property) is updated, regardless of whether the `lock` method is invoked, the dirty read and un-repeatable read events do not occur.
- When you specify `LockModeType.READ` with Cosminexus JPA Provider, the UPDATE statement is issued to check whether the database value corresponding to the entity is changed. Therefore, the UPDATE statement sets a lock for the database. The UPDATE statement is issued during the execution of `flush` processing and when the transaction is committed. The UPDATE statement is also issued when there is no change in the state of the entity object. However, the statement is not issued if the entity is deleted.

(3) Exclusive control of clients in HiRDB

The optimistic lock of Cosminexus JPA Provider is a locking method that assumes that the database Isolation level is accessed with `Read Committed`. If the database is HiRDB, the Isolation level is `Repeatable Read` by default; therefore, you must change the level to `Read Committed`.

Set the Isolation level for each client in the `PDISLLVL` parameter of the data guarantee level of the client environment variable. The default value is `Repeatable Read (2)`. Therefore, change the value to `Read Committed (1)`. An example of a change in setting is as follows:

Example of change: `PDISLLVL=1`

Specify the client environment variable in the value of the `environmentVariables` property with the *config-property* tag of the HITACHI Connector Property file or add the client environment variable in the configuration file for the client environment variable group of HiRDB.

If the data guarantee level of the client environment variable is operated with the default `Repeatable Read`, a lock is set in the shared mode. Therefore, note that if you combine the issue of reference series SQL such as the `find` method and the issue of update series SQL such as the `flush` method, a deadlock occurs easily.

6.11 Pessimistic lock in JPQL

The pessimistic lock is the method of exclusively locking the target records when multiple transactions update the same record on the database. If a transaction sets a pessimistic lock for a particular record, another transaction cannot reference or update that record (however, in Oracle, the record can be referenced). A pessimistic lock is only available when JPQL is used.

If you use a pessimistic lock, until the transaction that obtained the lock terminates, the lock awaits release. Therefore, though the concurrent executions are not more than the optimistic lock, you can prevent the transaction commit errors that occur in the optimistic lock.

How to specify a pessimistic lock

Implement the pessimistic lock by using the query hint supported by Cosminexus JPA Provider. Execute the pessimistic lock by specifying the `setHint()` method of the `Query` method or by specifying `@QueryHint` in the `@NamedQuery` attribute.

Example of implementing the pessimistic lock functionality

An example of implementing the pessimistic lock functionality is as follows:

Example of implementation 1

An example of specifying the pessimistic lock in the `setHint()` method of the `Query` method is as follows:

```
Query query = manager.createQuery("SELECT emp FROM Employee AS emp");
query.setHint("cosminexus.jpa.pessimistic-lock", "Lock");
```

Example of implementation 2

An example of specifying the pessimistic lock in `@QueryHint` of the `@NamedQuery` attribute is as follows:

```
@NamedQuery(
name="employee_list",
query="SELECT emp FROM Employee AS emp",
hints={ @QueryHint(name="cosminexus.jpa.pessimistic-lock", value="Lock") }
)
@Entity
public class Employee{
...
}
```

! Important note

The pessimistic lock is only available for JPQL. The pessimistic lock is not enabled even by specifying the `createNativeQuery` method or by specifying `@QueryHint` in the `hints` attribute of `@NamedNativeQuery`. The pessimistic lock is also not enabled by specifying the `hint` attribute of the `named-native-query` tag in the O/R mapping file. Note that the locking specifications for the pessimistic lock in Cosminexus JPA Provider conform to the specifications of the database used.

6.12 Creating an entity class

To create an application that uses the JPA, you define the entity class in the application. An entity class is used for handling the database table records as Java objects. When a user performs the `new` operation in the application, an entity class instance is generated.

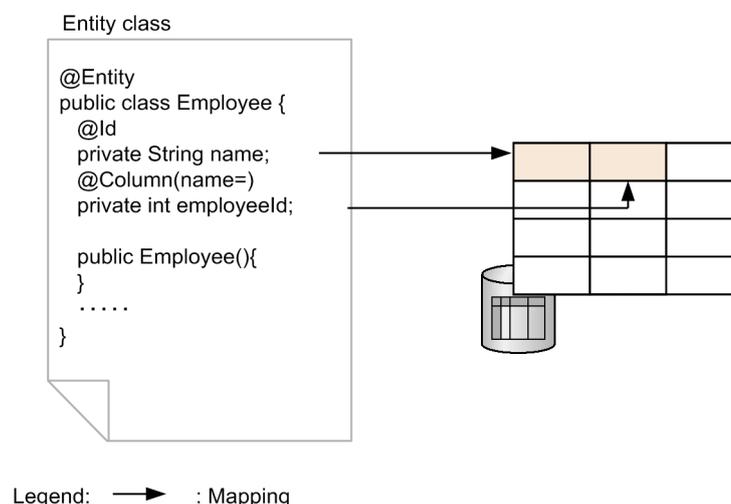
This section describes the creation of a JPA application.

6.12.1 Defining the mapping between an entity class and database

An entity class is mapped to a line in the database table. The fields stored in the entity class are mapped to the values in the table columns. If the user updates a field value for an entity class instance, Cosminexus JPA Provider also updates the corresponding column of the database table. Therefore, the user can change the database state without issuing an SQL statement for the database.

The following figure shows the mapping between an entity class and a database table.

Figure 6-22: Mapping between the entity class and database table



You define the correspondence relationship between the entity fields and database columns in an annotation or the O/R mapping file. You can define the correspondence in either an annotation or the O/R mapping file. However, if the definition is specified in both, the settings in the O/R mapping file are given priority over the annotation. If different values are specified for the same settings in the annotation and the O/R mapping file, the value in the annotation is overwritten with the value in the O/R mapping file.

6.12.2 Requirements for creating entity classes

When you create an application that uses the JPA, you must follow the requirements for creating the entity classes and the requirements for database mapping determined in the JPA specifications. The requirements for creating entity classes are as follows:

- The entity class must be specified in `@Entity` or in the `entity` tag of the O/R mapping file.
- The entity class has a constructor without an argument.
- Do not set `enum` and interfaces as entity classes.
- The entity or the mapped super-class that form the root of the class hierarchy of the entity class must have a primary key. Make sure that you define one primary key in the entity hierarchy.
- When the entity class instance is passed as a method argument with pass by value, you must implement the `Serializable` interface.

- The state of the database column is expressed by the instance variable of the entity and the instance variable corresponds to the JavaBean property. Do not change the value of the instance variable by direct access from the client. Change the value through the accessor method (*getter*/*setter* method) or the business method.
- Set the persistent instance variable of the entity to an access level that can be referenced from *private*, *protected*, or *package*.
- Declare the constructor without an argument as *public* or *protected*.
- Do not set the entity class to *final*. Also, do not set the persistence instance variable of the entity class and all the methods to *final*.

With Cosminexus JPA Provider, if these conditions are not satisfied, an exception might occur. Note that even if an exception does not occur, the operations might not function properly if an entity class that does not satisfy these conditions is created.

Furthermore, for Cosminexus JPA Provider, do not associate one database column with multiple fields in the entity class. If this condition is not satisfied, an exception might occur during the execution of the application. Even if the exception does not occur, the operations might not function properly.

6.12.3 Specifying the access methods for the entity class fields

Cosminexus JPA Provider accesses the entity class fields when the entity state is written to the database and the database state is read as an entity. The access method in this case is called *access type*. An access type includes properties and fields. You specify the access type in an annotation or in the O/R mapping file. The following table describes the access types and the specification methods.

Table 6-17: Access types and specification methods

Access type	Description	Specification method	
		Annotation	O/R mapping file
Property	Method of obtaining the instance variable through the <i>getter</i> method.	Specify the annotation in the <i>getter</i> method of the field.	Specify <i>PROPERTY</i> in the <i>access</i> tag.
Field	Method of directly referencing the instance variable.	Specify an annotation in the field.	Specify <i>FIELD</i> in the <i>access</i> tag.

If the access type is a property, the property stored by the entity is called the *persistence property*. Also, when the access type is field, the entity field is called the *persistence field*.

Notes on the access types

Remember the following points when you specify the access types:

- If the access type is a field, Cosminexus JPA Provider directly accesses the persistence field. The instance variable for which *@Transient* is not set is subject to persistence.
- If the access type is property, Cosminexus JPA Provider uses the accessor method to obtain the persistence property value. The property for which *@Transient* is not set in the accessor method is subject to persistence.
- If the access type is property, you cannot set the mapping annotation in the *setter* method. In the case of Cosminexus JPA Provider, the mapping annotation set in the *setter* method is ignored.
- You cannot set the mapping annotation in the field and property in which *@Transient* is specified or *transient* tag is specified in the O/R mapping file. If the mapping annotation is set, an exception occurs when the application starts.
- Set the accessor method of the property to *public* or *protected*. This is prohibited in the JPA specifications, but is not checked with Cosminexus JPA. Also, even in the case of *private*, an exception does not occur.
- If the mapping annotation is applied to the accessor methods of both, the persistence field and the persistence property, all the annotations set in the accessor method of the persistence property are ignored.

6.12.4 Creating the accessor method

This subsection describes the signature rules of the accessor method and the addition of the business logic to the accessor method.

(1) Method signature rules of the accessor method

When Cosminexus JPA Provider accesses a persistence property, the accessor method of the property must follow the same method signature rules as the following JavaBeans:

- `T getProperty()`
- `void setProperty(T t)`

For a property that returns the return value `boolean`, you can also change the name of the `getter` method to `isProperty`. Note that when you use Cosminexus JPA Provider, an exception occurs when the application starts if only the `getter` method or the `setter` method exists.

Also, when you handle collection values in the persistence fields and persistence properties, define the following collection values in the interface:

- `java.util.Collection`
- `java.util.Set`
- `java.util.List`
- `java.util.Map`

If the persistence property becomes a collection value, set the accessor method signature to one of these interfaces or you can also use the generic type of these collections (example: `Set<T>`).

(2) Adding the business logic to the accessor method

In addition to the `setter/getter` processing of the property, the accessor method can also include the business logic, such as verifying the values. If the access type is property, the business logic operates when Cosminexus JPA Provider invokes the accessor method.

In this case, however, note the following points:

- The order in which the accessor methods that load and store the persistent states are invoked, is not defined when Cosminexus JPA Provider is executed. Therefore, the execution order of the logic included in `getter` is not yet decided.
- For portability when the access type is property and lazy fetch is specified in the persistence property, we recommend that you do not access the entity contents until the entity contents are fetched by Cosminexus JPA Provider.
- When the access type is property and when a logic that changes the value is added as the business logic, Cosminexus JPA Provider does not guarantee data consistency.

If the `Runtime` exception occurs in the property accessor method, the current transaction is marked for rollback. If Cosminexus JPA Provider reads the persistent contents of the entity and throws an exception in the accessor method used for storing the contents, the transaction is rolled back. Also, the `PersistenceException` exception that wraps the application exception occurs.

6.12.5 Types of persistence fields and persistence properties of the entities

Set the persistence fields/ persistence properties of the entities to the following types:

- Java primitive type
- `java.lang.String`
- Other serialize types

- Primitive type wrapper class
- `java.math.BigInteger`
- `java.math.BigDecimal`
- `java.util.Date`
- `java.util.Calendar`
- `java.sql.Date`
- `java.sql.Time`
- `java.sql.Timestamp`
- User-defined serialize type
- `byte[]`
- `Byte[]`
- `char[]`
- `Character[]`
- `enum`
- Entity type and the collection duplicable in the entity type
- Embeddable class

When you use Cosminexus JPA Provider, the operations might not function properly if you specify types other than those mentioned above. An exception might also occur when the application is executed.

However, when you use Cosminexus JPA Provider, you must associate the instance variable type of the entity with the database column type. The JDBC driver associates the Java types and the database types. With Cosminexus JPA Provider, to connect to Oracle, you use Oracle JDBC Thin Driver provided by Oracle as the JDBC driver. To connect to HiRDB, you use HiRDB Type4 JDBC Driver as the JDBC driver. The user must convert the types supported by the JDBC driver as well as determine the instance variable type of the entity.

6.12.6 Specifying the primary key in the entities

With an entity, make sure that you specify the primary key in the class hierarchy. When you specify the primary key, follow the below rules:

- For a simple (not complex type) primary key, specify `@Id` in the persistence field or persistence property or specify the key in the O/R mapping file. Accordingly, associate the primary key with the entity fields.
- For a complex type primary key, specify the primary key class in a single field as `@EmbeddedId` or specify the complex primary key as a field set using `@IdClass` and `@Id`.
- For a complex type primary key, create a class that contains the primary key, called the *primary key class*.

If these conditions are not satisfied, an exception occurs when you start the application.

Also, do not change the primary key value of the entity in the application. If the primary key value is changed in the application, an exception occurs during execution.

(1) Primary key type

Set one of the following types for the simple or complex type primary keys:

- Java primitive type
- Type that wraps primitive
- `java.lang.String`
- `java.util.Date`
- `java.sql.Date`

Note that if the approximation type (for example, floating-point number type) is specified as the primary key, rounding off errors and problems such as lack of reliability in the results of the `equals` method occur in Cosminexus JPA Provider. Therefore, the operations when the approximation type is used in the primary key might not function properly with Cosminexus JPA Provider. If `java.util.Date` is used in a field/ property as the primary key, the `temporal` type attribute must be specified as the `DATE` type.

(2) Complex type primary key

The complex type primary key can be handled with an entity. The methods of specifying a complex type primary key in the entity include the method of using the embedded class and the method of using `@IdClass`. The following points describe each method:

(a) Method of using the embedded class

To use an embedded class, create a class that assigns `@Embeddable` and define the complex type primary key as a field in that class. With the entity class, define the type field of the class that assigns `@Embeddable` and annotate `@EmbeddedId`. The examples of entity class and embedded class are as follows:

- Example of an entity class

```
@Entity
public class Employee {
    private EmployeePK employeePK;

    public Employee(){
    }

    @EmbeddedId
    public EmployeePK getEmployeePK(){
        return this.employeePK;
    }

    public void setEmployeePK(EmployeePK employeePK){
        this.employeePK = employeePK;
    }
    ...
}
```

- Example of an embedded class

```
@Embeddable
public class EmployeePK {
    private String name;
    private int employeeId;

    public EmployeePK(){
    }

    public boolean equals(Object obj){
        ...
    }

    public int hashCode(){
        ...
    }
    ...
}
```

For details on the embedded class, see (3) *Embedded class*. Note that you can also use the O/R mapping file instead of the annotation.

(b) Method of using `@IdClass`

To use `@IdClass`, define multiple instance variables corresponding to the primary key in the entity class and assign `@Id`. Also, use `@IdClass` to specify the primary key class. With the primary key class, define a field or property with the same name and type as the primary key defined in the entity. The examples of entity class and primary key class are as follows:

- Example of an entity class

```

@Entity
@IdClass(EmployeePK.class)
public class Employee {
    private String name;
    private int employeeId;

    public Employee(){
    }

    @Id
    public String getName(){
        return this.name;
    }

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

    @Id
    public int getEmployeeId(){
        return this.employeeId;
    }

    public void setName(int employeeId){
        this.employeeId = employeeId;
    }
    ...
}

```

- Example of a primary key class

```

public class EmployeePK implements Serializable {
    private String name;
    private int employeeId;

    public EmployeePK(){
    }

    public boolean equals(Object obj){
        ...
    }

    public int hashCode(){
        ...
    }
    ...
}

```

As in the case of the embedded class, you can use the O/R mapping file instead of the annotation. Note that the access type of the primary key class is determined by the access type of the entity class corresponding to the primary key.

To handle a complex type primary key, use either an embedded class or `@IdClass`. However, follow the below rules:

- The primary key class must be `public` and must have a constructor without argument.
- When you use a persistence property, set the property of the primary key class to `public` or `protected`.
- The primary key class must be serializable.
- Define the `equals` and `hashCode` method with the primary key class. When the primary keys on the mapped database are equal, `true` must be returned for `equals` and the `hashCode` value must be equal.
- The complex primary key must be mapped as an embedded class or must map multiple fields/ properties of the entity class.
- When a primary key class is mapped to the complex fields/ properties of the entity class, match the field/ property name of the primary key of the primary key class with the name of the entity class. Also, unify the types.

With Cosminexus JPA Provider, the operations might not function properly if these conditions are not satisfied. If the conditions are not satisfied, an exception might also occur when you start the application.

(3) Embedded class

If you prepare a class that brings together some fields for persistence, you can store the fields as entity fields. Such a class is called an *embedded class*.

The embedded class is embedded in an entity and is mapped to the same database table as the entity. Therefore, unlike the entity, the embedded class does not have a primary key.

When using the embedded class, the user specifies `@Embeddable` in the embedded class. Also, specify `@Embedded` in the embedding destination field or property in the entity class that is embedded. Note that you can also define the embedded class similarly in the O/R mapping file instead of the annotation.

You can also use the embedded class for defining the complex type primary key. In this case, specify `@EmbeddedId` instead of `@Embedded` in the embedded entity class.

Make sure that you conform to the following creation requirements for the embedded class:

1. Make sure that the embedded class is defined in `@Embeddable` or in the `embeddable` tag of the O/R mapping file.
2. Do not set `enum` and the interfaces as an embedded class.
3. When the entity class containing the embedded class is passed by value as a detached object, implement the `Serializable` interface.
4. Do not set the embedded class, the persistence instance variables of the embedded class, and all the methods to `final`.
5. The embedded class must have a constructor without an argument.
6. Declare the constructor without an argument as `public` or `protected`.
7. The instance variable of the embedded class must be referable from `private`, `protected`, or `package`.
8. Specify settings so that the persistence instance variable of the embedded class is not accessed directly from the client. Do not access the persistence instance variable of the embedded class with the `accessor` method (`getter/ setter` method) of the entity and the other business methods.

With Cosminexus JPA Provider, if the conditions 1 and 2 are not satisfied, an exception occurs and the application fails to start. Also, in the case of conditions 3 to 8, an exception might occur, but even if an exception does not occur, the operations might not function properly.

You determine the access type of the embedded class using the access type of the entity class on the embedded side.

When using the embedded class, set one embedded hierarchy. Also, the embedded class object cannot be shared by multiple entities. If these conditions are not satisfied, the operations might not function properly with Cosminexus JPA Provider.

6.12.7 Default mapping rules for the persistence fields and persistence properties

If the O/R mapping information is not specified for the persistence fields or persistence properties other than relationship, the following default mapping rules are applied:

- For a class in which `@Embeddable` is annotated, the field/ property is mapped to the database according to the specification in the entity in `@Embedded`.
- If the persistence field/ persistence property type is one of the following, the mapping method is the same as when `@Basic` is defined:
 - Java primitive type
 - Primitive type wrapper
 - `java.lang.String`
 - `java.math.BigInteger`
 - `java.math.BigDecimal`
 - `java.util.Date`
 - `java.util.Calendar`
 - `java.sql.Date`
 - `java.sql.Time`

- `java.sql.Timestamp`
- `byte[]`
- `Byte[]`
- `char[]`
- `Character[]`
- `enum`
- Any type that implements `Serializable`

Note that the operations might not function properly if types other than those mentioned above are specified.

6.13 Procedure for inheriting an entity class

The inheritance of an entity class has the following features:

- An entity class can use either an abstract class or a concrete class. Apart from the abstract class and concrete class, you can also define `@Entity`. Furthermore, both abstract class and concrete class can be mapped as entities and queries can be issued for both the classes.
- An entity class can be inherited from another entity class.
- An entity class can inherit a non-entity class. A non-entity class can also inherit an entity class.

This section describes the inheritance class types and the inheritance mapping strategy of the entity classes.

6.13.1 Inheritance class types

The inheritance class types of an entity class include the abstract entity class, mapped super-class, and non-entity class. The following is a description of each type:

(1) Abstract entity class

You can define an abstract class as an entity class. An abstract entity class differs from the concrete entity class in that an abstract entity class cannot create an instance directly. An abstract entity class can be mapped as an entity and can also be specified as a query target for operating/ obtaining the subclass entity.

With a subclass, the accessor method of the property can be overridden. However, if the O/R mapping information on the persistence fields and persistence properties is overridden in a subclass, the operations might not function properly. To override the O/R mapping information in a subclass, use `@AssociationOverride` or `@AttributeOverride`.

You specify the abstract entity class in `@Entity` or the O/R mapping file.

(2) Mapped superclass

A mapped superclass is a class that can become the superclass of an entity class. You can define the persistence fields and mapping information and also set a configuration to inherit the persistence fields and mapping information.

A mapped superclass cannot specify a specific table, so `@Table` cannot be specified. Therefore, the mapped superclass cannot be set as an entity. Unlike the entity class, a mapped superclass cannot issue a query, so the mapped superclass cannot be passed to the arguments of `EntityManager` and query operations. Also, a mapped superclass cannot be set as a relationship target.

A mapped superclass can be defined as an abstract class as well as a concrete class. To define a class as a mapped superclass, define `@MappedSuperclass` or *mapped-superclass* tag in the O/R mapping file. The mapping information is provided for the inherited entity class.

A class designed as a mapped superclass can be mapped using the same method as the entity class, except when the mapping can only be provided for the subclasses.

If an entity is applied as a subclass, the information specified in the mapped superclass is inherited in the subclass. By using `@AssociationOverride` or the corresponding XML element in the O/R mapping file, the mapping information can be overridden in the subclass.

(3) Non-entity class of the entity inheritance hierarchy

An entity class can have a superclass of a non-entity class. A superclass can be defined as a concrete class and abstract class.

A superclass of a non-entity class has the following features:

- Only the behavior is inherited.
- The state is not persistent.
- All the inherited states are not persistent even in the inherited entity classes.

- A non-persistent state is not controlled by `EntityManager`.
- The annotation of a superclass is ignored.

Do not set a non-entity class in the method arguments of `EntityManager` and the `Query` interface and in the mapping information. In Cosminexus JPA Provider, an exception occurs during the execution of the application.

6.13.2 Inheritance mapping strategy

When an entity is inherited, you can specify the method of mapping the class hierarchy to a table as the inheritance mapping strategy. You specify the inheritance mapping strategy by using `@Inheritance` or the *inheritance* tag of the *entity* tag in the O/R mapping file.

There are three types of inheritance mapping strategies:

- **SINGLE TABLE strategy**
The SINGLE TABLE strategy is a strategy method of mapping all the classes existing in the inheritance hierarchy of the entity class to one table.
- **JOINED strategy**
The JOINED strategy is a strategy method of mapping the top-level of the class hierarchy to a single table.
- **TABLE PER CLASS strategy**
The TABLE PER CLASS strategy is a strategy method of mapping each class existing in the class hierarchy of the entity class to separate tables.

However, with Cosminexus JPA Provider, the TABLE PER CLASS strategy is not available. If the TABLE PER CLASS strategy is used when Cosminexus JPA Provider is used, an exception occurs when the application starts.

! Important note

When you use Cosminexus JPA Provider, note the following points:

- You cannot combine and specify multiple inheritance strategies in a class hierarchy. Also, no check is performed to verify whether multiple inheritance strategies are combined. If multiple inheritance strategies are specified, the operations might not function properly.
- If the column specified in `@DiscriminatorColumn` is defined in an entity field, the value set in the entity field is not applied to the database even if the `persist` operation and then `commit` is executed. The value specified in `@DiscriminatorValue` or the default value is applied. Also, note that after `commit`, the value set in the field before `commit` is stored as is.

These strategies are specified in the value of the enumeration type `javax.persistence.DiscriminatorType`. The following points describe each strategy:

(1) SINGLE TABLE strategy

The SINGLE TABLE strategy is a strategy method of mapping all the classes present in the inheritance hierarchy of the entity class to one table. Therefore, the table must have an identification column as the column for identifying the class.

Specify the identification column in `@DiscriminatorColumn` or in the O/R mapping file. The default identification column name is `DTYPE`. If an identification column does not exist in the database, an exception occurs during the execution of the application.

If you want to specify the value stored in the identification column, use `@DiscriminatorValue` or the *discriminator-value* tag beneath the *entity* tag of the O/R mapping file.

! Important note

When you use the SINGLE TABLE strategy, the user must be able to specify a null value in the table column corresponding to the subclass field.

(2) JOINED strategy

The JOINED strategy is a strategy method of mapping the top-level of the class hierarchy to a single table. Each subclass is indicated by the subclass-specific fields that are not inherited from the superclass and by different tables with the primary key string that functions as the external key of the primary key for the superclass table.

In the case of the JOINED strategy, like in the case of the SINGLE TABLE strategy, the table to which the superclass is mapped must have an identification column.

Important note

When using the JOINED strategy, binding must be executed multiple times during the generation of the subclass instances. Therefore, if the hierarchy structure becomes deep, the performance might deteriorate. Also, JOIN is necessary for issuing queries across the class hierarchy.

6.14 Procedure for using EntityManager and EntityManagerFactory

This section describes the procedure for using `EntityManager` and `EntityManagerFactory` that are used from the application.

6.14.1 Entity lifecycle management with EntityManager

`EntityManager` is an object with an interface for executing the following operations for the database:

- Registering and deleting entities.
- Searching entities using the primary keys.
- Issuing a query across the entities.

If you register an entity for `EntityManager`, the entity state is perpetuated in the database at an appropriate time such as when the transaction is committed.

Also, `EntityManager` has a relation with the persistence context that expresses the entity set. When you register the entity in `EntityManager`, the entity belongs to a specific persistence context. Also, `EntityManager` manages the entity lifecycle.

The entity set managed by `EntityManager` is defined with a unit called the *persistence unit*. You define the persistence unit in the application configuration file `persistence.xml`.

The notes on the persistence context and persistence unit are as follows:

- The entity must be unique in the persistence context. Therefore, set one entity expressing the same line of the database in the same persistence context. Note that if the persistence contexts are different, you can have multiple entities expressing the same line in the database. For the locking method in the database in this case, see *6.10 Optimistic lock* or *6.11 Pessimistic lock in JPQL*.
- Each persistence unit is mapped to a single database. For details on the definition, see *5.8 Definitions in persistence.xml*.

6.14.2 How to set up EntityManager and EntityManagerFactory

You set up the `EntityManager` and `EntityManagerFactory` you want to use in an application in an annotation or the DD.

- **EntityManagerFactory settings**
 - In an annotation: Specify the settings in `@PersistenceUnit`.
 - In the DD: Specify the settings in the *persistence-unit-ref* tag.
- **EntityManager settings**
 - In an annotation: Specify the settings in `@PersistenceContext`.
 - In the DD: Specify the settings in the *persistence-context-ref* tag.

For details on annotations, see *2. Annotations and Dependency Injection Supported by Application Server* in the *uCosminexus Application Server API Reference Guide*. For details on the tags to be set for DD, see the *uCosminexus Application Server Definition Reference Guide*.

6.14.3 Notes on the API functions of EntityManager

This subsection describes the notes on the API functions provided by `EntityManager`. For details on the `EntityManager` APIs, see *2.7 javax.persistence Package* in the *uCosminexus Application Server API Reference Guide*.

- When `EntityManager` of transaction scope persistence context is used, the `persist`, `merge`, `remove`, and `refresh` methods must be executed in the transaction context. If the transaction context does not exist, `javax.persistence.TransactionRequiredException` is thrown.
- The `find` method and `getReference` method do not require execution in the transaction context. Therefore, if `EntityManager` of transaction scope persistence context is used, the resulting entity has a detached state. Also, if `EntityManager` of extended persistence context is used, the resulting entity has a managed state.
- If the argument of the `createQuery` method is not a valid JPQL string, the `IllegalArgumentException` exception is sent and the execution of the query fails.
- If the executed native query does not match the specifications for the database to be connected to or if the defined result set is not compatible with the query results, the execution of the query fails and the `PersistenceException` exception is thrown when the query is executed.
- If a runtime exception is sent from a method of the `EntityManager` interface, the current transaction is marked for rollback.
- The `Query` object obtained from `EntityManager` and the `EntityTransaction` object are enabled while `EntityManager` is open.

6.15 Procedure for specifying the callback method

To receive an entity lifecycle, you can specify a method in the lifecycle callback method. The method defined as a callback method is invoked corresponding to the persistence-related lifecycle event.

This section describes the location for specifying the callback method, the implementation methods, and the order of invocation.

6.15.1 Location for specifying the callback method

You can specify the callback method at the following locations:

- In the entity class or mapped superclass
- Entity listener class associated with the entity class or mapped superclass

The entity listener class is a dedicated class for implementing the callback method. If you use the entity listener class, you can separate the parts for implementing the callback method.

Specify the callback method in an annotation or the O/R mapping file. However, the default callback method is specified in the O/R mapping file and cannot be specified in an annotation. Note that the default callback method indicates an entity listener applied to all the entities in a persistence unit.

This subsection describes how to specify the callback listener.

(1) Specifying the callback method in an annotation

If you use an annotation for specifying the callback method, set the annotations listed and described in the following table in the method. You can invoke the method in compliance with a lifecycle event.

Table 6-18: Specifying the callback method using annotations

Annotation	Executed contents
<code>@PostLoad</code>	The callback method is executed after the entity is loaded in the persistence context or after the <code>refresh</code> operation is applied. The method is executed after the entity is read from the cache or the <code>SELECT</code> statement is issued for the database.
<code>@PrePersist</code> <code>@PreRemove</code>	The callback method is invoked before <code>EntityManager</code> executes the <code>persist</code> or <code>remove</code> operation of the entity. When the merge operation is applied and a new <code>managed</code> instance is created, the <code>PrePersist</code> callback method is invoked for the <code>managed</code> instance after the entity state is copied. The <code>PrePersist</code> or <code>PreRemove</code> callback methods are also invoked for all the instances where the operations are cascaded. The <code>PrePersist</code> or <code>PreRemove</code> method is always invoked synchronously as a part of the <code>persist</code> , <code>merge</code> , or <code>remove</code> operations.
<code>@PostPersist</code> <code>@PostRemove</code>	The <code>PostPersist</code> and <code>PostRemove</code> callback methods are invoked after the entity is perpetuated or deleted by the <code>persist</code> or <code>remove</code> operations. These callback methods are also invoked for all the entities where the operations are cascaded. The <code>PostPersist</code> or <code>PostRemove</code> methods are respectively invoked after the database <code>insert</code> or <code>delete</code> operations are performed. These database operations are executed immediately after the <code>persist</code> , <code>merge</code> , or <code>remove</code> operations or after the <code>flush</code> method is invoked. However, the database operations may also be executed at the end of a transaction. The generated primary key can be used with the <code>PostPersist</code> method.
<code>@PreUpdate</code> <code>@PostUpdate</code>	The <code>PreUpdate</code> and <code>PostUpdate</code> callback methods are respectively invoked before and after the database update operation of the entity data. These database operations are executed when the entity state is updated or when the state is flushed in the database. However, the database operations might also be executed at the end of a transaction. When an entity is perpetuated and then updated and when an entity is updated and then deleted in one transaction, the <code>PreUpdate</code> and <code>PostUpdate</code> callback might not occur.

To use the entity listener class, you must specify the entity listener class by specifying `@EntityListener` for the entity. The following is an example of specifying an entity listener class:

```

@Entity
@EntityListeners(CallbackListener.class)
public class Employee implements Serializable{
    ...
}

```

(2) Specifying the callback listener in the O/R mapping file

Specify the following settings to use the O/R mapping file for specifying the callback method:

- To specify the entity listener class and the callback method of that class, use the *entity-listener* tag of the O/R mapping file. Specify the lifecycle listener method by using the *pre-persist* tag, *post-persist* tag, *pre-remove* tag, *post-remove* tag, *pre-update* tag, *post-update* tag, and *post-load* tag beneath the *entity-listener* tag.
- When you specify the callback method of the entity listener class, you can specify maximum one method for each callback event by using the tags beneath the *entity-listener* tag.
- If you specify the *entity-listener* tag of the O/R mapping file for the lower tags of the *entity-listeners* tag in the *persistence-unit-defaults* tag, you can specify the default callback method.
- If you specify the *entity-listener* tag in the lower tags of the *entity-listeners* tag existing in the *entity* tag or *mapped-subclass* tag, the callback listener is specified for the entity or mapped superclass and the subclasses.
- The callback listener is invoked in the order of listeners specified in the *entity-listeners* tag. For details on the order for invoking the listeners, see 6.15.3 *Order of invoking the callback methods*.

6.15.2 Implementing the callback methods

The user implements the callback method as and when required. The callback method signature differs in the callback method implemented in the entity class and mapped superclass and in the callback method of the entity listener class.

The callback method defined in the entity class and mapped superclass has the following signature:

```
void <METHOD>
```

The callback method defined in the entity listener class has the following signature:

```
void <METHOD>(Object)
```

In the argument `Object`, you specify the entity instance in which the callback method is executed.

(1) Notes on using the callback methods

Note the following regarding the callback methods. If the following conditions are not satisfied, an exception occurs at application startup and the application fails to start:

- A constructor without argument must be specified in `public`.
- The `public`, `private`, `protected`, and package level access is allowed with the callback method. However, `static` and `final` are not available.
- One class cannot have multiple lifecycle callback methods for the same lifecycle event. However, the same method might be used in multiple callback events.

(2) Rules applied to the callback methods

The following rules are applied to the callback methods:

- With the callback method, the issuing of unchecked or runtime exceptions are allowed. The runtime exception thrown by the callback method executed in the transaction rolls back the transaction. If multiple callback methods are specified, after the runtime exception is thrown, the remaining callback methods are not executed.
- With the callback method, you can execute JNDI, JDBC, JMS, and Enterprise Bean.
- Do not perform the following operations with a callback method:

- Invoking `EntityManager`.
- Executing a query operation.
- Accessing other entity instances.
- Updating a relationship.

If a callback method is used for such methods, the operations might not function properly.

- If the callback method is invoked in the Java EE environment, the entity callback listener shares the naming context of the components to be invoked. Furthermore, the callback method of the entity is invoked in the transaction and in the security context of the invocation source components used when the callback method is invoked.

6.15.3 Order of invoking the callback methods

If multiple callback methods are defined for an entity, the invocation order follows the below rules:

1. The default listeners are invoked in the order defined in the O/R mapping file.
Unless `@ExcludeDefaultListeners` or the `exclude-default-listeners` tag of the O/R mapping file is explicitly specified, the default listeners are applied to all the entities in the persistence unit.
2. The callback methods are invoked in the order specified in `@EntityListeners`.
Note that if you use the O/R mapping file, you can execute the following operations:
 - Specifying the order of invoking the callback methods for the entities.
 - Overriding the order specified in the annotations.
3. The callback method specified in the entity (or mapped superclass) is invoked.

(1) Invocation order in the inheritance hierarchy

If the entity listener is defined multiple times in the inheritance hierarchy of the entity class and mapped superclass, the invocation order is as follows:

1. The default callback listener, if present, is invoked first.
2. The callback methods of the entity listener class are invoked sequentially from the listener specified in the superclass. At this time, if items such as `@EntityListener` are specified, that order is followed.
3. After the callback methods of all the entity listeners are invoked, the callback methods defined in the entity (or mapped superclass) are invoked sequentially from the listener specified in the superclass.

If the callback method is overridden in the subclass, the overridden method is not invoked. If the overridden callback method specifies different lifecycle events or if the overridden callback method is not a lifecycle callback method, the overridden method is invoked. Also, the callback method settings of the method are overridden.

(2) Excluding the callback methods

- If you specify `@ExcludeDefaultListeners` or the `exclude-default-listeners` tag of the O/R mapping file, the default entity listener is not invoked in the entity class (or mapped superclass) and the subclasses.
- If `@ExcludeSuperclassListeners` or the `exclude-superclass-listeners` tag of the O/R mapping file is applied to the entity class and mapped superclass, the listener callback method is not invoked in that class and the subclasses. `@ExcludeSuperclassListeners` or the `exclude-superclass-listeners` tag of the O/R mapping file does not exclude the invocation of the default entity listener.
- If you use the O/R mapping file to explicitly specify the default or superclass listener excluded for the entity and mapped superclass, the default or superclass listener is applied to the entity and the subclasses.

6.16 Procedure for referencing and updating the database with the query language

A query expresses a processing request (inquiry) for a database as a string. A query is used to issue commands such as commands to search, update, and delete the data in a database to a system. To execute a query, use the `javax.persistence.Query` interface. The types of query include JPQL and native query. This section describes the procedure for referencing and updating the database with a query.

6.16.1 Procedure for referencing and updating the database with JPQL

JPQL is a query language used for searching and updating the database and for using the database functionality such as the set function. While the SQL is a query language using a table as the target, JPQL is a query language defined in the JPA specifications using an entity class as the target.

You can define a query in an annotation or the O/R mapping file. If an entity is defined in the same persistence unit as the query, you can use the abstract schema type expressing the entity set in the query. Also, if you use the path expression, you can use a query across the relationship defined in the persistence unit. For details on the path expression, see *6.17.4(2) Path expression*.

If you code and execute JPQL in an application, the SQL statements are issued for the database to be connected to in the following order:

1. If JPQL is executed, Cosminexus JPA Provider interprets the JPQL contents.
2. Based on the annotation and O/R mapping file information coded in the target entity class, JPQL is set up in the SQL statements specific to the database product to be connected to and then issued.

This subsection describes how to use JPQL.

(1) How to obtain the Query object

To use JPQL for obtaining the `Query` object, use the following methods of the `EntityManager` interface provided by Cosminexus JPA Provider. This subsection describes the methods of the `EntityManager` interface.

(a) Query `createQuery(String JPQL statement)`

An example of coding `createQuery` is as follows. In the argument, specify the JPQL statement you want to execute.

```
Query q = em.createQuery(
    "SELECT c " +
    "FROM Customer c " +
    "WHERE c.name LIKE 'Smith');
```

(b) Query `createNamedQuery(String query name)`

A query that can be given a name and defined in advance is called a named query. You define a named query by assigning `@NamedQuery` in any entity class. Specify the query name in the `name` attribute of `@NamedQuery` and then specify the JPQL statement in the `query` attribute.

In Cosminexus JPA Provider, you cannot specify multiple named queries with the same name. If multiple named queries with the same name are specified, a warning message KDJE55535-W is output. If such multiple named queries with the same name are specified in Cosminexus JPA Provider, there is no certainty about which query will be operated.

The following is an example of defining `@NamedQuery`. In this example, `@NamedQuery` is used and the query is registered beforehand with the name `findAllCustomersWithName`. By passing the named query name registered in the `createNamedQuery` method of the application, the query registered beforehand is obtained and used.

- Registering the query name in `@NamedQuery`

```

@NamedQuery(
    name="findAllCustomersWithName",
    query="SELECT c FROM Customer c WHERE c.name LIKE :custName"
)
@Entity
public class Customer {
    ...
}

```

- Example of coding the named query with the `createNamedQuery` method

```

@Stateless
public class MySessionBean {
    ...
    @PersistenceContext
    public EntityManager em;
    ...
    public void doSomething() {
        ...
        Query q = em.createNamedQuery("findAllCustomersWithName")
                    .setParameter("custName", "Smith");
    }
}

```

Note that with the same persistence unit, you can also use the named query defined in another entity.

(2) How to specify the parameters

When you generate a query with JPQL, you can use a parameter in the conditional expression coded in the `WHERE` clause and set the value dynamically. Set the parameter value with the `setParameter` method of the `Query` interface. The parameters include location parameters and named parameters. The following is a description of each parameter:

Location parameters

Code a combination of '?' (question mark) and a numeric value in the location where you want to insert the parameter in the `WHERE` clause. Specify the parameter value with the `setParameter` method of the `Query` interface. The format and example of coding the location parameters is as follows:

- Coding format

```
Query setParameter(int location, Object value)
```

- Coding example

```

Query q = em.createQuery(
    "SELECT c FROM Customer c WHERE c.balance < ?1")
    .setParameter(1, 20000);

```

Named parameters

Code a combination of ':' (colon) and any string (however, excluding the characters 0 to 9) in the location where you want to insert the parameter in the `WHERE` clause. Specify the parameter value with the `setParameter` method of the `Query` interface. The format and example of coding the named parameters is as follows:

- Coding format

```
Query setParameter(String parameter-name, Object value)
```

Do not specify ':' (colon) at the beginning of the parameter name. Also, the named parameters are case sensitive.

- Coding example

```

Query q = em.createQuery(
    "SELECT c FROM Customer c WHERE c.name LIKE :custName")
    .setParameter("custName", "John");

```

Note the following when you use parameters:

- Do not mix and use the location parameters and named parameters in one query. In Cosminexus JPA Provider, the operations might not function properly if the parameters are mixed.

- In the `setParameter` method, specify as follows when you want to specify the `java.util.Date` type or `java.util.Calendar` type objects as parameter values. Note that the time type of the parameter value must be specified using the enumeration type `TemporalType`. For details, see the *Java documentation*.
 - Query `setParameter(int location, Date date, TemporalType time-type)`
 - Query `setParameter(String parameter-name, Date date, TemporalType time-type)`
 - Query `setParameter(int location, Calendar calendar, TemporalType time-type)`
 - Query `setParameter(String parameter-name, Calendar calendar, TemporalType time-type)`

(3) Obtaining and executing the query results

You use the following methods of the `Query` interface to execute the generated query, to return the query results, and to execute update query. The following points describe each method:

(a) Object `getSingleResult()`

Use this method to return the query result as a single object.

When you execute this method, the data is searched. As a result of the search, the single hit line is stored in the entity object and returned with the `Object` type. The return value of the `Object` type must be cast in the target entity class.

If multiple lines are hit, the `NonUniqueResultException` exception occurs. If no lines are hit, the `NoResultException` exception occurs.

(b) List `getResultList()`

Use this method to return the query result as a list.

When you execute this method, the data is searched. As a result of the search, the multiple hit lines are stored in the entity object and returned in a list. Multiple lines are assumed to be returned as execution results; therefore, if no lines are hit, an empty list is returned.

(c) int `executeUpdate()`

Use this method to execute the update query.

When you execute this method, a query will be executed to simultaneously delete or update multiple lines in a table. The execution result returns the number of lines hit.

6.16.2 Procedure for referencing and updating the database with the native query

With Cosminexus JPA Provider, as a query language other than JPQL, you can directly code a database-specific native query and reference or update a database.

This subsection explains how to use native queries.

(1) How to obtain the Query object

To use native query for obtaining the `Query` object, you use the following methods of the `EntityManager` interface provided by Cosminexus JPA Provider.

(a) Query `createNativeQuery(String SQL statement)`

An example of coding `createNativeQuery` is as follows. In the argument, specify the native query you want to execute.

```
Query q = em.createNativeQuery(
    "SELECT o.id, o.quantity, o.item " +
    "FROM Order o, Item i " +
    "WHERE (o.item = i.id) AND (i.name = 'widget')");
```

(b) Query createNativeQuery(String SQL statement, Class result storing class)

An example of coding `createNativeQuery` is as follows. Specify the native query you want to execute in the first argument and the class object for storing the execution result in the second argument.

```
Query q = em.createNativeQuery(
    "SELECT o.id, o.quantity, o.item " +
    "FROM Order o, Item i " +
    "WHERE (o.item = i.id) AND (i.name = 'book')",
    com.hitachi.Order.class);
```

In this example, if a query is executed, the collection of all the `Order` entities is returned for the item named 'book'.

Note that if the query results specified in the `SELECT` clause and the class object specified in the argument are inconsistent, an exception occurs.

(c) Query createNativeQuery(String SQL statement, String result set mapping name)

Specify the native query you want to execute in the first argument and the result set mapping name for storing the execution result in the second argument. Specify the result set mapping with `@SqlResultSetMapping`. For details on result set mapping, see (2) *Result set mapping*.

An example of defining `@SqlResultSetMapping` and an example of coding `createNativeQuery` are as follows:

- Example of defining `@SqlResultSetMapping`

```
@SqlResultSetMapping(name="BookOrderResults",
    entities=@EntityResult(entityClass=com.hitachi.Order.class))
```

- Example of coding `createNativeQuery`

```
Query q = em.createNativeQuery(
    "SELECT o.id, o.quantity, o.item " +
    "FROM Order o, Item i " +
    "WHERE (o.item = i.id) AND (i.name = 'book')",
    "BookOrderResults");
```

In this example, if a query is executed, the collection of all the `Order` entities is returned for the item named 'book'.

By using `@SqlResultSetMapping`, you can obtain the same results as the coding example when `@NamedQuery` is used in 6.16.1(1) *How to obtain the Query object*.

Note that if the query results specified in the `SELECT` clause and the `@SqlResultSetMapping` settings specified in the argument are inconsistent, an exception occurs.

(d) Query createNamedQuery(String query name)

Like JPQL, you can use the `createNamedQuery` method for a native query. In the native query, specify the named native query name in the argument.

You define the named native query by assigning `@NamedNativeQuery` in any entity class. In the query name argument, use the name specified in the `name` attribute of `@NamedNativeQuery`.

In Cosminexus JPA Provider, you cannot specify multiple named queries with the same name. If multiple named queries with the same name are specified, a warning message KDJE55522-W is output. If such multiple named queries with the same name are specified in Cosminexus JPA Provider, there is no certainty about which query will be operated.

The following is an example of using the `createNamedQuery` method. In this example, `@NamedNativeQuery` is used and the query is registered beforehand with the name `findBookOrder`. By passing the named query name registered in the `createNamedQuery` method of the application, the query registered beforehand is obtained and used.

- Registering the query name in `@NamedNativeQuery`

```
@NamedNativeQuery( name="findBookOrder",
    query="SELECT o.id, o.quantity, o.item " +
    "FROM Order o, Item i " +
    "WHERE (o.item = i.id) AND (i.name = 'book')")
)
```

```
@Entity
public class Order {
    ...
}
```

- Example of coding the named native query with the `createNamedQuery` method
-

```
@Stateless
public class MySessionBean {
    ...
    @PersistenceContext
    public EntityManager em;
    ...
    public void doSomething() {
        ...
        Query q = em.createNamedQuery("findBookOrder ");
    }
}
```

Note that with the same persistence unit, you can also use the named native query defined in another entity.

(2) Result set mapping

Result set mapping is a functionality used for mapping and receiving the execution results of the native query in any entity class and for receiving the execution results of the native query using the scalar value.

With result set mapping, the mapping information of the column values obtained as the execution results of the native query is assigned to any entity class by specifying `@SqlResultSetMapping`.

(a) Coding format of `@SqlResultSetMapping`

The coding format of `@SqlResultSetMapping` is as follows:

```
@SqlResultSetMapping(
    name= Result-set-mapping-name,
    entities= Specify-the-entity-class-for-mapping-the-result-(@EntityResult-array),
    columns= Specify-the-column-for-mapping-the-result-(@ColumnResult-array) )
```

name **attribute**

Specify the result set mapping name.

entities **attribute**

Specify the `@EntityResult` array. The coding format of `@EntityResult` is as follows:

```
@EntityResult(
    entityClass= Specify-the-class-for-mapping-the-result,
    fields= Specify-the-field-for-mapping-the-result-(@FieldResult-array) )
```

In the `entityClass` attribute of `@EntityResult`, specify the entity class for storing the column value. In the `fields` attribute, specify the `@FieldResult` array.

The coding format of `@EntityResult` is as follows:

```
@FieldResult(
    name= Name-of-persistent-property-(or-field)-of-class,
    column= Column-name-of-SELECT-clause-(or-optional-name) )
```

In the `name` attribute of `@FieldResult`, specify the persistence field name of the entity class specified in the `entityClass` attribute of `@EntityResult`. Also, in the `column` attribute, specify the column name.

columns **attribute**

The `columns` attribute specifies the `@ColumnResult` array to receive the execution results of the native query as a scalar value without being stored in the entity class. If you do not need to extract the scalar value, you need not specify the `columns` attribute. In the `name` attribute of `@ColumnResult`, specify the column name for extracting the value. The coding format of `@ColumnResult` is as follows:

```
@ColumnResult(
    name= Column-name-of-SELECT-clause-(or-optional-name) )
```

Note that you can also specify the column name with the alias name specified by AS. If the SELECT clause contains multiple columns with the same name, use the optional name of the column.

(b) Example of usage

In the following example, the query result for employee number 12003 is mapped from the employee table (Employee) and department table (Department) to any entity class (EmployeeSetmap) and the scalar value (EMP_MONTHLY_SALARY column) is received.

Specify the result set mapping name (NativeQuerySetMap) in the second argument of createNativeQuery and execute the result set mapping.

- Example of coding the native query using result set mapping

```
query = em.createNativeQuery(
    "SELECT e.EMPLOYEE_ID AS EMP_EMPLOYEE_ID, " +
    "e.EMPLOYEE_NAME AS EMP_EMPLOYEE_NAME, " +
    "d.DEPARTMENT_NAME AS DEP_DEPARTMENT_NAME, " +
    "e.MONTHLY_SALARY AS EMP_MONTHLY_SALARY " +
    "FROM EMPLOYEE e, DEPARTMENT d " +
    "WHERE e.DEPARTMENT_ID = d.DEPARTMENT_ID " +
    "AND e.EMPLOYEE_ID = 12003",
    "NativeQuerySetMap");
```

- Optional entity class for storing the execution result of the native query

```
@Entity
public class EmployeeSetmap implements Serializable {
    ...
    @Id
    public int getEmployeeId() { return employeeId; }
    public String getEmployeeName() { return employeeName; }
    public String getDepartmentName() { return departmentName; }
    ...
}
```

- Example of coding @SqlResultSetMapping

```
@SqlResultSetMapping(
    name="NativeQuerySetmap",
    entities={ @EntityResult(
        entityClass=EmployeeSetmap.class,
        fields={ @FiledResult(
            name="employeeId",
            column="EMP_EMPLOYEE_ID"),
            @FiledResult(
            name="employeeName",
            column="EMP_EMPLOYEE_NAME"),
            @FiledResult(
            name="departmentName",
            column="DEP_DEPARTMENT_NAME") } ) },
    columns={ @ColumnResult(
        name="EMP_MONTHLY_SALARY") }
```

Note that the Object type array for the result of executing the native query by executing @SqlResultSetMapping is as follows:

```
Object[0]
    Object of EmployeeSetmap class
    (Values of EMP_EMPLOYEE_ID column, EMP_EMPLOYEE_NAME column, and
    DEP_DEPARTMENT_NAME column are stored in each field)
Object[1]
    Value of EMP_MONTHLY_SALARY column
```

(3) How to specify the parameters

As in the case of JPQL, the native query can set a value dynamically by using parameters. Code a combination of '?' (question mark) and a numeric value in the location where you want to insert the parameter in the WHERE clause. Specify the parameter value with the setParameter method of the Query interface. However, you cannot use the named parameters of JPQL with the native query.

The coding format of the parameters is as follows:

```
Query setParameter(int location, Object value)
```

(4) Obtaining and executing the native query results

As in the case of JPQL, use the following `Query` interface methods for obtaining and executing the native query results:

- `Object getSingleResult()`
- `List getResultList()`
- `int executeUpdate()`

For details on these methods, see *6.16.1 Procedure for referencing and updating the database with JPQL*.

6.16.3 Specifying the range of query result items

You can obtain only an optionally specified item from multiple query results and only the query result specified in the starting position. To obtain this information, use the `Query` interface methods. This subsection describes the methods.

(1) setMaxResults method

To obtain only an optionally specified item from multiple query results, use the `setMaxResults` method of the `Query` interface. The coding format of the `setMaxResults` method is as follows:

```
Query setMaxResults(int maximum-number-of-search-results)
```

In the method argument, specify the maximum number of search results.

(2) setFirstResult method

To obtain only the query result specified in the starting position, use the `setFirstResult` method. The coding format of the `setFirstResult` method is as follows:

```
Query setFirstResult(int starting-position-of-search-results)
```

In the method argument, specify the starting position of the search results. Specify a numeric value beginning with 0 as the starting position.

(3) Example usage of the Query interface methods

The examples of usage of the `setMaxResults` method and `setFirstResult` method are as follows. In this example, five `Employee` objects are obtained sequentially from the tenth employee with the highest monthly salary (`e.monthlySalary`) from the employee data (`Employee`).

```
Query query = em.createQuery( "SELECT e FROM Employee AS e " +
                             "ORDER BY e.monthlySalary DESC" )
    .setFirstResult(9)
    .setMaxResults(5);
List resultList = query.getResultList();
```

(4) Notes

When you use the `setFirstResult` method to specify the starting position, the time period from the invocation of the `getResultList` method and `getSingleResult` method until the obtaining of the result values varies depending on the value specified in the argument. Typically, the time taken until the result value returns is in proportion to the value of the starting position specified in the argument.

6.16.4 Specifying the flush mode

You can specify how the query would handle an uncommitted operation executed for an entity object. This is called *specifying the flush mode*.

You set the flush mode with the `setFlushMode` method of the `javax.persistence.Query` interface. With Cosminexus JPA Provider, you can only set `AUTO` as the value. You cannot specify `COMMIT`.

In `FlushModeType.AUTO`

When a query is executed in a transaction, the changes in all the entities existing in the persistence context that affect the query results are applied to the query results.

This setting is applied to a query regardless of the flush mode of the `setFlushMode` method of the `EntityManager` interface.

6.16.5 Specifying a query hint

During the execution of a query, you can specify a query hint as a vendor-dependent hint. With Cosminexus JPA Provider, you specify a query hint when you use the pessimistic lock.

Specify the query hint at the following locations:

- The argument of the `setHint()` method of the `Query` object
- The argument `@Hint` of `@NamedQuery`
- The `hint` tag that is a lower element of the `named-query` tag in the O/R mapping file

If you specify a value outside the range specifiable in the query hint, an exception occurs. Note that the specified value is not case sensitive.

The time when an exception occurs differs depending on the location where the query hint is specified. The timing for the occurrence of an exception is as follows:

- In the `setHint()` method, when the application query is executed
- In the annotation, when the application is deployed
- In the O/R mapping file, when the application starts

For details on the query hints supported by Cosminexus JPA Provider, see *2.1 Range of supported annotations* in the *uCosminexus Application Server API Reference Guide* if you use annotations, and *6.3 O/R mapping file* in the *uCosminexus Application Server Definition Reference Guide* if you use the O/R mapping file.

6.16.6 Notes on executing a query

This subsection describes the notes on executing a query.

- In the `setMaxResults` method or `setFirstResult` method, if a query in which collection contains `FETCH JOIN` is executed, the results might be incorrect.
- The `Query` methods other than the `executeUpdate` method need not be executed in a transaction. Particularly, the `getResultList` and `getSingleResult` methods need not be executed in a transaction.
- When the query is executed with `EntityManager` of the transaction scope persistence context, the resulting entity has the `detached` state. When the query is executed with `EntityManager` of the extended persistence context, all the entities have the `managed` state.
- The runtime exceptions other than `NoResultException` and `NonUniqueResultException` thrown from the `Query` interface methods roll back the current transaction.

6.17 JPQL coding method

This section describes JPQL coding method.

6.17.1 JPQL syntax

A JPQL statement includes the `SELECT` statement, `UPDATE` statement, and `DELETE` statement.

You can dynamically specify a JPQL statement or statically define a JPQL statement using the annotation and O/R mapping file tags. You can also specify parameters in all the JPQL statements.

JPQL is a typed language and all the expressions have a type. The abstract schema type defined in the expression configuration and the identification variable, the type for evaluating the persistence fields and relationships, and the literal type configure the expression types. For details on the syntax, see *Appendix D BNF for JPQL*.

Important note

In Cosminexus JPA Provider, an exception might occur if you use JPQL that does not conform to the BNF syntax. Even if an exception does not occur, the operation might not function properly. Also, even if you use JPQL that conforms to the BNF syntax, the operations might not function properly if the relevant functionality is not supported in the database used.

(1) Abstract schema type

With JPQL, a query is issued for an entity. Therefore, the abstract schema of the target entity must be defined with the query. The abstract schema type of the entity is defined based on the entity class and O/R mapping information provided by the annotations or the O/R mapping file.

The abstract schema type indicates an entity class. The fields and the O/R mapping information of the annotation configure the entity class.

The abstract schema type of the entity has the following fields:

- **State field**
A state field is a field or property in which a relationship-based relation does not exist in the persistence field or persistence property of the entity class.
- **Relation field**
A relation field is a persistence field or persistence property associated by relationship with the entity class. If the relationship is `OneToMany` or `ManyToMany`, the field is collection.

(2) Abstract schema name

With JPQL, you must specify the abstract schema type in order to indicate an entity. The name used for indicating the abstract schema type is called the *abstract schema name*. You define the abstract schema name in the `name` attribute of `@Entity` (or in the `name` attribute of the *entity* tag in the O/R mapping file). If the `name` attribute is not specified, the name becomes the class name of the entity class (without package name).

The abstract schema name is specific to each persistence unit.

(3) Query domain

The query domain can reference the abstract schema type of all the entities defined in the persistence unit. The abstract schema type of the other related entities can be referenced using the relation field defined in the abstract schema type.

6.17.2 SELECT statement

The `SELECT` statement contains the following clauses:

- **SELECT clause**

Specifies the value based on the type or set function of the object to be searched. Make sure that you specify the `SELECT` clause.

- **FROM clause**
Specifies the range for which a search is applied. Make sure that you specify the `FROM` clause.
- **WHERE clause**
Used for narrowing down the search results. You can omit the `WHERE` clause.
- **GROUP BY clause**
Used for grouping the search results. You can omit the `GROUP BY` clause.
- **HAVING clause**
Used for filtering the grouped search results. You can omit the `HAVING` clause.
- **ORDER BY clause**
Used for the ordered classification of the search results. You can omit the `ORDER BY` clause.

If the `SELECT` clause and `FROM` clause are not specified in the `SELECT` statement, an exception occurs.

6.17.3 SELECT clause

The `SELECT` clause expresses the query results. One or more values are returned from the `SELECT` clause of the query. You specify the following elements, demarcated by commas, in the `SELECT` clause. Note that the cluster demarcated by commas is called a *select expression*.

- Identifier of the abstract schema or persistence field that is assigned an identifier
- Path expression
- Set function
- Constructor expression

Specify the `DISTINCT` keyword when you want to exclude the duplicated values from the query result.

An example of coding the `SELECT` clause is as follows:

```
SELECT e.employeeName, e.monthlySalary
FROM Employee AS e
WHERE e.monthlySalary < 150000
```

(1) Constructor expression

The constructor expression is used in the select expression of the `SELECT` clause that returns one or more Java instances. The generated name specifies the fully qualified name.

The constructor expression can be obtained as a combination of some or all entity columns and the other related entity columns. Note that the class that stores this result need not be an entity.

Specify the syntax of the constructor expression by assigning the `NEW` operator in the select expression. If the class that stores the result is an entity, the state of the entity class instance becomes `new`. An example of coding the constructor expression is as follows:

```
SELECT NEW com.hitachi.jp.a.test.entity.EmployeeTmp
(e.employeeId, e.employeeName, d.departmentName)
FROM Department AS d, d.employees AS e
WHERE e.employeeId = 12003
```

(2) Set function

You can use the set function with the `SELECT` clause. The following table lists and describes the available set functions.

Table 6-19: Set functions available in the `SELECT` clause of JPQL

Set function	Argument	Result type	Result when the applied value does not exist
AVG	Numeric type state field #	Double type	null
MAX	Field type that can specify the order (numeric type, string type, character type, or date type) #	Type of the applied field	null
MIN	Field type that can specify the order (numeric type, string type, character type, or date type) #	Type of the applied field	null
SUM	Numeric type state field #	<ul style="list-style-type: none"> • For the integer type: Long type • For the floating point type: Double type • For the BigInteger type: BigInteger type • For the BigDecimal type: BigDecimal type 	null
COUNT	Identification variable (when you specify the path expression in the argument, specify the state field or relation field)	Long type	0

Note: Regardless of whether `DISTINCT` is specified, the null value is removed before the set function is applied.

If the path expression is specified in the argument, you cannot specify the relation field.

For details on the syntax of the set function expression, see *Appendix D BNF for JPQL*.

(3) Execution results of the `SELECT` clause

The type of the query results defined in the `SELECT` clause of the query is one of the following. If multiple types are present, the types are serialized.

- Entity abstract schema type
- Field type
- Set function results
- Constructor expression results

The result type of the `SELECT` clause is defined according to the result type of the select expression included in the `SELECT` clause. If multiple select expressions are used in the `SELECT` clause, the query result is `Object []` type. The elements of this result match the order specified in the `SELECT` clause and with the result type of each select expression.

6.17.4 FROM clause

This subsection describes the `FROM` clause.

(1) Range variable declaration and identification variables

A range variable declaration is a declaration that codes the logical name of the entity class in the `FROM` clause and then specifies `AS` and the identifier (`AS` can be omitted). The identifier of this range variable declaration is called the *identification variable*. An example of range variable declaration and identification variable is as follows:

```
SELECT ... (omitted) ...
FROM Department AS dep
```

WHERE ... (omitted) ...

The part 'Department AS dep' is the range variable declaration. Also, 'dep' is the identification variable. The syntax of the range variable declaration is as follows:

```
range_variable_declaration ::=
abstract_schema_name [AS] identification_variable
```

The syntax of the identification variable in the range variable declaration is the same as the SQL syntax. A description of the syntax is as follows:

- The use of the keyword AS is optional.
- You cannot omit the identification variable. However, you can omit the AS specified between the abstract schema and the identification variable. Specify the identification variable in the FROM clause.
- A reserved identifier cannot be used. If used, an exception occurs.
- The same name as another entity in the same persistence unit cannot be used. With Cosminexus JPA Provider, the operations might not function properly if an entity with the same name is used.
- The identification variable is not case sensitive.
- You cannot specify the same name as the abstract schema name. With Cosminexus JPA Provider, the operations might not function properly if the abstract schema with the same name is specified.
- The syntax must begin with a Java identifier character and all the other characters must be partial characters of the Java identifier. If other characters are specified, an exception occurs. For the first character, use a character by which the return value of the `Character.isJavaIdentifierStart` method becomes `true` (including underscore (`_`) and dollar mark (`$`) characters). For the characters other than the first character, use a character by which the return value of the `Character.isJavaIdentifierPart` method becomes `true` (however, question mark (`?`) is a reserved word in JPQL and cannot be used).
- The reserved words in JPQL are as follows:
SELECT, FROM, WHERE, UPDATE, DELETE, JOIN, OUTER, INNER, LEFT, GROUP, BY, HAVING,
FETCH, DISTINCT, OBJECT, NULL, TRUE, FALSE, NOT, AND, OR, BETWEEN, LIKE, IN, AS,
UNKNOWN, EMPTY, MEMBER, OF, IS, AVG, MAX, MIN, SUM, COUNT, ORDER, BY, ASC, DESC,
MOD, UPPER, LOWER, TRIM, POSITION, CHARACTER_LENGTH, CHAR_LENGTH, BIT_LENGTH,
CURRENT_TIME, CURRENT_DATE, CURRENT_TIMESTAMP, NEW, EXISTS, ALL, ANY, and SOME
Note that UNKNOWN is not used in JPA 1.0, but is a reserved word in Cosminexus JPA Provider.

(2) Path expression

A path expression is an expression in which a period (`.`) is added after the identification variable and is used for continuing the state field or relation field. Therefore, the path expression type becomes the state field or relation field type.

You can build up a path expression further from the relation fields obtained by tracing the path expression. However, if the path expression type that forms the base is a collection relation field, you cannot build a path expression. Creating the path expression from the collection type becomes a structural error.

Note that if the relation field in the middle of the path expression is a null value, the path assumes that the value does not exist, so the query result is not affected.

You can use the path expression with the syntax using `inner join`. For details on the path expression syntax, see *Appendix D BNF for JPQL*.

Reference note

The types of relation fields are as follows:

- A collection relation field (`collection_valued_association_field`) is one in which the relation field is specified using a collection. The collection relation field is indicated by `OneToMany` or `ManyToMany` relation.
- A non-collection relation field (`single_valued_association_field`) is one in which the relation field is specified using `single-valued`. The non-collection relation field is indicated by `OneToOne` or `ManyToOne` relation.

- The embedded class field is the field name of the entity corresponding to the embedded class.

(3) Joins expression

The Joins expression is available in the `FROM` clause. The following table lists and describes the available Joins expressions.

Table 6-20: Joins expressions available in the `FROM` clause

Joins expression	Contents	Syntax name of BNF syntax #1
Inner Joins	This expression binds two entity classes and extracts only the entity objects with relation, in the fields to be related.	<code>join, join_spec</code>
Left Outer Joins	This expression binds two entity classes and extracts the entity objects with relation as well as the entity objects without relation, in the fields to be related.	<code>join, join_spec</code>
Fetch Joins	This expression binds two entity classes in the related fields. Note that a relation exists between the entities due to relationship, so specify only one entity class in the <code>Select</code> clause. #2	<code>fetch_join</code>

#1 For details on the syntax name of the BNF syntax, see *Appendix D BNF for JPQL*.

#2 When the entity is obtained using Fetch Joins, the information about the relation destination entity specified on the right is obtained at the same time as the execution of the query. As a result, you can obtain the relation destination information without dependence on the fetch strategy. An example of coding Fetch Joins is as follows:

```
SELECT emp FROM Employee AS emp JOIN FETCH emp.company
```

Notes on using the Join expression

The notes on using the Join expression are as follows:

Notes on Inner Joins

The keyword `INNER` is used optionally.

Notes on Left Outer Joins

The keyword `OUTER` is used optionally.

Notes on Fetch Joins

- With Fetch Joins, the entity information of two entities is specified in one entity. The information of the specified entity and the information of another entity related to that entity are bound.
Note that in Cosminexus JPA Provider, you specify entities with relationship because the entities are bound by one entity information. If entities without relationship are specified, an exception occurs.
- The relation referenced on the right side of `JOIN FETCH` must be a relation belonging to the entity returned as a query result. In Cosminexus JPA Provider, if the relation does not belong to the entity, an exception occurs.
- An identifier cannot be specified in the entity referenced on the right side of `JOIN FETCH`. Therefore, the entity cannot be referenced in the query.

(4) Declaring the collection members

The identification variables for declaring the collection members are declared by using the reserved identifier `IN`. You can obtain the collection value for the identification variable defined in the collection member expression, by using the path expression. An example of coding the collection member expression is as follows:

```
SELECT emp.employeeId, emp.employeeName, dep.departmentName
FROM Department AS dep, IN (dep.employees) AS emp
WHERE dep.departmentId = 3
```

For details on the syntax of the collection member expression, see *Appendix D BNF for JPQL*.

(5) Notes

This section describes the notes on the `FROM` clause.

- Effect of the identification variable
Even if the identification variable declared in the `FROM` clause is not used in the `WHERE` clause, the declared identification variable is applied to the query result.
- Notes on polymorphism
JPQL is a polymorphism. The specific entity class instances referenced explicitly by the `FROM` clause as well as the subclasses become the target. Therefore, the instances that can be obtained using a query include subclass instances that satisfy the query conditions.

6.17.5 WHERE clause

The `WHERE` clause is made up of conditional expressions used for searching the objects or variables that satisfy the expression. The `WHERE` clause specifies the result of the `select` statement and the scope of the `update` and `delete` operations.

(1) Conditional expressions that can be used in the WHERE clause

The following table describes the conditional expressions that can be used in the `WHERE` clause.

Table 6-21: Conditional expressions that can be used in the `WHERE` clause

Expression	Contents	Syntax name of BNF syntax#
<code>BETWEEN</code>	Evaluates whether the value is included in the scope specified for the field.	<code>between_expression</code>
<code>IN</code>	Evaluates that the value matches with some value specified in the field.	<code>in_expression</code>
<code>LIKE</code>	Evaluates that the value matches with the string after a wild card sign is allocated in the field.	<code>like_expression</code>
<code>IS [NOT] NULL</code>	Tests whether the value is a null value.	<code>null_comparison_expression</code>
<code>IS [NOT] EMPTY</code>	Tests whether the specified collection value is empty.	<code>empty_collection_comparison_expression</code>
<code>[NOT] MEMBER [OF]</code>	Tests whether the specified collection value is a collection member. When an empty collection is expressed, the value of the <code>MEMBER OF</code> expression is <code>FALSE</code> and the value of the <code>NOT MEMBER OF</code> expression is <code>TRUE</code> .	<code>collection_member_expression</code>
<code>EXISTS</code>	Determines the sub-query result. If one or more values exist, <code>true</code> is returned and in other cases, <code>false</code> is returned.	<code>exists_expression</code>
<code>ALL</code>	Compares a value with all the values returned by the sub-query.	<code>all_or_any_expression</code>
<code>ANY (SOME)</code>	Compares a value with some value returned by the sub-query.	<code>all_or_any_expression</code>
Sub-query	Results can be coded based on <code>select</code> .	<code>subquery</code>
Functional expression	See (2) <i>Functional expressions</i> .	See the syntax of the relevant function

For details on the syntax name of the BNF syntax, see *Appendix D BNF for JPQL*.

The notes on using the `WHERE` clause are as follows:

- **IN expression**
 - If the field to be evaluated is `null` or `unknown`, the expression value becomes `unknown`.
 - The list that defines the value set, demarcated by commas, for the `IN` expression, must have at least one or more elements. In Cosminexus JPA Provider, an exception occurs if one or more elements do not exist.

- The field to be evaluated must have a string, numeric, or enum value.
- The literal and input parameter values, and the sub-query results must have the same abstract schema type as the field to be evaluated. In Cosminexus JPA Provider, an exception occurs if the abstract schema type is not the same.
- **LIKE expression**
 - If the value of `string_expression` or `pattern_value` indicated in the BNF syntax is `null` or `unknown`, the value of the LIKE expression becomes `unknown`.
 - If `escape_character` indicated in the BNF syntax is specified and if the value is `null`, the value of the LIKE expression becomes `unknown`.
 - In the specified pattern value, an underscore (`_`) corresponds to one character and a percent (`%`) character corresponds to continuous characters (including continuous null characters). Note that the other characters indicate the search string.
 - The optional escape character is a string literal or character. Use the escape character to interpret the underscore and percent characters of the pattern string as the standard characters.
- **IS [NOT] NULL expression**
If the specified collection value is `null` or `unknown`, the value of the IS [NOT] EMPTY expression becomes `unknown`.
- **[NOT] MEMBER [OF] expression**
If the specified value is `null` or `unknown`, the return value becomes `unknown`.
- **ALL expression or ANY (SOME) expression**
 - If the conditional operation is neither `true` nor `false`, the value becomes `unknown`.
 - The comparison operators that can be used together are `=`, `<`, `<=`, `>`, `>=`, `<>`.
 - The sub-query result type must be the same as the comparison operator type. In Cosminexus JPA Provider, an exception occurs if the types are different.
 - SOME is the synonym of ANY. #

This note is only applicable to ANY (SOME) expression.
- **Sub-query**
Used in the `WHERE` clause and `HAVING` clause. Cannot be used in the `FROM` clause.

(2) Function expression

With JPQL, the functions listed in the following table are available in the `WHERE` clause and `HAVING` clause. If the value of the function expression argument is `null` or `unknown`, the value of the function expression becomes `unknown`.

Table 6-22: Functions available with JPQL

Category	Functions	Return value	Supplement to arguments
String function	CONCAT	Concatenated string of the argument	--
	SUBSTRING	String with the start position and length specified in the argument	The second and third arguments specify the start position and length of the returned sub-string as an integer. The position of the beginning of the string is 1.
	TRIM	String with a specific character removed	If no character is to be removed, a space (blank) is assumed. The optional <code>trim</code> character is the <code>character</code> input string. If the trim method is not specified, BOTH is used.
	LOWER	De-capitalized string	--
	UPPER	Capitalized string	--
	LENGTH	Integer value of the string length	--

Category	Functions	Return value	Supplement to arguments
String function	LOCATE	Integer value of the given string and the position where the string is first found after searching from a specified position (if the string is not found, 0)	The first argument indicates the string to be searched and the second argument indicates the searched string. The optional third argument indicates the position of the character to start the search (by default, the search is performed from the beginning). The start position of the string is 1.
Arithmetical function	ABS	Absolute value with the same type as the function argument (Integer, Float, or Double)	A numerical value is passed as an argument.
	SQRT	Square root of the numerical value passed in the argument (real number)	A numerical value is passed as an argument.
	MOD	Remainder of two numerical values passed in the argument (integer)	Two integers are passed.
	SIZE	Number of collection elements (integer)	Collection is passed.
Date and time function	CURRENT_DATE	Database date	--
	CURRENT_TIME	Database time	--
	CURRENT_TIMES TAMP	Database timestamp	--

Legend:

--: Not applicable

(3) Notes

This section describes the notes on the WHERE clause.

(a) Priority of the operators

The priority of the operators is as follows:

1. Period (.)
2. Arithmetic operators
Unary operation (+, -), multiplication and division (*, /), addition and subtraction (+, -)
3. Comparison operators
=, >, >=, <, <=, <> (not equal), [NOT] BETWEEN, [NOT] LIKE, [NOT] IN, IS [NOT] NULL, IS [NOT] EMPTY, [NOT] MEMBER [OF]
4. Logical operators
NOT, AND, OR

(b) Notes on conditional expressions

- A conditional expression is made up of a conditional expression, comparison expression, and a logical operator, and the evaluation result is made up of the path expression that forms a boolean value, boolean literal, and input parameters of boolean type.
- An arithmetic expression is available in the comparison expression. An arithmetic expression is made up of other arithmetic expressions and the four arithmetic operations. The result is made up of the path expression that forms a numerical value, number literals, and the input parameters of the numerical type.
- An arithmetic operation uses numeric promotion.
- To specify the evaluation order of the expression, you can enclose the evaluation order in parentheses.
- The set function is only available in the conditional expression of the HAVING clause.

- In a conditional expression, do not use fields mapped as a serialized format or `lob`s.

(c) Notes on using literals

- Enclose a character literal in single quotes (example: 'literal'). For a string literal containing single quotes, use two single quotes.
- As in the case of the `JavaString` literal, use Unicode character encoding for the query string literal.
- The use of Java escape expression is not supported in the query string literal.
- For the enum literal, you can use the literal of the Java enum literal syntax. The enum literal requires an enum class name.
- The boolean literal is `TRUE` and `FALSE`. This literal is not case sensitive.

(d) Notes on identification variables

- The identification variable is an identifier declared in the `FROM` clause of the query, so the identification variable cannot be declared in another clause. If the variable is declared in a clause other than the `FROM` clause, an exception occurs. For all the identification variables used in the `WHERE` or `HAVING` clause of the `SELECT` statement or `DELETE` statement, use the identification variables defined in the `FROM` clause.
- The range of the identification variables is determined by the `WHERE` clause and `HAVING` clause. An identification variable can specify the collection members and instances of the abstract schema type of the entity, but cannot specify the collection itself. If a collection is specified, an exception occurs.

(e) Notes on path expressions

In the path expression of the collection in the `WHERE` or `HAVING` clause, do not use values other than `IS [NOT] EMPTY` expression and `[NOT] MEMBER [OF]` expression, or arguments for the `SIZE` operation as a part of the conditional expression.

(f) Notes on input parameters

- The input parameters are available in the `WHERE` clause or `HAVING` clause of the query.
- Do not mix the location parameters and named parameters in one query. If the parameters are mixed, the operations might not function properly.
- If the value of the input parameters is `null`, the value returned by the comparison operation or arithmetic operation containing the input parameters becomes `unknown`.

For details on how to use the location parameters and the named parameters, see *6.16.1(2) How to specify the parameters*.

6.17.6 GROUP BY clause and HAVING clause

With the `GROUP BY` clause, the query results are compiled for each group. With the `HAVING` clause, you can specify conditions for narrowing down the query results further. After specifying the group, specify the `HAVING` clause conditions.

If a query contains both the `WHERE` clause and `GROUP BY` clause, the `WHERE` clause is executed first, the format is adjusted with the `GROUP BY` clause, and then filtering is performed according to the `HAVING` clause.

The items other than the set function appearing in the `SELECT` clause must also be specified in the `GROUP BY` clause. With grouping, the null value is also included and is handled as a condition. The notes related to the `GROUP BY` clause and `HAVING` clause are as follows:

- Entity-based grouping can be performed, but serialized fields and lob fields cannot be included. If serialized fields and lob fields are specified with Cosminexus JPA Provider, an exception occurs.
- In the `HAVING` clause, search conditions are specified for the group items, so the set function applicable to the group items must be specified. With Cosminexus JPA Provider, if the search conditions are not specified, an exception occurs.

- Do not use the `HAVING` clause when the `GROUP BY` clause does not exist. If used, an exception occurs.

An example of coding the `GROUP BY` clause and `HAVING` clause is as follows:

```
SELECT e.department.departmentId
FROM Employee AS e
GROUP BY e.department.departmentId
HAVING COUNT(e.department.departmentId) <= 2
```

For details on the syntax of the `GROUP BY` clause and `HAVING` clause, see *Appendix D BNF for JPQL*.

6.17.7 ORDER BY clause

With the `ORDER BY` clause, the objects and values are placed in order and then the query results are returned. An example of coding the `ORDER BY` clause is as follows:

```
SELECT e
FROM Employee AS e
ORDER BY e.monthlySalary DESC
```

A description of the `ORDER BY` clause is as follows:

- If one or more `order` items are specified, the priority is determined from the left to the right of the `orderby` item elements and the left-most `orderby` item has the highest priority.
- The `ORDER BY` clause is specified when `ASC` is ascending and `DESC` is descending. Note that the default value is `ASC`.
- The SQL rules are applied to the order when the `null` value exists.
- When the `ORDER BY` clause is used, the order of the query results is stored in the result of the query method.

For details on the syntax of the `ORDER BY` clause, see *Appendix D BNF for JPQL*.

Furthermore, the `ORDER BY` clause must satisfy the following conditions:

- The `order` items specified in the `ORDER BY` clause can be placed in order.
- The `order` items specified in the `ORDER BY` clause can be traced from the `select` expression of the `SELECT` clause.

If these conditions are not satisfied, an exception might occur in Cosminexus JPA Provider. However, even if an exception does not occur, the operations might not function properly.

6.17.8 Bulk UPDATE statement and Bulk DELETE statement

Update of multiple records in a batch is called *bulk update*. To perform bulk update, you use the Bulk `UPDATE` statement and Bulk `DELETE` statement. The bulk `UPDATE` and `DELETE` operations are applied to a stand-alone entity class or an entity class matching with the subclass. With the `UPDATE` statement, you define the identification variable in the `UPDATE` clause, and with the `DELETE` statement, you define the identification variable in the `FROM` clause and specify one type of entity.

Notes on using the Bulk `UPDATE` statement and Bulk `DELETE` statement:

- The `delete` operation is only applied to the entities of the specified class and the subclasses. The related entities are not cascaded.
- The update value specified in the update operation (`new_value`) must be compatible with the allocated field type. If the update value is not compatible, an exception occurs.
- The Bulk `UPDATE` statement executes the database update operation without setting the optimistic lock, so the processing related to the optimistic lock is not executed. Therefore, the value in the version column must be referenced and updated manually.

! Important note

When you execute the `Bulk UPDATE` statement and the `Bulk DELETE` statement, note that mismatch occurs between the database and the entities of the activated persistence context. The operations in the `Bulk UPDATE` statement and the `Bulk DELETE` statement must be executed when separated from the transaction or when the transaction is started.

An example of coding the `Bulk UPDATE` statement and `Bulk DELETE` statement is as follows:

```
UPDATE EMPLOYEE
SET MONTHLY_SALARY = MONTHLY_SALARY + 1000
WHERE DEPARTMENT_ID = 3

DELETE FROM EMPLOYEE e
WHERE e.EMPLOYEE_NAME IS NULL AND e.monthlySalary IS EMPTY
```

For details on the syntax of the `Bulk UPDATE` statement and `Bulk DELETE` statement, see *Appendix D BNF for JPQL*.

6.17.9 Notes on using JPQL

This subsection describes the notes on using JPQL.

(1) Notes on the null value

- **null value in the query result**
 - If the query result value corresponds to a relation field or state field with a `null` value, that `null` value is returned as the result of the query method.
 - You can use the `IS NOT NULL` syntax to remove the `null` value from the query result set.
 - In the fields defined using the numeric series primitive type of Java, a `null` value cannot be generated as a query result.
- **null value in the comparison and conditional expressions**
 - If the reference target does not exist in the database, the value is assumed to be `null`. To search a `null` value, you can only use the comparison conditions `IS NULL` and `IS NOT NULL`.
 - If the `null` value is used in a condition other than `IS NULL` and `IS NOT NULL` and if that result depends on the `null` value, the result becomes `unknown`.
 - The result of the comparison or arithmetic operation of the `null` value is always an `unknown` value.
 - The result of comparing two `null` values is an `unknown` value.
 - The result of the comparison or arithmetic operation with an `unknown` value is always an `unknown` value.
 - The boolean operator uses three-valued-logic. The following table describes the three-valued-logic expressions for `AND`, `OR`, and `NOT`.

Table 6-23: Three-valued-logic expression for `AND` (result of `A AND B`)

A	B		
	True	False	Unknown
True	True	False	Unknown
False	False	False	False
Unknown	Unknown	False	Unknown

Table 6-24: Three-valued-logic expression for OR (result of A OR B)

A	B		
	True	False	Unknown
True	True	True	True
False	True	False	Unknown
Unknown	True	Unknown	Unknown

Table 6-25: Three-valued-logic expression for NOT (result of NOT A)

A	Result of NOT A
True	False
False	True
Unknown	Unknown

(2) Notes on using JPQL in HiRDB

- You cannot specify location parameters and named parameters in the arguments of the following JPQL functions: TRIM, SQRT, ABS, LENGTH, LOWER, MOD, LOCATE, UPPER, CONCAT, SUBSTRING, IS [NOT] NULL
If location parameters and named parameters are specified in the arguments of these functions, the operations might not function properly. Note that if the location parameters and named parameters are specified, a HiRDB SQL syntax error occurs and the `PersistenceException` exception containing the `SQLException` exception might be thrown.
If you want to use the functionality of these functions, use the native query.
- You cannot specify the ? (question mark) parameter on both sides of the four arithmetic operators (+, -, *, /) and both sides of the comparison operators (=, >, >=, <, <=, <>). Also, you cannot specify the ? (question mark) parameter on one side of the comparison operators and a literal on the other side. If such a value is specified, the operations might not function properly. If such a value is specified, a HiRDB SQL syntax error occurs and the `PersistenceException` exception containing the `SQLException` exception might be thrown.
Instead of executing the four arithmetic operations and comparison operations in JPQL, execute the four arithmetic operations and comparison operations before using JPQL and then use JPQL.

The examples of coding queries that cannot be used when JPQL is used in HiRDB are as follows:

Example 1 of query that cannot be used: Specifying a location parameter in the function argument

```
Query query1 = em.createQuery(
"SELECT o FROM TestEntity o "+
    "WHERE o.name=TRIM(LEADING FROM ?1)")
    .setParameter(1, " HitachiTaro");
```

Example 2 of query that cannot be used: Specifying a location parameter in the four arithmetic operators

```
int no_A=2;
int no_B=4;
Query query2 = em.createQuery(
"SELECT o FROM TestEntity o WHERE o.id = ?1 + ?2")
    .setParameter(1, no_A)
    .setParameter(2, no_B);
```

Example 3 of query that cannot be used: Specifying a location parameter in the comparison operators

```
int cmp_no=3;
Query query3 = em.createQuery(
"SELECT o FROM TestEntity o WHERE o.id = ?1 AND ?1 < 9")
    .setParameter(1, cmp_no);
```

6.17.10 Exceptions thrown when queries are used

In Cosminexus JPA Provider, if there is a syntactic error in a query-related annotation, an exception occurs when the application is deployed. Also, if the arguments of the query-related method are invalid and if the specified string is not a valid JPQL string, the `IllegalArgumentException` exception occurs or the execution of the query fails.

If the query is not valid for the database query that uses a native query or if the defined result set is not compatible with the query result, the execution of the query fails. In Cosminexus JPA Provider, the `PersistenceException` exception is thrown during the execution of the query.

(1) Exceptions thrown in the API functions of the query-related interfaces in `EntityManager`

The exceptions thrown in the API functions of the query-related interfaces in `EntityManager` are as follows:

- If the query string of the `createQuery` method is incorrect, the `IllegalArgumentException` exception is thrown.
- If the query is not defined with the name specified in the `createNamedQuery` method, the `IllegalArgumentException` exception is thrown.

(2) Exceptions thrown in the API functions of the `Query` interface

The exceptions thrown in the API functions of the `Query` interface are as follows:

- If the JPQL `UPDATE` statement or `DELETE` statement is invoked with the `getResultList` method, the `IllegalStateException` exception is thrown.
- If the query result does not exist, the `NoResultException` exception is thrown in the `getSingleResult` method. Note that if there are two or more query results, the `NonUniqueResultException` exception is thrown. If the JPQL `UPDATE` statement or `DELETE` statement is invoked, the `IllegalStateException` exception is thrown.
- If the JPQL `SELECT` statement is invoked with the `executeUpdate` method, the `IllegalStateException` exception is thrown. At this time, if the transaction does not exist, the `TransactionRequiredException` exception is thrown.
- If the second argument of the `setHint` method is not correct for implementation, the `IllegalArgumentException` exception is thrown.
- If the position of the arguments for the `setParameter` method does not match with the location parameter of the query, or if the parameter name of the argument does not match with the parameter of the query string, the `IllegalArgumentException` exception is thrown.
- If the arguments of the `setMaxResults` or `setFirstResult` method are negative, the `IllegalArgumentException` exception is thrown.

For details on the API functions, see the *Java documentation*.

6.18 Defining persistence.xml

This section describes the definition for the cache functionality of the entity objects, which is a Cosminexus JPA Provider-specific functionality, and the notes on data source specification for defining `persistence.xml`.

(1) Defining the cache functionality of the entity objects

You define the cache functionality of the entity objects provided with Cosminexus JPA Provider in the *property* tag of `persistence.xml`. The following table describes the definition of the cache functionality of the entity objects in `persistence.xml`.

Table 6-26: Definition of the cache functionality of the entity objects in `persistence.xml`

Specified properties	Settings
<code>cosminexus.jpa.cache.size.<ENTITY></code>	Specify the cache size for caching the entity.
<code>cosminexus.jpa.cache.size.default</code>	Specify the default cache size for caching the entity.
<code>cosminexus.jpa.cache.type.<ENTITY></code>	Specify the cache type of the entity.
<code>cosminexus.jpa.cache.type.default</code>	Specify the default cache type of the entity.
<code>cosminexus.jpa.target-database</code>	Specify the name of the database to be connected to.

For details on tags, see 6.2.2 *Cosminexus JPA Provider-specific properties that can be specified in the property tag* in the *uCosminexus Application Server Definition Reference Guide*.

Reference note

The properties described here are properties unique to Cosminexus JPA Provider. In `persistence.xml`, you can specify other properties defined in the JPA specifications. However, in Cosminexus JPA Provider, you cannot use properties beginning with `javax` that are defined in the JPA specifications.

(2) Notes on data source specification

With the data source specification in `persistence.xml`, you can use the user-specified Namespace functionality, which is an Application Server functionality, to assign an optional name to the resource adapter. If you set an optional name for the resource adapter in the `persistence.xml` settings, the optional name of the resource adapter must also be defined in the HITACHI Connector Property file. For details, see 6.19 *Settings in the execution environment*.

6.19 Settings in the execution environment

To use Cosminexus JPA Provider, you must specify the J2EE server settings and the DB Connector settings.

(1) J2EE server settings

You implement the J2EE server settings with the Easy Setup definition file. Specify the settings for the log files output by Cosminexus JPA Provider in the *configuration* tag of the logical J2EE server (`j2ee-server`) in the Easy Setup definition file.

The following table describes the settings in the Easy Setup definition file for the log files output by Cosminexus JPA Provider.

Table 6-27: Settings in the Easy Setup definition file for the log files output by Cosminexus JPA Provider

Specified parameters	Settings
<code>ejbserver.logger.channels.define.JPAOperationLogFile.filenum</code>	Specify the number of operation log files.
<code>ejbserver.logger.channels.define.JPAMaintenanceLogFile.filenum</code>	Specify the number of maintenance log files.
<code>ejbserver.logger.channels.define.JPAOperationLogFile.filesize</code>	Specify the size of the operation log.
<code>ejbserver.logger.channels.define.JPAMaintenanceLogFile.filesize</code>	Specify the size of the maintenance log.
<code>cosminexus.jpa.logging.level.operation.category</code>	Specify the log level of the operation log.

For details on the Easy Setup definition file and the specified parameters, see *4.6 Easy Setup definition file* in the *uCosminexus Application Server Definition Reference Guide*.

(2) DB Connector settings

You implement the DB Connector settings with the server management commands and the HITACHI Connector Property file.

The following table describes the settings in the HITACHI Connector Property file.

Table 6-28: Settings in the HITACHI Connector Property file

Specified tags	Settings
<code><resource-external-property>-<optional-name> tag</code>	If the optional name of the resource adapter is used in the data source specification of <code>persistence.xml</code> , the optional name of the resource adapter is set up. If the optional name of the resource adapter is not used in <code>persistence.xml</code> , the settings need not be specified.
<code><resource-external-property>-<res-auth> tag</code>	If the optional name of the resource adapter is used in the data source specification of <code>persistence.xml</code> , make sure you specify <code>Container</code> as the resource authentication method.
<code><resource-external-property>-<res-sharing-scope> tag</code>	If the optional name of the resource adapter is used in the data source specification of <code>persistence.xml</code> , make sure you specify <code>Shareable</code> to specify whether the resource connections will be shared.
<code><property>-<property-name> tag</code> <code><property>-<property-value> tag</code>	Specify the information for connecting to the database. <ul style="list-style-type: none"> Specifying the user name

Specified tags	Settings
<p><property>-<property-name> tag</p> <p><property>-<property-value> tag</p>	<p>Specify <code>User</code> in the <property-name> tag and the user name to be used for the database connection in the <property-value> tag.</p> <ul style="list-style-type: none"> • Specifying the password <p>Specify <code>Password</code> in the <i>property-name</i> tag and the password to be used for the database connection in the <i>property-value</i> tag.</p>
<p><connection-definition>-<transaction-support> tag</p>	<p>Specify the transaction support level. Match the value specified here with the value specified in the <code>transaction-type</code> attribute of the <i>persistence-unit</i> tag in <code>persistence.xml</code>.</p>

For details on the HITACHI Connector Property file and the specified tags, see [4.1 HITACHI Connector Property file](#) in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

7

Cosminexus JMS Provider

This chapter describes the functionality of using Cosminexus JMS Provider to send and receive messages. Cosminexus JMS Provider is a functionality used for sending and receiving messages conforming to the JMS specifications.

7.1 Organization of this chapter

This chapter describes the Cosminexus JMS Provider functionality.

The following table describes the organization of this chapter.

Table 7-1: Organization of this chapter (Cosminexus JMS Provider functionality)

Category	Title	Reference location
Explanation	Overview of Cosminexus JMS Provider	7.2
	Allocating the CJMSP resource adapters and CJMSP Broker	7.3
	Types of messaging models	7.4
	Configuration of messages	7.5
	Selecting the received messages using Message Selector	7.6
	Mechanism for ensuring a highly-reliable message delivery	7.7
	CJMSP Broker functionality	7.8
	CJMSP resource adapter functionality	7.9
	Invoking a Message-driven Bean	7.10
Implementation	Restrictions on implementing applications	7.11
	Definitions in the DD	7.12
Settings	Flow of execution environment setup	7.13
	CJMSP Broker settings	7.14
	CJMSP resource adapter settings	7.15
	J2EE application settings	7.16
Operations	Starting and stopping the system when Cosminexus JMS Provider is used	7.17
	Checking the troubleshooting information	7.18
Notes	Notes on using Cosminexus JMS Provider	7.19

7.2 Overview of Cosminexus JMS Provider

This section provides an overview of Cosminexus JMS Provider.

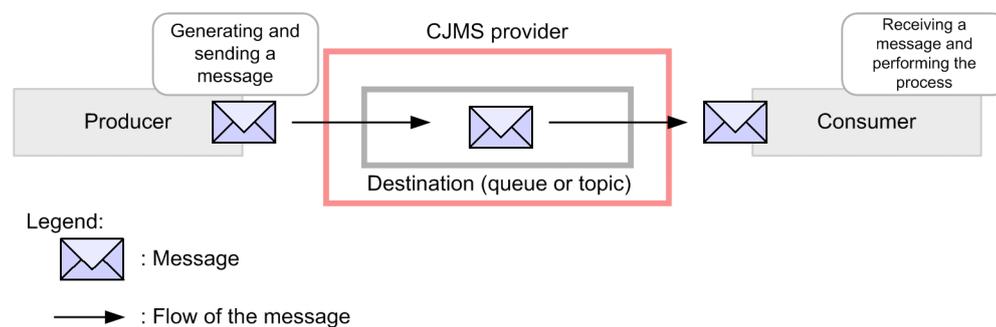
7.2.1 Cosminexus JMS Provider

Cosminexus JMS Provider is an Application Server functionality for sending and receiving messages. The functionality conforms to the JMS 1.1 specifications.

The messages that are sent and received with Cosminexus JMS Provider are created and sent using the *producer*. The sent messages are registered in the *destination* (queue or topic) managed by Cosminexus JMS Provider. Thereafter, the messages are delivered to the *consumer* and the processing is executed.

The following figure provides an overview of the sending and receiving of messages using Cosminexus JMS Provider.

Figure 7-1: Overview of the sending and receiving of messages using Cosminexus JMS Provider



The asynchronous processing at the message sending side and receiving side is implemented using the mechanism of exchanging messages through the destination.

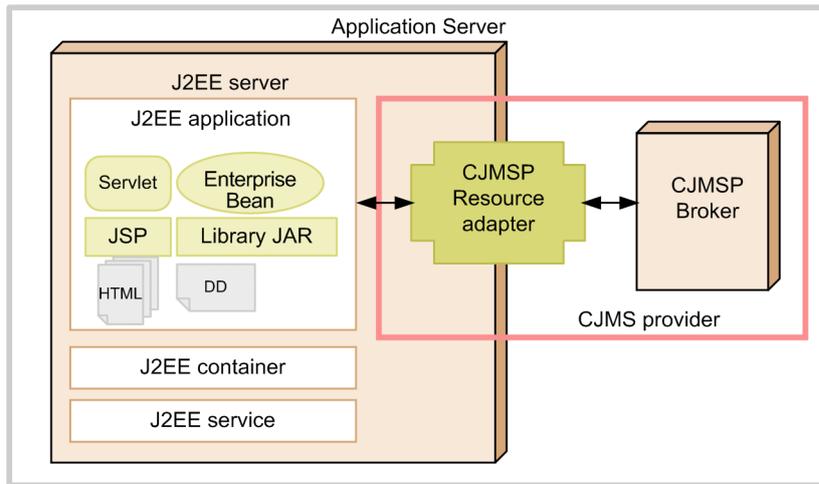
7.2.2 Location of Cosminexus JMS Provider within Application Server

Cosminexus JMS Provider is configured from the two components, named CJMSP Broker and CJMSP resource adapter. *CJMSP Broker* is a component used to manage the message destination. CJMSP Broker operates as a process separate from the J2EE server. The *CJMSP resource adapter* is used to connect CJMSP Broker from the J2EE server.

These components can only be used with Cosminexus Application Server.

The following figure shows the location of Cosminexus JMS Provider within Application Server.

Figure 7-2: Location of Cosminexus JMS Provider within Application Server



Legend:
 ↔ : Flow of the message

The J2EE application that uses Cosminexus JMS Provider is configured with items such as the servlets, JSPs, and Enterprise Beans. The J2EE application uses the CJMSP Broker functionality through the CJMSP resource adapter.

The producers and consumers that send and receive messages by using Cosminexus JMS Provider include J2EE applications on the J2EE server. The producer, which is the message-sending side, operates the servlets, JSPs, and Enterprise Beans. The consumer, which is the receiving side, operates the Message-driven Beans. If the sending and receiving J2EE servers exist on separate computers, the CJMSP resource adapter is required on both the computers.

Note that the J2EE application that uses the JMS functionality is called the *JMS application*.

7.2.3 Overview of the Cosminexus JMS Provider functionality

Cosminexus JMS Provider provides the functionality for sending and receiving messages conforming to the JMS specifications, and the functionality such as the management functionality and monitoring functionality to send and receive messages efficiently.

The following table provides an overview of the Cosminexus JMS Provider functionality. For details on the functionality, see the description in the reference location.

Table 7-2: Cosminexus JMS Provider functionality

Functionality	Overview	Reference location
Sending and receiving messages conforming to the JMS specifications	This functionality sends and receives the JMS messages with the PTP messaging model or Pub/Sub messaging model. With the Pub/Sub messaging model, you can also use the persistence subscriber. With the receiving side, you can use Message Selector to select the messages to be received.	<i>Types of messaging models</i> 7.4 <i>Configuration of messages</i> 7.5 <i>Selecting the received messages using Message Selector</i> 7.6
Mechanism for ensuring a highly-reliable message delivery	This functionality ensures a highly-reliable message delivery by using the JTA transactions and controlling the flow rate for each message type.	7.7
CJMSP Broker functionality	This functionality manages the message destinations as well as the persistent messages. Apart from this, CJMSP Broker also provides the functionality to perform operations, such as managing the connections and routing and monitoring the performance.	7.8

Functionality	Overview	Reference location
CJMSP resource adapter functionality	This functionality connects J2EE applications on the J2EE server and CJMSP Broker.	7.9
Invoking a Message-driven Bean	This functionality executes the business using a Message-driven Bean as the message consumer.	7.10

Apart from the functionality described in the table, Cosminexus JMS Provider also provides the operation management functionality required for real operations and the functionality for troubleshooting. You can set up and operate a reliable messaging system using the above functionality.

7.3 Allocating the CJMSP resource adapters and CJMSP Broker

This section describes the allocation of the CJMSP resource adapters and CJMSP Broker. You can allocate the CJMSP resource adapters and CJMSP Broker on the same or different computers.

You can allocate multiple CJMSP resource adapters for one CJMSP Broker. However, you can deploy one CJMSP resource adapter to one J2EE server.

For details on allocating components other than the CJMSP resource adapter and CJMSP Broker, see *3.8 Determining the configuration for asynchronous communication between servers* in the *uCosminexus Application Server System Design Guide*. Note that the persistence messages managed by CJMSP Broker are stored as files on the same computer as CJMSP Broker.

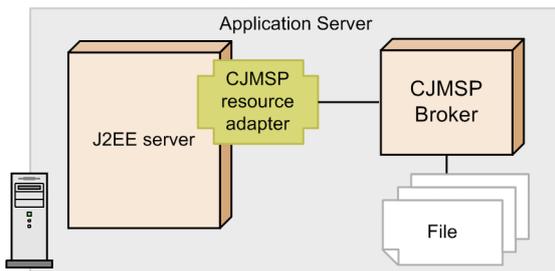
7.3.1 Configuration in which one CJMSP Broker is allocated to one CJMSP resource adapter

In this configuration, you allocate one CJMSP resource adapter and CJMSP Broker at a time. Allocate CJMSP Broker on the same computer as the J2EE server or on a different computer. The persistence messages and management information managed by CJMSP Broker are managed using files.

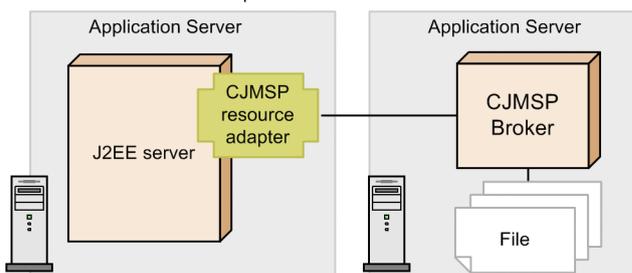
The following figure shows a configuration in which one CJMSP Broker is allocated to one CJMSP resource adapter.

Figure 7-3: Configuration in which one CJMSP Broker is allocated to one CJMSP resource adapter

- When CJMSP resource adapter and CJMSP Broker are set on the same machine



- When CJMSP resource adapter and CJMSP Broker are set on different machines



7.3.2 Configuration in which one CJMSP Broker is allocated to multiple CJMSP resource adapters

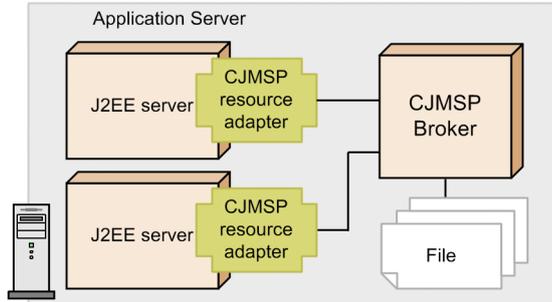
In this configuration, you allocate one CJMSP Broker to multiple CJMSP resource adapters. Allocate CJMSP Broker on the same computer as the J2EE server or on a different computer. You can deploy one CJMSP resource adapter for each J2EE server.

With this configuration, multiple CJMSP resource adapters share the destinations and resources managed by one CJMSP Broker. The persistence messages and management information managed by CJMSP Broker are managed using files.

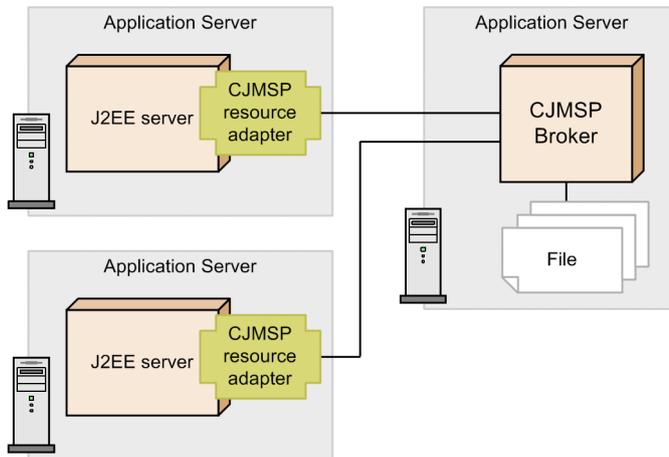
The following figure shows a configuration in which one CJMSP Broker is allocated to multiple CJMSP resource adapters.

Figure 7-4: Configuration in which one CJMSP Broker is allocated to multiple CJMSP resource adapters

- When CJMSP resource adapter and CJMSP Broker are set on the same machine



- When CJMSP resource adapter and CJMSP Broker are set on different machines



7.4 Types of messaging models

The method of sending and receiving messages differs according to the messaging model.

Cosminexus JMS Provider supports the following two messaging models:

- PTP messaging model
- Pub/Sub messaging model

This section provides an overview of sending and receiving messages with each messaging model.

Note that these messaging models conform to the JMS specifications.

7.4.1 PTP messaging model

The *PTP messaging model* sends and receives messages using the Point-to-Point method.

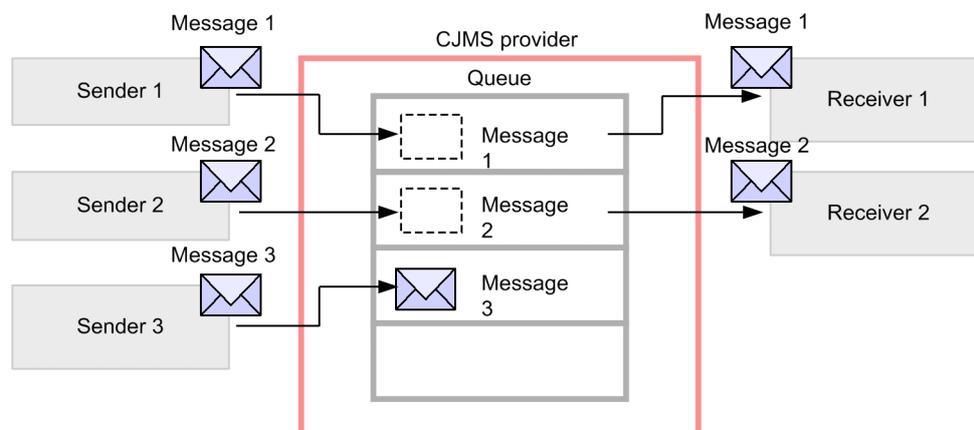
(1) Sending and receiving messages using the PTP messaging model

With the PTP messaging model, the sending client (producer) that creates and sends messages is called the *sender*. Also, the client that receives the messages (consumer) is called the *receiver*.

The messages sent from the sender are registered in a destination called the *queue*. When the receiver receives a message stored in the queue, the message is deleted from the queue.

The following figure shows the flow of messages in the PTP messaging model.

Figure 7-5: Flow of messages in the PTP messaging model



Legend:

→ : Flow of the message

The messages are sent from the sender and registered in a queue. The messages registered in the queue are delivered to one of the receivers running at that time (message1 and message2). The messages delivered to the receiver are deleted from the queue. If the delivery destination receiver does not exist, the messages accumulate in the queue (message3).

Note that you can use the queue browser provided in the JMS specifications to check the status of the queue in which the messages are registered. For details on the queue browser, see the JMS specifications.

(2) Features of the PTP messaging model

The features of the PTP messaging model are as follows:

Features from the sending of a message to the execution of processing

This point describes the features from the sending of a message to the execution of processing with the PTP messaging model.

- One or multiple senders can send messages to the queue.
- The receiver receives and processes the messages regardless of the sender that sent the message.
- One message is processed by one receiver only.
- The senders and receivers have no dependency on the process execution timing. Even if the receiver is not running when the sender sends the message, the receiver can receive the message when the receiver starts next.
- When multiple receivers receive messages from the same queue, each receiver executes the processing when the order of the messages is not important.
- The messages are registered in the queue in the order in which the sender sent the messages.
- The processing order of the messages is determined based on the validity period of the messages, the priority set for the messages, and Message Selector being used by the receiver, and the receivers are invoked according to this order.
- If no receiver is running, the sent messages are stored in the queue.

Features of the system

- You can dynamically add or delete the senders and receivers. Due to this, you can extend or scale-down the system according to the usage.
- Multiple senders can share the connections used for establishing a connection with Cosminexus JMS Provider. Also, multiple receivers can share the connections managed by Cosminexus JMS Provider. For example, sender1 and sender2 can share a connection, and receiver1 and receiver2 can share a connection. For details on the connections, see 7.8.1 *Connection services*.

7.4.2 Pub/Sub messaging model

The *Pub/Sub messaging model* sends and receives messages using the Publish-Subscribe method.

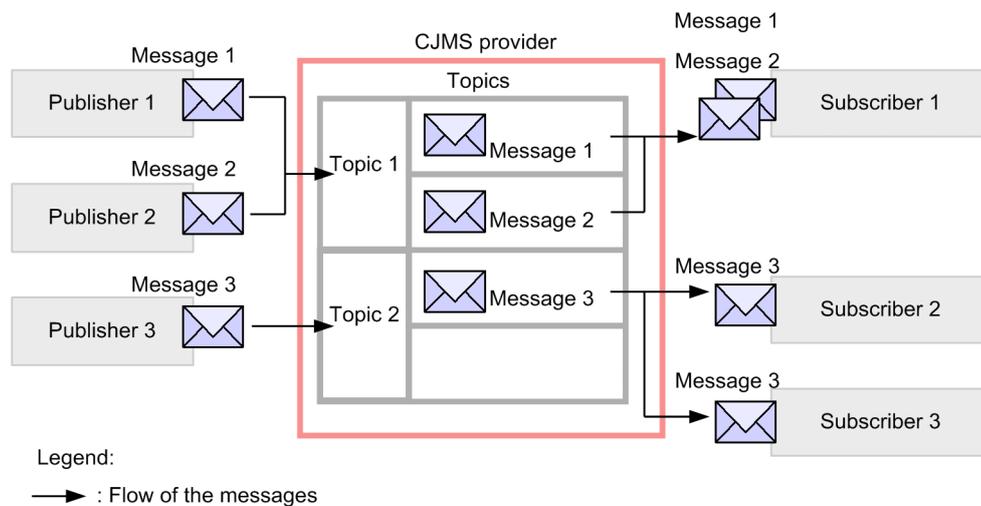
(1) Sending and receiving messages using the Pub/Sub messaging model

With the Pub/Sub messaging model, the sending client (producer) that creates and sends the messages is called the *publisher*. Also, the client (consumer) that receives the messages is called the *subscriber*.

The messages sent from the publisher are registered in a destination called the *topic*. The messages registered in the topic are delivered to one or multiple subscribers that applied to that topic for a delivery.

The following figure shows the flow of messages in the Pub/Sub messaging model.

Figure 7-6: Flow of messages in the Pub/Sub messaging model



The messages are sent from the publisher and registered in the topic. In the figure, subscriber1 is registered in topic1, and subscriber2 and subscriber3 are registered in topic2. At this time, message1 and message2 registered in topic1 are delivered to subscriber1, and message3 registered in topic2 is delivered to subscriber2 and subscriber3.

(2) Features of the Pub/Sub messaging model

The features of the Pub/Sub messaging model are as follows:

Features from the sending of a message to the execution of processing

- One or multiple publishers can register the messages into a topic.
- One or multiple subscribers can fetch and process the messages from the topic.
- The subscriber can receive and process all the messages registered in the topic to which a delivery application is made. However, the subscriber cannot receive messages not applicable to the standards set with Message Selector, or the messages with validity periods that have expired before the message was received.
- The messages are registered in the topic in the order in which the publisher sent the messages. However, the order processed with the subscriber is determined based on the validity period and the priority order of each message, or the Message Selector contents specified in the subscriber.
- The publishers and subscribers have a dependency on the process execution timing. A message registered in the topic is only delivered to the subscriber started before the message was registered.
- If `NoLocal` is specified in the subscriber attribute, the receiving of messages sent using the same connection as the connection used by the subscriber, can be controlled. The default value of this attribute is `false`.

Features of the system

- You can dynamically add or delete the publishers and subscribers. Due to this, you can extend or scale-down the system according to the usage.
- Multiple producers can share the connections used for establishing a connection with Cosminexus JMS Provider. Note that multiple producers can send messages to the same topic regardless of whether the connection is shared.
- Also, multiple subscribers can share the connections used for establishing a connection with Cosminexus JMS Provider. Note that regardless of whether the connection is shared, multiple subscribers can connect to the same topic.

(3) Using a persistence subscriber

Only the subscriber that was running when the message was registered can receive the message registered in the topic. A normal subscriber cannot receive messages that were registered during the period when the subscriber was stopped.

On the other hand, by making the subscriber persistent, even the messages registered during the period when the subscriber was stopped, can be received. A subscriber that is made persistent is called the *persistence subscriber*.

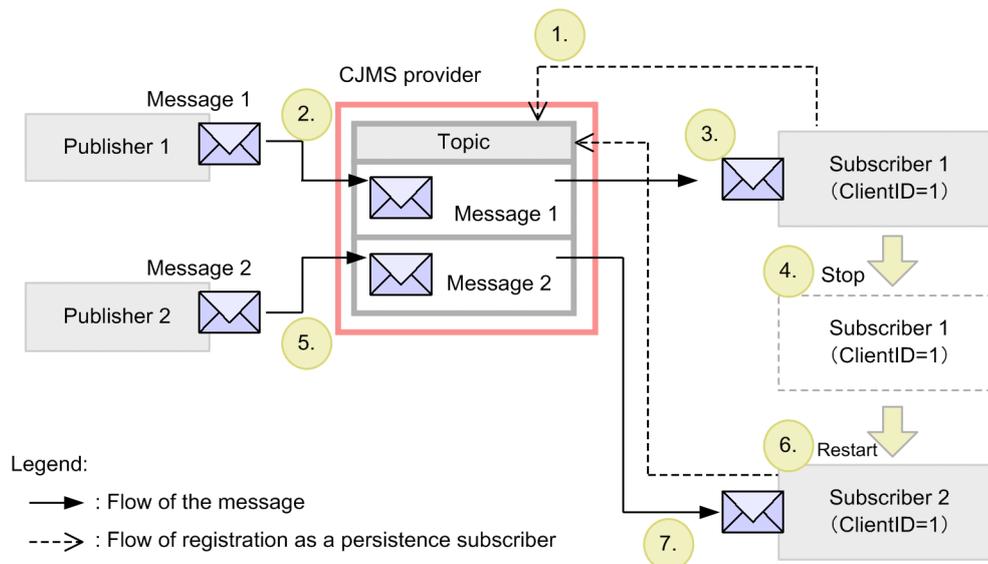
The messages in the topic where the persistence subscriber is registered are not deleted until one of the following statuses is reached:

- When the messages are delivered to the persistence subscriber
- When the message validity period is exceeded

When you use a persistence subscriber, register an identifier and name specific to that subscriber. The registered information is stored by CJMSP Broker. The messages registered when the persistence subscriber is stopped are delivered to the subscriber with the registered identifier and name when the subscriber is restarted.

The following figure shows the flow of sending and receiving messages when the persistence subscriber is used.

Figure 7-7: Flow of sending and receiving messages when the persistence subscriber is used



The following points describe the flow of sending and receiving messages when the persistence subscriber is used. Note that the numbers in the description correspond to the numbers in the figure.

1. Register subscriber1 in the topic as the persistence subscriber. Assume the identifier (`ClientID`) as 1.
2. Publisher1 registers message1 in the topic.
3. Subscriber1 receives message1 from the topic.
4. Stop subscriber1. In the stopped status, subscriber1 cannot receive the messages registered in the topic.
5. Publisher2 registers message2 in the topic. Because subscriber1 is stopped, message2 cannot be received. However, subscriber1 is registered as the persistence subscriber, so this message is stored in the topic until the registered persistence subscriber receives the message.
6. Register subscriber2 in the topic as the persistence subscriber. At this time, the identifier (`ClientID`) is assumed as 1, and the subscriber name is set to the same value as subscriber1. From the identifier and name, CJMS Provider storing the persistence subscriber information determines that subscriber2 is the restarted registered persistence subscriber.
7. Message2 stored in the topic is delivered to subscriber2.

To cancel the specified persistence subscriber, use one of the following methods:

- Execute the `cjmsicmd destroy dur` command
- Use the `unsubscribe` method

For details on the `cjmsicmd destroy dur` command, see *cjmsicmd destroy dur (Destroying the persistence subscribers)* in the *uCosminexus Application Server Command Reference Guide*.

If you use the `unsubscribe` method, you can cancel the status of the topic stored for the subscriber. However, do not use this method at the following times:

- When the subscriber started for the topic, for which you want to cancel the specified persistence subscriber, exists
- If the message delivered to the topic is an uncompleted transaction
- If the delivery in that session is not authorized

Note that when the persistence subscriber is created by a session with acknowledgement mode `CLIENT_ACKNOWLEDGE`, the message remains in the destination if the `unsubscribe` method is executed when the message receipt has not been confirmed. To avoid this situation, when you want to execute the `unsubscribe`

method, first execute the `cjmsicmd purge dur` command to delete all the messages associated with the persistence subscriber.

! Important note

You cannot register a persistence subscriber for a topic created as a temporary destination. If you make such an attempt, an exception is thrown.

7.5 Configuration of messages

The messages handled with Cosminexus JMS Provider are configured by the following elements:

- Message header
- Message properties
- Message body

These elements are provided as JMS specifications. For details, see the JMS specifications.

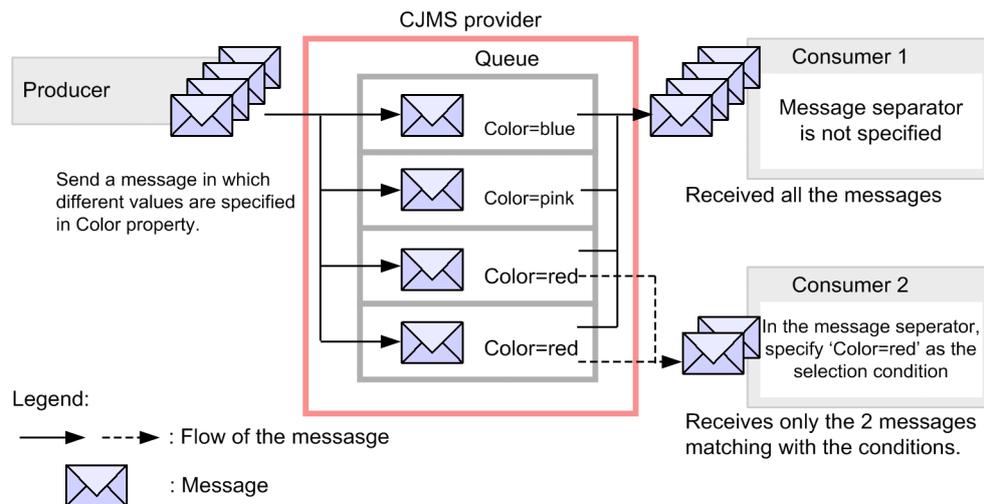
7.6 Selecting the received messages using Message Selector

The consumer can select the messages it wants to receive by using *Message Selector*. The consumers that use Message Selector can receive only the necessary messages from the messages registered in the destination.

With Message Selector, you use the message property values to specify the messages to be selected. The consumer compares the selection conditions specified in Message Selector and the values of the message properties, and receives only those messages that match the conditions.

The following figure provides an overview of selecting the received messages using Message Selector.

Figure 7-8: Overview of selecting the received messages using Message Selector



In this figure, the select condition 'Color=red' is specified for consumer2. Message Selector is not specified for consumer1.

When the producer sends four messages with different `Color` properties, consumer1 that does not use Message Selector receives all the messages. Consumer2 only receives two messages that match the selection conditions.

Note that you specify the selection conditions of Message Selector using the Message-driven Bean attributes operated in the consumer, or the JMS APIs. For details on the attributes, see *7.12 Definitions in the DD*. Also, for details on the syntax and APIs of the selection conditions, see the JMS specifications.

Reference note

You can also use Message Selector with a queue browser.

7.7 Mechanism for ensuring a highly-reliable message delivery

This section describes the mechanism for ensuring a highly-reliable message delivery using Cosminexus JMS Provider.

7.7.1 Types of problems occurring during message delivery and how to ensure reliability

A message is delivered in the following two stages:

1. Delivery from the producer to the destination managed by CJMSP Broker
2. Delivery from the destination managed by CJMSP Broker to the consumer

In the meantime, if a problem occurs at the following times, the message might be lost:

- During message transmission
 - During message transmission from the producer to the destination managed by CJMSP Broker
 - During message transmission from the destination managed by CJMSP Broker to the consumer
- When an error occurs in CJMSP Broker

The methods of ensuring reliability in these cases are as follows:

- **When a problem occurs during message transmission**
To ensure message reliability during message transmission, you can use the mechanism of message delivery acknowledgement (Acknowledge Mode), provided in the JMS specifications.
Also, if a problem occurs from the sending to the receiving of the message based on the use of transactions, the system integrity can be protected.
- **When an error occurs in CJMSP Broker**
To store the messages registered in the destination when an error occurs in CJMSP Broker, you must perpetuate the messages in advance. With Cosminexus JMS Provider, you can store the messages in a file and perpetuate the messages.

From these methods, this section describes how to ensure reliability by using transactions in *7.7.2 Using transactions*.

Also, during message delivery, a memory resource problem might occur depending on the amount of sent and received messages. The method of controlling the message flow rate in order to operate the memory resources appropriately is described in *7.7.3 Controlling the message flow rate*.

For details on the message delivery acknowledgement, see the JMS specifications. The CJMSP Broker-based file persistence is described in *7.8.4 Management information and message persistence services*.

7.7.2 Using transactions

If you use transactions, you can compile the operations from the creation of a message to the execution of processing as a series of processing. All the operations included in a transaction finish when the client-side application (application that sends the message) commits the transaction.

You can use the JTA transactions with Cosminexus JMS Provider.

A JTA transaction is managed by the transaction management API (JTA) of Application Server. If the operations in the transaction fail, Application Server processes the exception, and the processing in the transaction is retried or rolled back. You can use the BMT or CMT to manage the local transactions. For details on the BMT or CMT-based transaction management, see *2.7 Transaction management in Enterprise Beans* in the *uCosminexus Application Server EJB Container Functionality Guide*.

! Important note

When you use the JTA transactions with Cosminexus JMS Provider, the container manages the transactions of the JMS session instead of the Beans. Therefore, do not specify `true` in the value of the argument `transacted` in the following methods:

- `createSession(boolean transacted, int acknowledgeMode)` method
- `createQueueSession(boolean transacted, int acknowledgeMode)` method
- `createTopicSession(boolean transacted, int acknowledgeMode)` method

If you specify `true`, the value is ignored and a warning message is displayed.

Note that when you use the EJB applications, do not use the JMS authorization method in a transaction. The container manages the message authorization within the transaction contexts that do not specify the authorization method.

Also, if a large amount of messages are sent in one transaction, the size of the file that manages the information related to message delivery confirmation increases, and the disk capacity might be compressed. To send a large amount of messages, implement a performance test, and estimate the required file size.

7.7.3 Controlling the message flow rate

This subsection describes the controlling of the message flow rate.

During the sending and receiving of messages using Cosminexus JMS Provider, apart from the messages sent from the producer to the consumer, the control messages used by Cosminexus JMS Provider are also sent and received. The sending and receiving of these messages has a mutual impact. For example, in the cases such as when the authorization of the CJMSP Broker acknowledgement is delayed for the messages sent from the producer to the consumer, the sending and receiving of control messages is also delayed, and the entire system performance deteriorates.

By controlling the flow rate of these messages from the connection point of view, you can enhance the reliability of sending and receiving messages and improve the system performance.

With Cosminexus JMS Provider, you can limit the flow rate of the messages sent to each consumer in order to control the message flow rate.

For example, when the messages are being sent and received using the PTP messaging model, if the processing speed of each consumer is low, the messages delivered to each consumer are limited. Due to this, the processing is executed with the round robin method using multiple consumers, enabling you to improve the performance of the entire system. Determine an appropriate value for the message flow rate for consumers considering the relationship with the increased overhead of distributing and delivering the messages to multiple consumers.

Set the limit value for consumers (`consumerFlowLimit`) using the `cjmsicmd create dst` command at each destination. For the command details, see *cjmsicmd create dst (Creating the physical destinations)* in the *uCosminexus Application Server Command Reference Guide*. Note that you must set the `consumerFlowLimit` value considering the relationship with the number of concurrent executions (maximum number of pool instances) of the Message-driven Beans running on the consumer. The examples of specified patterns are as follows:

When "*consumerFlowLimit*-value < *maximum-number-of-instances-in-the-Message-driven-Bean-pool*"

With the Message-driven Beans, the processing is executed with the number of threads specified in `consumerFlowLimit`.

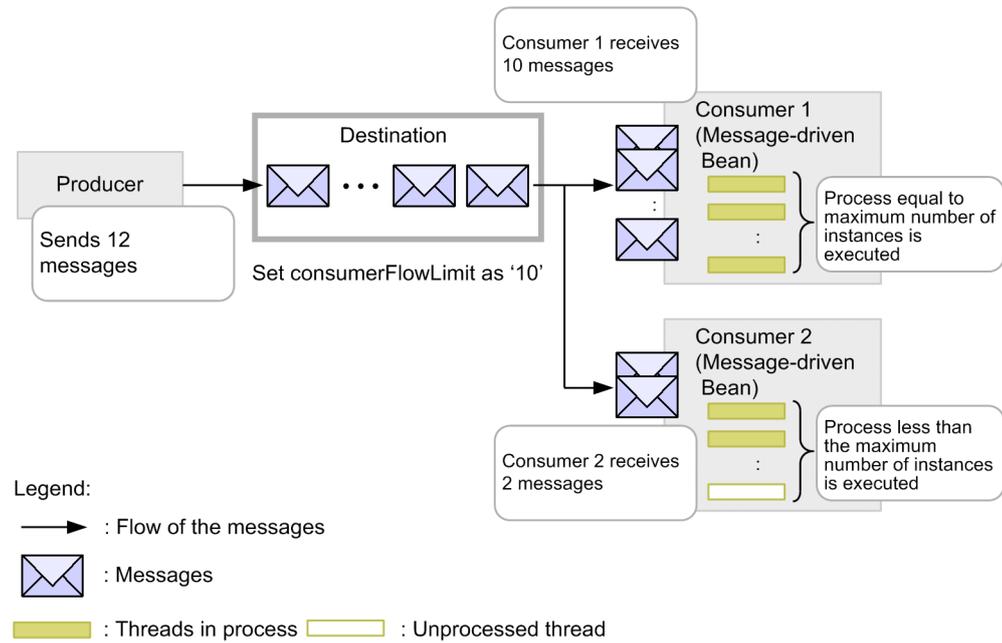
When "*consumerFlowLimit*-value = *maximum-number-of-instances-in-the-Message-driven-Bean-pool*"

If the number of messages sent from the producer is `consumerFlowLimit` or less, all the processing is executed with one consumer.

If the number of messages sent from the producer exceeds `consumerFlowLimit`, the processing is executed with multiple consumers.

The following figure provides an overview of the flow rate control based on the `consumerFlowLimit` settings.

Figure 7-9: Overview of the flow rate control based on the consumerFlowLimit settings



In this example, as the limit value for consumers, `consumerFlowLimit` is set to '10', when the producer sends 12 messages, the limit value 10 messages are sent from the destination to consumer1, and the remaining messages are sent to consumer2. With each consumer, the processing is executed using threads no more than the number specified with the maximum number of Message-driven Bean instances.

7.8 CJMSP Broker functionality

This section describes the CJMSP Broker functionality.

The main CJMSP Broker functionality is as follows:

- Connection services
- Destination management and routing services
- CJMSP Broker performance monitoring
- Management information and message persistence services

Note that apart from the above, CJMSP Broker also provides the log output functionality. For details on the CJMSP Broker log output, see *7.18 Checking the troubleshooting information*.

7.8.1 Connection services

The connection service is used to manage the CJMSP Broker and CJMSP resource adapter connection on the TCP layers (jms connection), and the CJMSP Broker and system management user connection (admin connection).

(1) Types of connection services

The connection services provided by Cosminexus JMS Provider include the following two types:

- **jms service**

This service manages the connection from the CJMSP resource adapter to CJMSP Broker. When messages are sent and received, the CJMSP resource adapter uses this service to connect to CJMSP Broker.

- **admin service**

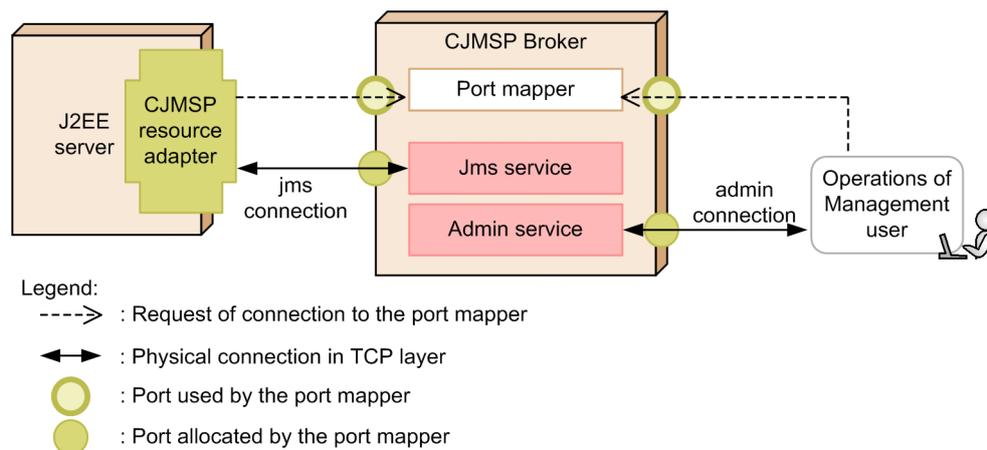
This service manages the connection from the system management user to CJMSP Broker. The management user uses this service to perform the following operations by executing commands:

- Creating the physical destinations
- Destination or CJMSP Broker query

The connection service uses a dedicated port that is statically or dynamically allocated by the port mapper.

The following figure provides an overview of the allocation of ports to the connection service by the port mapper.

Figure 7-10: Overview of the allocation of ports to the connection service



The port mapper uses the 7676 port by default. To use other ports, specify the value in the `imq.portmapper.port` property of CJMSP Broker.

The flow of port allocation is as follows:

1. A request is sent from the CJMSP resource adapter to the port mapper to establish a connection in the jms service, or from the management user to the port mapper to establish a connection in the admin service.
2. The port mapper dynamically allocates the port for the connection service request. However, if the `imq.jms.tcp.port` property or `imq.admin.tcp.port` property is specified as the CJMSP Broker property beforehand, the specified port is allocated to each connection service.

Note that if the port mapper receives multiple connection requests simultaneously, the pending requests are stored in the OS backlog and stand by.

For details on the properties used with the connection service, see *7.4 config.property (CJMSP Broker Property File)* in the *uCosminexus Application Server Definition Reference Guide*.

(2) Managing the thread pool of the connection service

A connection service is operated with multithreads, enabling the management of multiple connections concurrently. The threads used with the connection service are managed using the thread pool managed by CJMSP Broker.

Note that the connection service thread pool is managed using independent methods in order to improve the processing performance. Two threads, for message receiving and for sending, are used with one connection.

7.8.2 Destination management and routing services

This subsection describes the CJMSP Broker-based destination management and routing services.

CJMSP Broker creates and manages the queues and topics in the system memory as physical destinations.

The routing service is used to send and receive messages between the producers, consumers, and destinations reliably and smoothly while using the resources efficiently.

(1) Types of destinations

The destinations can be classified into the following two types based on the method of creation and period of existence:

- Destinations created using commands
- Temporary destinations created using APIs

The features of each destination are as follows:

Destinations created using commands

This destination is explicitly created using commands by the user who manages the system.

The command-created destinations are not deleted until explicitly deleted using commands.

Temporary destinations created using APIs

This destination is created by APIs in the J2EE application. The destination required for sending and receiving messages is created and deleted in the programs. A temporary destination is only stored in the connection in which the destination was created. Also, message persistence cannot be performed, and the destination is not recreated even if CJMSP Broker is restarted.

! Important note

A consumer that uses a temporary destination can only be created in the connection in which the temporary destination was created.

Apart from the destinations used for the normal sending and receiving of messages, there is a destination called the *dead message queue* that is automatically created by CJMSP Broker.

The *dead message queue* is created when CJMSP Broker is first started. CJMSP Broker registers the following messages (**dead messages**) in this destination:

- Messages that cannot be processed
- Messages with an expired validity period

The messages that could not be processed at other destinations and are discarded are stored in the dead message queue; therefore, the messages can be used to check for problems occurring during the sending and receiving of messages.

For details on the management of the dead message queue, see (3) *Managing the dead message queue*.

(2) Setting up and managing the destinations

With CJMSP Broker, you can execute the following processing to set up and manage the destinations:

- Creating, pausing, restarting, and deleting destinations
- Displaying a list of all the destinations
- Managing the limit values for individual messages and all the messages managed by CJMSP Broker
- Managing the maximum number of messages
- Displaying the destination status and properties
- Compressing the disk that stores the persistence messages
- Managing the dead message queue

You execute the above processing using commands. For details, see 5.3 *Details of commands for managing CJMSP Broker* in the *uCosminexus Application Server Command Reference Guide*.

(3) Managing the dead message queue

The settings specify that the dead message queue always be used with all the destinations.

You specify the processing to be performed when a message is registered in the dead message queue, in the message properties for each message.

The following table describes the content that can be set in the message properties.

Table 7-3: Content that can be set in the message properties as the processing for registration in the dead message queue

Property	Data type	Explanation
CJMS_PRESERVE_UNDELIVERED [#]	Boolean	Specifies the processing to be performed when a message cannot be delivered. <ul style="list-style-type: none"> • If <code>true</code> is specified, the message is registered in the dead message queue. • If <code>false</code> is specified, the message is not registered in the dead message queue.
CJMS_LOG_DEAD_MESSAGES [#]	Boolean	Specifies whether the information about the messages deleted from the destination and registered in the dead message queue will be output to the log. <ul style="list-style-type: none"> • If <code>true</code> is specified, the information is output to the log. • If <code>false</code> is specified, the information is not output to the log.
CJMS_TRUNCATE_MSG_BODY [#]	Boolean	Specifies whether to delete the message body of the message registered in the dead message queue. <ul style="list-style-type: none"> • If <code>true</code> is specified, the message is registered in the dead message queue after deleting the message body. • If <code>false</code> is specified, the message is registered in the dead message queue without deleting the message body.

[#] This property is not supported with CJMS Provider.

Also, CJMSP Broker sets the properties for the dead messages registered in the dead message queue. Based on the specified property information, you can check the reason for the registration of the message in the dead message queue, and the processing that formed the contributing factor.

The following table describes the properties specified for the dead messages by CJMSP Broker.

Table 7-4: Properties specified for the dead messages by CJMSP Broker

Property	Data type	Explanation
JMSXDeliveryCount	Integer	Specifies the maximum number of messages delivered to a specific consumer. This value is only specified when <code>ERROR</code> or <code>UNDELIVERABLE</code> is set in the <code>CJMS_DMQ_UNDELIVERED_REASON</code> property.
CJMS_DMQ_UNDELIVERED_TIMESTAMP	Long	Specifies the time registered in the dead message queue in milliseconds.
CJMS_DMQ_UNDELIVERED_REASON	String	Specifies the value indicating the reason for registration in the dead message queue. The following values are set up: <ul style="list-style-type: none"> • <code>OLDEST</code> Indicates that the message was not processed because it was the oldest. • <code>LOW_PRIORITY</code> Indicates that the message was not processed because the priority order was low. • <code>EXPIRED</code> Indicates that the message was not processed because the validity period of the message had expired. • <code>UNDELIVERABLE</code> Indicates that the message was not delivered. • <code>ERROR</code> Indicates that an error occurred. The messages for which this reason is specified cannot be processed due to internal reasons; therefore, the messages must be resent from the producer. <p>Note that if multiple reasons are applicable, only one of the reasons is specified.</p>
CJMS_DMQ_PRODUCING_BROKER	String	Specifies the name and port number of CJMSP Broker that created this message. If <code>null</code> is specified, it indicates a locally operating CJMSP Broker.
CJMS_DMQ_DEAD_BROKER	String	Specifies the name and port number of CJMSP Broker that manages the dead message queue. If <code>null</code> is specified, it indicates a locally operating CJMSP Broker.
CJMS_DMQ_UNDELIVERED_EXCEPTION	String	Specifies the name of the exception occurring in the application or CJMSP Broker, for the messages that were not delivered because an exception occurred.
CJMS_DMQ_UNDELIVERED_COMMENT	String	Specifies the additional reasons, if any, due to which the messages were not sent.
CJMS_DMQ_BODY_TRUNCATED	Boolean	Specifies whether the message body was deleted during registration in the dead message queue. If <code>true</code> , the message body has been deleted. If <code>false</code> , the message body has not been deleted.

7.8.3 CJMSP Broker performance monitoring

With CJMSP Broker, you can monitor the CJMSP Broker performance information (metrics).

The CJMSP Broker performance information includes information such as the heap size, connection status, percentage of received messages, and percentage of sent messages. This information is displayed on the console and then output to the log file.

Specify how to output the CJMSP Broker performance information in the `imq.metrics.interval` property. The performance information is output at the intervals specified in this property. Note that the default value is 0 (do not output).

An example of output of the CJMSP Broker performance information is as follows:

```
KDAN24558-I Displaying broker metrics :
Connections: 10 JVM Heap: 13082624 bytes (1501448 free) Threads: 22 (14-1010)
In: 3001 msgs (631149 bytes) 4297 pkts (795219 bytes)
Out: 1253 msgs (263099 bytes) 5495 pkts (690645 bytes)
Rate In: 298 msgs/sec (62622 bytes/sec) 419 pkts/sec (77978 bytes/sec)
Rate Out: 125 msgs/sec (26271 bytes/sec) 540 pkts/sec (66573 bytes/sec)
```

The following table describes the output items.

Table 7-5: Items output in the CJMSP Broker performance information

Output items	Explanation
Connections	Number of connections for the connection status.
JVM Heap	The JavaVM heap size used and the amount of memory available.
Threads	<p>Number of threads used.</p> <p>Note that the number of threads in the example of output is 22. When the connection to CJMSP Broker is only established for a specific time, the number of threads used is automatically updated.</p> <p>The numbers within () (parentheses) are the minimum and maximum values of the thread pool.</p> <p>Minimum value (14) Indicates the minimum thread pool value (the minimum value of the jms service is 10, and the minimum value of the admin service is 4. Therefore, the minimum value becomes 14).</p> <p>Maximum value (1010) Indicates the maximum thread pool value (the maximum value of the jms service is 1000, and the maximum value of the admin service is 10. Therefore, the maximum value becomes 1010).</p>
In	Number of messages received by CJMSP Broker.
Out	Number of messages sent from CJMSP Broker.
Rate In	Percentage of messages received by CJMSP Broker. This item also displays the number of messages in one second and the message packets per second.
Rate Out	Percentage of messages sent from CJMSP Broker. This item also displays the number of messages in one second and the message packets per second.

7.8.4 Management information and message persistence services

If an error occurs in CJMSP Broker, the information that indicates the status of the message sending and receiving operations before the error occurred is required for recovery. With CJMSP Broker, the status information required for recovery is stored and managed in files. If an error occurs, you can recover the status of CJMSP Broker on the basis of the stored information and restart the operations, thereby ensuring a reliable message delivery.

CJMSP Broker stores and manages the following information in files:

- Destination information

- Persistence subscriber information
- Message information
- Transaction information
- Information about the acknowledgement for the message delivery

If you restart CJMSP Broker after an error, based on the stored information, the destination and persistence subscriber are re-created, the persistence messages are recovered, the transaction is rolled back, and the route is re-created for the un-delivered messages. Thereafter, the delivery of messages is restarted.

(1) Mechanism of management information and message persistence

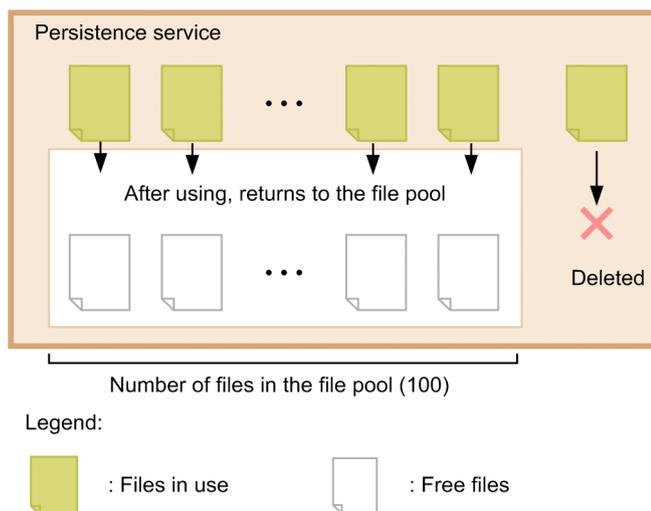
CJMSP Broker creates and manages files for each type of information to be perpetuated. The storage files are automatically created when CJMSP Broker is created. The storage destination files differ for each information type.

The messages with sizes no more than the threshold value (1 MB) are stored in one variable length file per message. Also, the messages with sizes exceeding the threshold value are managed with multiple files in the file pool. The size of each file in the file pool is 1 MB.

(2) Using the file pool

The following figure provides an overview of the file pool managed by CJMSP Broker.

Figure 7-11: Overview of the file pool managed by CJMSP Broker



The features of the file pool are as follows:

- The files that are no longer required are returned to the file pool as **free files**. A file pool is managed for each destination.
- The maximum number of files managed in the file pool is 100. When the number of files exceeds the maximum value, the files that are no longer required are deleted without being returned to the pool.
- The files in the file pool remain even after you stop and restart CJMSP Broker.

(3) File write timing

You write data to a file using the OS functionality. At this time, you can select whether to write synchronously or asynchronously with the `imq.persist.file.sync.enabled` property. Note that asynchronous writing is implemented by default.

The following table describes the advantages and disadvantages of synchronous writing and asynchronous writing.

Table 7-6: Advantages and disadvantages of synchronous writing and asynchronous writing in files

Write timing	Advantages	Disadvantages
Asynchronous	The processing performance is not affected by the write processing.	If an error occurs, some of the data might be lost.
Synchronous	The data until the point when error occurs is stored, so you can restart the processing of CJMSP Broker from the status just before the error occurred.	The processing performance is affected by the write processing.

(4) File storage destination

The files storing the persistence data managed by CJMSP Broker are stored in the following directory by default. Note that you can change the storage destination of the `var` directory, with the `-varhome` option of the `cjmsbroker` command.

In Windows

```
Cosminexus-installation-directory\CC\cjmsp\var\instances\CJMSP-Broker-name
\fs370
```

In UNIX

```
/opt/Cosminexus/CC/cjmsp/var/instances/CJMSP-Broker-name/fs370
```

The following table describes the configuration beneath the `fs370` directory.

Table 7-7: Configuration beneath the `fs370` directory

Directory or file name	Explanation
message directory	A sub-directory is created for each destination beneath this directory. An individual file for each message is stored in each sub-directory.
interest file	This file stores the consumer information.
destination file	This file stores the destination information.
property file	This file stores the internal information.
txn file	This file stores the transaction ID.
txnnack file	This file stores the delivery confirmation for all the transaction IDs.
configrecord file	This file stores the internal information.

Manage the contents of this directory appropriately by setting up permissions, such as the OS access permissions, so as to prevent unauthorized access.

! Important note

- Do not move the files beneath the `fs370` directory and do not change the file attributes to read only. CJMSP Broker might not operate correctly.
- You cannot use the network drives or the files on the network. To change the storage destination of the `var` directory with the `-varhome` option of the `cjmsbroker` command, make sure you select a directory on the same computer as CJMSP Broker.
- If you process a large amount of messages in one transaction, the size of the `txnnack` file might increase greatly. To send a large amount of messages, implement a performance test to estimate the required file size.

(5) File backup

You obtain the backup of the persistence data for each CJMSP Broker. Copy and store the `instances` directory that is beneath the `var` directory at another location.

To recover from an error, stop CJMSP Broker first. Overwrite the backed up files in the `instances` directory on to the `var` directory, and then restart CJMSP Broker.

7.9 CJMSP resource adapter functionality

The CJMSP resource adapter connects the J2EE server and CJMSP Broker. The CJMSP resource adapter sends and receives messages between the J2EE applications, operating on the Web container or EJB container of the J2EE server, and CJMSP Broker.

Specify the operations for the CJMSP resource adapter in the HITACHI Connector Property file. Note that the extent of the impact of the specified contents differs according to the location where the attributes are set up. The following table describes the location of specification and the extent of the impact.

Table 7-8: Specification location and contents of the HITACHI Connector Property file

HITACHI Connector Property file tags [#]	Specified contents
<resourceadapter>	Specifies attributes affecting the operations of the entire CJMSP resource adapter.
<connection-definition>	Specifies the settings for each connection interface.

[#]: Supports JavaBean implementation.

7.10 Invoking a Message-driven Bean

A Message-driven Bean is an Enterprise Bean that operates as a consumer and executes the asynchronous processing of messages. A Message-driven Bean becomes active when a message arrives at the destination, and receives and processes the message.

This section describes the operations of the consumer Message-driven Bean.

7.10.1 Features of message processing using the Message-driven Beans

A Message-driven Bean cannot be directly invoked from the producer that is a client application. The producer indirectly invokes a Message-driven Bean by sending a message to the destination.

Typically, a Message-driven Bean does not maintain the status and does not execute the processing in association with a specific client application. Therefore, multiple Message-driven Beans can concurrently parallel-process the sent messages.

! Important note

- Multiple messages are parallel-processed by the concurrent execution of multiple Message-driven Bean instances. The order of the processing executed by the Message-driven Bean class instances is not guaranteed.
- If an attempt is made to deliver a large amount of messages to the Message-driven Beans, the `RejectedExecutionException` exception might occur. When this exception occurs, some of the messages are not delivered to the Message-driven Beans. To avoid this situation, set the maximum number of thread pools to at least the number of Message-driven Bean instances, as the runtime attribute of the CJMSP resource adapter.
- The sending of the messages might be delayed due to the restriction on the number of concurrently executed instances specified for the Message-driven Beans. To prevent this problem, specify a value of the number of Message-driven Bean instances or more as the number of threads executed with the CJMSP resource adapters.

Also, the number of Message-driven Bean endpoints becomes the number of messages that can be processed in parallel. To prevent problems occurring due to the restrictions on the number of endpoints, specify a value of the number of Message-driven Bean instances or more as the number of endpoints. The expression is as follows:

```
Total-number-of-Message-driven-Bean-instances <= number-of-Work-threads-of-the-CJMSP-
resource-adapter
```

```
number-of-Message-driven-Bean-instances >= number-of-endpoint-instances
```

Note that you define the number of Message-driven Bean instances in `cosminexus.xml` or the `MessageDrivenBean` property file. Define the number of Message-driven Bean endpoints in the DD. Define the number of CJMSP resource adapter threads in the HITACHI Connector Property file.

For details on `cosminexus.xml`, see 2. *Cosminexus Application Property File (cosminexus.xml)* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*. For details on the `MessageDrivenBean` property file, see 3.6 *MessageDrivenBean property file* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*. For details on the HITACHI Connector Property file, see 4.1 *HITACHI Connector Property file* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

- If you use CJMS Provider to deliver messages to Message-driven Beans, and stop CJMSP Broker while the messages are being delivered, the messages are not delivered even if you restart CJMSP Broker. If you stop CJMSP Broker, restart the J2EE server.
-

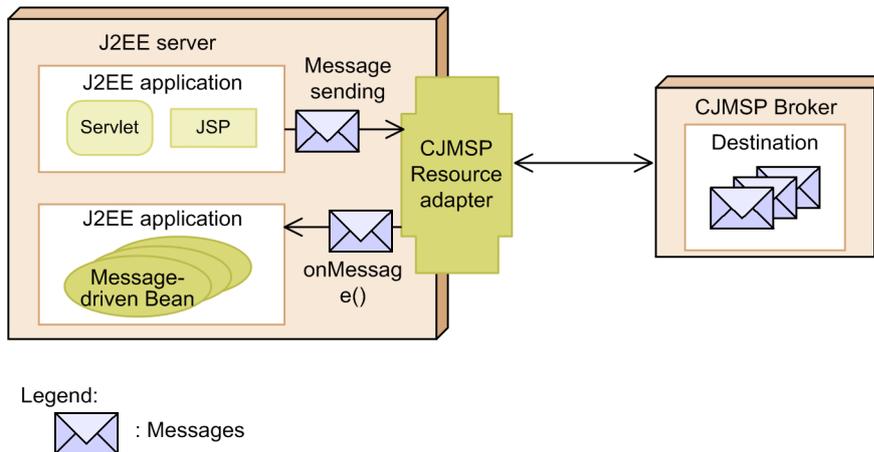
7.10.2 Procedure of invoking the Message-driven Beans

The Message-driven Beans are the message consumers for the applications that send the messages. If an application sends a message to the destination that is the message listener for a Message-driven Bean, the Message-driven Bean receives the message and executes the processing as a consumer.

When a message arrives, the `javax.jms.MessageListener#onMessage` method, which is the message listener method of the Message-driven Beans, is invoked by the J2EE container. The processing is executed according to the business logic included in the message listener method.

The following figure shows the procedure of operating the Message-driven Beans.

Figure 7-12: Procedure of operating the Message-driven Beans



Note that the destination for operating the Message-driven Beans must be set up in the JNDI Namespace of the sending source J2EE applications.

The Message-driven Beans and the destination queues or topics are associated when the J2EE applications containing the Message-driven Beans are deployed in the J2EE server.

7.10.3 Settings required in the Message-driven Beans

Set up the Message-driven Bean operations as the attributes of the Message-driven Beans. You can specify the information such as the destination information, the message acknowledgement mode, and Message Selector. Specify the attributes in the `<activation-config>` tag of the DD or `MessageDrivenBean` property file. For details, see *7.12 Definitions in the DD* and *3.6 MessageDrivenBean property file* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

This subsection describes the main settings.

(1) Setting up the destinations

Specify either queue (`javax.jms.Queue`) or topic (`javax.jms.Topic`) as the type of destination associated with the Message-driven Beans. If you specify the type as topic, you further specify whether the persistence subscriber will be used.

If you use a queue as the destination, or specify the settings to use the persistence subscriber, the destination stores the messages in the cases such as when the J2EE application that operates the Message-driven Bean is not running.

If you use a topic as the destination, and specify settings so that the persistence subscriber is not used, the sent messages are not delivered to the Message-driven Beans when the J2EE application that operates the Message-driven Bean is not running.

For details on the destination types and the persistence subscribers, see *7.4 Types of messaging models*.

(2) Setting up the message acknowledgement mode

Set up either `AUTO_ACKNOWLEDGE` or `DUPS_OK_ACKNOWLEDGE` as the message acknowledgement mode. For details on each mode, see the JMS specifications.

The container executes message acknowledgement. For a CMT-managed transaction, acknowledgement is executed as a part of the transaction-commit processing. For a BMT-managed transaction, the container executes the authorization processing separate from the transaction. Make sure you do not execute acknowledgement using APIs.

(3) Setting up Message Selector

Specify the selection criteria for the messages to be received by the Message-driven Beans as Message Selector. Due to this, you can operate the Message-driven Beans by receiving only the messages with a specific value set in the message properties.

For details on Message Selector, see *7.6 Selecting the received messages using Message Selector*.

7.10.4 Setting up the transaction context

You set up the transaction context that indicates the scope of the transactions that invoke the Message-driven Bean message listener and the timeout callback method.

If you use the CMT for transaction management, specify the `NOT_SUPPORTED` attribute.

Note that if the Message-driven Bean using the BMT manages transactions with the `javax.transaction.UserTransaction` interface, the receiving of messages is not included in the transaction processing.

7.11 Restrictions on implementing applications

This section describes the restrictions on implementing applications by using Cosminexus JMS Provider.

Cosminexus JMS Provider conforms to the JMS 1.1 specifications, but some of the interfaces and methods cannot be used. If you use an interface or method that cannot be used with an application, the `JMSEException` exception might be thrown.

The interfaces that cannot be used are as follows:

- `javax.jms.ServerSession`
- `javax.jms.ServerSessionPool`
- `javax.jms.ConnectionConsumer`
- All the `javax.jms.XA` interfaces

Furthermore, some of the methods of the interfaces listed in the following table are the methods used by the applications executed with the application client container. These methods cannot be used with Cosminexus JMS Provider.

The following table describes the methods that cannot be used with Cosminexus JMS Provider for each interface.

Table 7-9: Methods that cannot be used with Cosminexus JMS Provider

Interface	Methods that cannot be used
<code>javax.jms.Session</code>	<ul style="list-style-type: none"> • <code>setMessageListener(MessageListener listener)</code> • <code>getMessageListener()</code> • <code>run()</code>
<code>javax.jms.Connection</code>	<ul style="list-style-type: none"> • <code>setExceptionListener(ExceptionListener listener)</code> • <code>stop()</code> • <code>setClientID(String clientID)</code> • The methods that create the <code>Session</code> object with processed transactions (For example, you cannot specify <code>true</code> in the first argument <code>transacted</code> of the <code>createSession</code> method).
<code>javax.jms.QueueConnection</code>	<ul style="list-style-type: none"> • <code>createConnectionConsumer(Queue queue, String messageSelector, ServerSessionPool sessionPool, int maxMessages)</code> • The methods that create the <code>QueueSession</code> object with processed transactions (For example, you cannot specify <code>true</code> in the first argument <code>transacted</code> of the <code>createQueueSession</code> method).
<code>javax.jms.TopicConnection</code>	<ul style="list-style-type: none"> • <code>createConnectionConsumer(Topic topic, String messageSelector, ServerSessionPool sessionPool, int maxMessages)</code> • <code>createDurableConnectionConsumer(Topic topic, String subscriptionName, String messageSelector, ServerSessionPool sessionPool, int maxMessages)</code> • The methods that create the <code>TopicSession</code> object with processed transactions (For example, you cannot specify <code>true</code> in the first argument <code>transacted</code> of the <code>createTopicSession</code> method).

Furthermore, with the application components operating on the Web container or EJB container, do not create multiple active session objects using one connection. If an attempt is made to create an object using the `createSession` method when an active session object already exists, the `JMSEException` exception might be thrown.

7.12 Definitions in the DD

This section describes the items defined in the DD of the Message-driven Beans.

With the DD of the Message-driven Beans, you set up the items listed in the following table using the attributes specified beneath the `<activation-config>` tag.

Table 7-10: Items specified with the attributes beneath the `<activation-config>` tag

Attribute	Data type	Specifiable value	Default value	Explanation
<code>destination</code>	String	--	--	Specifies the name of the destination that sends the message to the Message-driven Bean. Specify the name set in Name of the Administered object for the CJMSP resource adapter in this attribute.
<code>destinationType</code>	String	<ul style="list-style-type: none"> <code>javax.jms.Queue</code> <code>javax.jms.Topic</code> 	--	Specifies the queue (<code>javax.jms.Queue</code>) or topic (<code>javax.jms.Topic</code>) as the destination type.
<code>messageSelector</code>	String	--	--	Specifies Message Selector to select the messages to be received.
<code>subscriptionName</code>	String	--	--	Specifies the name of the persistence subscriber. Make sure you specify this attribute when <code>Durable</code> is specified in the <code>subscriptionDurability</code> attribute.
<code>subscriptionDurability</code>	String	<ul style="list-style-type: none"> <code>Durable</code> <code>NonDurable</code> 	<code>NonDurable</code>	Specifies whether to set the persistence subscriber. If you want to set a persistence subscriber, specify <code>Durable</code> , and if you do not want to set a persistence subscriber, specify <code>NonDurable</code> . This attribute is only valid when <code>javax.jms.Topic</code> is specified in the <code>destinationType</code> attribute. Also, make sure you specify the following attributes when you specify <code>Durable</code> : <ul style="list-style-type: none"> <code>subscriptionName</code> <code>clientId</code>
<code>clientId</code>	String	--	--	Specifies the identifier of the persistence subscriber for connecting to CJMSP Broker. Make sure you specify this attribute when you specify <code>Durable</code> in the <code>subscriptionDurability</code> attribute.
<code>acknowledgeMode</code>	String	<ul style="list-style-type: none"> <code>Auto-acknowledge</code> <code>Dups-ok-acknowledge</code> 	<code>Auto-acknowledge</code>	Specifies the acknowledgement mode. For details on the modes to be specified, see the JMS specifications.
<code>endpointExceptionRedeliveryAttempts</code>	Integer	1 to 2147483647	6	Specifies the number of re-delivery attempts if an exception occurs during message delivery.
<code>sendUndeliverableMsgsToDMQ</code>	Boolean	<ul style="list-style-type: none"> <code>true</code> 	<code>true</code>	Specifies whether to register the un-delivered messages in the dead message queue when a

Attribute	Data type	Specifiable value	Default value	Explanation
sendUndeliverableMsgsToDMQ	Boolean	<ul style="list-style-type: none"> false 	true	<p>Message-driven Bean throws a runtime exception and when the re-delivery attempts exceed the count specified in the <code>endpointExceptionRedeliveryAttempts</code> attribute.</p> <p>If you want to register the un-delivered messages in the dead message queue, specify <code>true</code>, and if you do not want to register the un-delivered messages in the dead message queue, specify <code>false</code>.</p> <p>If you specify <code>false</code>, the applicable messages are re-delivered to the valid consumers by CJMSP Broker (might even be delivered to the same Message-driven Bean again).</p>
endpointPoolMaxSize	Integer	1 to 2147483647	1	Specifies the maximum endpoint pool value.
endpointPoolSteadySize	Integer	0 to 2147483647	1	Specifies the minimum endpoint pool value.
endpointExceptionRedeliveryInterval	Integer	1 to 2147483647	500	Specifies the interval for re-delivering a message to the endpoint, when an exception occurs, in milliseconds.

Legend:

--: No restrictions on the specifiable value or default value

7.13 Flow of execution environment setup

This section describes the flow of the execution environment setup when Cosminexus JMS Provider is used.

Use the management portal (Management Server) in advance to set up the execution environment of the J2EE applications that send and receive messages. For details on the procedure of setting up the J2EE application execution environment, see 3. *Setting Up and Deleting a System for Executing J2EE applications* in the *uCosminexus Application Server Management Portal User Guide*.

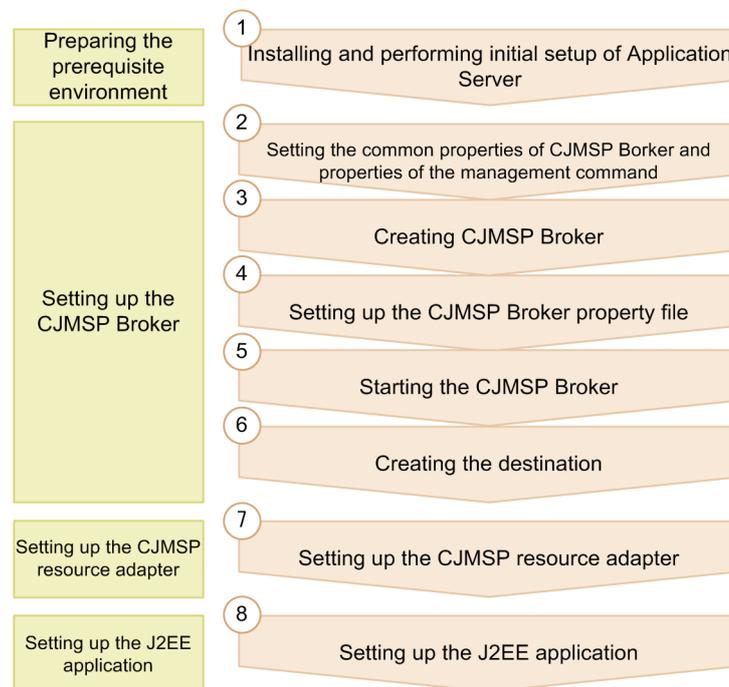
Note that CJMSP Broker is not managed by Management Server. Therefore, set up and operate CJMSP Broker using commands and definition files.

Tip

For the use cases of Cosminexus JMS Provider, see *Appendix E. Use Cases for Cosminexus JMS Provider*.

The following figure shows the flow of the execution environment setup.

Figure 7-13: Flow of the execution environment setup



The following points describe the flow of operations. The numbers in the description correspond to the numbers in the figure.

1. To prepare the prerequisite environment, install and initialize Application Server.
To set up CJMSP Broker and the J2EE server on different computers, install Application Server on each computer.
For details on Application Server installation and initialization, see 2.2.1 *Installing Application Server* in the *uCosminexus Application Server System Setup and Operation Guide*.
2. Set up the common properties and the management command properties of CJMSP Broker.
Set up the common properties for all CJMSP Brokers created in the same computer. Also, set up the properties related to the output of the management command log. For details on how to specify the settings, see 7.14.1 *Setting up the common properties and the management command properties of CJMSP Broker*.
3. Create CJMSP Broker.
For details on how to create CJMSP Broker, see 7.14.2 *Creating CJMSP Broker*.
4. Set up the individual properties of CJMSP Brokers.

Set up the individual properties for each CJMSP Broker. For details on how to specify the settings, see *7.14.3 Setting up the individual properties of CJMSP Brokers*.

5. Start CJMSP Broker.

For details on how to start CJMSP Broker, see *7.14.4 Starting CJMSP Broker*.

6. Create a destination for each CJMSP Broker.

For details on how to create a destination, see *7.14.5 Creating destinations*.

7. Set up and start the CJMSP resource adapter.

Use the HITACHI Connector Property file to set up the attributes for the CJMSP resource adapter.

After you set up the attributes, start the CJMSP resource adapter.

For details on how to set up and start the CJMSP resource adapter, see *7.15 CJMSP resource adapter settings*.

8. Set up and start J2EE applications.

Import each producer and consumer J2EE application and set up the attributes. Note that you use the DD, `cosminexus.xml`, or MessageDrivenBean property file to set the attributes for the Message-driven Beans of the consumer J2EE applications.

For details on how to set up and start J2EE applications, see *7.16 J2EE application settings*.

Tip

The files related to Cosminexus JMS Provider are stored in the following directory during installation:

- **In Windows**

Cosminexus-installation-directory\CC\cjmsp

- **In UNIX**

/opt/Cosminexus/CC/cjmsp

! Important note

Before you start the CJMSP resource adapter, CJMSP Broker must be running. If CJMSP Broker is stopped while the CJMSP resource adapter is starting, the CJMSP resource adapter tries reconnecting to CJMSP Broker every 120 seconds.

7.14 CJMSP Broker settings

This section describes the CJMSP Broker settings.

Note that there are two methods by which you specify the CJMSP Broker properties. If you specify different values using multiple methods, the settings are given priority in the order of the numbers described in the following table. We recommend that you use `commonconfig.property` if you want to specify common settings for all CJMSP Brokers.

Table 7-11: Method of setting up CJMSP Broker properties

Priority	Setting method	Location of settings	Scope of properties
1	Setting method using CJMSP Broker Property File	<code>config.property</code> ^{#1}	Valid only for individual CJMSP Brokers.
2	Setting method using CJMSP Broker Common Property File	<code>commonconfig.property</code> ^{#2}	Valid for all CJMSP Brokers in the same computer.

#1: Stored in the following directory by default. Note that the `var` directory can be used after CJMSP Broker is started first. For details on the CJMSP Broker Property File (`config.properties`), see 7.4 *config.properties (CJMSP Broker Property File)* in the *uCosminexus Application Server Definition Reference Guide*.

In Windows

`Cosminexus-installation-directory\CC\cjmsp\var\instances\CJMSP-Broker-name\props`

In UNIX

`/opt/Cosminexus/CC/cjmsp/var/instances/CJMSP-Broker-name/props`

#2: Stored in the following directory. For details on the CJMSP Broker Common Property File (`commonconfig.properties`), see 7.3 *commonconfig.properties (CJMSP Broker Common Property File)* in the *uCosminexus Application Server Definition Reference Guide*.

In Windows

`Cosminexus-installation-directory\CC\cjmsp\lib\props\broker`

In UNIX

`/opt/Cosminexus/CC/cjmsp/lib/props/broker`

Specify the properties for the output method of the management command (`cjmsicmd`) log in `admin.property`. `admin.property` is stored in the following directory.

For details on `admin.properties` (management command property file), see 7.2 *admin.properties (Management command property file)* in the *uCosminexus Application Server Definition Reference Guide*.

In Windows

`Cosminexus-installation-directory\CC\cjmsp\var\admin\config`

In UNIX

`/opt/Cosminexus/CC/cjmsp/var/admin/config`

! Important note

When you specify the CJMSP Broker settings, do not edit the files other than the following files from among the files beneath the `cjmsp` directory:

- `config.property`
- `commonconfig.property`
- `admin.property`

Also, do not change the access permissions by using the OS functionality even for these files.

Note that in a system that uses firewalls, you must specify the settings to permit Cosminexus JMS Provider to use ports.

For example, in Windows, usage can be permitted by executing the following command:

```
netsh firewall add allowedprogram program= "<COSMINEXUS_HOME>/jdk/bin/java.exe"
name="Java Virtual Machine" mode=ENABLE.
```

The port numbers used by Cosminexus JMS Provider are as follows. The string within () (parentheses) is the property that specifies the port number. You can specify the settings in `config.property`.

- Port number used by the CJMSP Broker port mapper (`imq.portmapper.port`)
- Port number used by the jms service (`imq.jms.tcp.port`)
- Port number used by the admin service (`imq.admin.tcp.port`)

7.14.1 Setting up the common properties and the management command properties of CJMSP Broker

The default values are set in the CJMSP Broker properties. As and when required, edit `commonconfig.property` to change the values of the properties specified for all CJMSP Brokers.

Also, edit `admin.property` to set up the properties related to the output of the management command log.

7.14.2 Creating CJMSP Broker

Create CJMSP Broker by executing the `cjmsbroker` command.

To create CJMSP Broker

1. Execute the `cd` command to move the current directory to another directory.

In Windows

```
Cosminexus-installation-directory\CC\cjmsp\bin
```

In UNIX

```
/opt/Cosminexus/CC/cjmsp/bin
```

2. Execute the `cjmsbroker` command.

To execute the command, specify the name of CJMSP Broker in the `-name` option.

Example of execution (when CJMSP Broker is created by specifying the name 'MyBroker')

```
cjmsbroker -name MyBroker
```

3. Execute the `cjmsicmd shutdown bkr` command.

The created CJMSP Broker stops. This is in order to set up the individual CJMSP Broker properties hereafter.

Example of execution

```
cjmsicmd shutdown bkr
```

When you execute `cjmsbroker` in step 2, a directory is created with the name ('MyBroker' in the example of execution) specified under `instances` in the `var` directory. If no name is specified, the default CJMSP Broker starts up.

Note that you specify the storage location of the `var` directory with the `-varhome` option of the `cjmsbroker` command.

! Important note

When you specify the directory with the `-varhome` option, specify the same value in the `-varhome` option when CJMSP Broker is started the second and subsequent times as well.

If you do not specify the `-varhome` option or specify a different location when CJMSP Broker is started the second and subsequent times, new instances are created beneath the specified `var` directory or the default directory even if you specify the same name in the `-name` option.

7.14.3 Setting up the individual properties of CJMSP Brokers

This subsection describes the setting up of the individual properties of CJMSP Brokers.

You can set up the individual properties of CJMSP Broker by specifying `config.property`. However, `config.property` can only be specified for CJMSP Broker that is started using the `cjmsbroker` command once and is created beneath the `var` directory.

An example of the specification of the properties using `config.property` is as follows:

Example of the specification of the properties using `config.property`

```
img.portmapper.port=7777
broker.logger.MessageLogFile.filenum=4
```

Meaning of the specified contents

- The port number used by the CJMSP Broker port mapper is 7777.
- The number of message log files is 4.

! Important note

For systems starting multiple CJMSP Brokers, make sure you change the value of the `img.portmapper.port` property. If CJMSP Broker using the default port (7676) exists, a new CJMSP Broker cannot be created.

7.14.4 Starting CJMSP Broker

This subsection describes the starting of CJMSP Broker.

The procedure is as follows. The current directory is presumed to have been moved to the following directory:

In Windows

```
Cosminexus-installation-directory\CC\cjmsp\bin
```

In UNIX

```
/opt/Cosminexus/CC/cjmsp/bin
```

1. Execute the `cjmsbroker` command.

The following is an example of execution of starting 'MyBroker':

Example of execution (to start 'MyBroker')

```
cjmsbroker -name MyBroker
```

7.14.5 Creating destinations

This subsection describes the creation of the destinations managed by CJMSP Broker.

The procedure is as follows. The current directory is presumed to have been moved to the following directory:

In Windows

```
Cosminexus-installation-directory\CC\cjmsp\bin
```

In UNIX

```
/opt/Cosminexus/CC/cjmsp/bin
```

1. Execute the `cjmsicmd create dst` command.

The following are the examples of execution for creating a queue and a topic:

Example of execution (to create the queue Queue1)

```
cjmsicmd create dst -t q -n Queue1
```

Example of execution (to create the topic Topic1)

```
cjmsicmd create dst -t t -n Topic1
```

Note that apart from creating a destination using commands, you create a destination using APIs in the applications. For details, see *7.8.2 Destination management and routing services*.

7.15 CJMSP resource adapter settings

The CJMSP resource adapter is stored as `cjmsra.rar` beneath the following directory. The default name is `Cosminexus_JMS_Provider_RA`.

In Windows

`Cosminexus-installation-directory\CC\cjmsp\lib`

In UNIX

`/opt/Cosminexus/CC/cjmsp/lib`

For the procedure from importing to starting a resource adapter, see *3.3 Setting up the resources* in the *uCosminexus Application Server Management Portal User Guide*.

You can set up the following items as attributes in the CJMSP resource adapter:

- Administered object attributes, such as `ConnectionFactory` and destination
- Transaction support levels
- Attributes related to log output

To change these attributes, edit the HITACHI Connector Property file specified in the CJMSP resource adapter.

The CJMSP resource adapter attributes include items specified for the resource adapters and items specified for the `ManagedConnectionFactory` classes. The following table describes the main attributes specified in the CJMSP resource adapter.

Table 7-12: Main attributes specified in the CJMSP resource adapter

Target	HITACHI Connector Property file tags	Specified items
Attributes for the resource adapters	<config-property> beneath <resourceadapter>	Attributes affecting the entire CJMSP resource adapter: <ul style="list-style-type: none"> • <code>connectionURL</code> • <code>reconnectEnabled</code> • <code>reconnectAttempts</code> • <code>reconnectInterval</code> Log-related attributes <ul style="list-style-type: none"> • <code>MsgLogLevel</code> • <code>MsgLogFileNum</code> • <code>MsgLogFileSize</code> • <code>ExpLogFileNum</code> • <code>ExpLogFileSize</code>
Attributes for the <code>ManagedConnectionFactory</code> classes	<config-property> beneath <connection-definition>	Attributes for the connection interfaces: <ul style="list-style-type: none"> • <code>clientId</code> • <code>reconnectEnabled</code> • <code>reconnectAttempts</code> • <code>reconnectInterval</code>
Administered object attributes	<adminobject-name>, <adminobject-interface>, and <adminobject-class> beneath <adminobject>	The name, destination type, and class information of the Administered object
	<config-property> beneath <adminobject>	Name of the destination (value specified in the destination attribute of a Message-driven Bean) <ul style="list-style-type: none"> • <code>Name</code>

Note that Cosminexus JMS Provider provides a template file for the HITACHI Connector Property file. Use this template file as and when required. For details on the storage destination of the template file, see *4.1.14 Template file of the HITACHI Connector Property file* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

7.16 J2EE application settings

Import the J2EE applications that use Cosminexus JMS Provider.

For the procedure from importing to starting a J2EE application, see *3.4 J2EE application settings* in the *uCosminexus Application Server Management Portal User Guide*.

You can specify the attributes to be set up for J2EE applications in the DD or `cosminexus.xml`. For the settings that must be specified in the DD for using Cosminexus JMS Provider, see *7.12 Definitions in the DD*. Note that to set up the attributes after importing the J2EE applications not containing `cosminexus.xml` into the J2EE server, use the property files and specify the settings using the server management commands.

7.17 Starting and stopping the system when Cosminexus JMS Provider is used

This section describes the procedure of starting and stopping the system when Cosminexus JMS Provider is used.

7.17.1 System starting procedure

To start a system using Cosminexus JMS Provider, use the following procedure. Note that steps 1 and 2 are in a random order.

! Important note

Before you start the CJMSP resource adapter, make sure you start CJMSP Broker. The CJMSP resource adapter cannot be started before starting CJMSP Broker. Also, if you stop the J2EE server when the CJMSP resource adapter is running, stop CJMSP Broker, and then restart the J2EE server, the CJMSP resource adapter fails to restart.

1. Start each execution environment process of the J2EE applications containing the J2EE server.

You can batch start a system set up with Smart Composer. For details, see *4.1.1 System starting procedure* in the *uCosminexus Application Server Management Portal User Guide*.

! Important note

At this point, the CJMSP resource adapter and the J2EE application are not running.

2. Start CJMSP Broker.

The following is an example of execution for starting CJMSP Broker 'MyBroker':

Example of execution

In Windows

```
Cosminexus-installation-directory\CC\cjmsp\bin\cjmsbroker -name MyBroker
```

In UNIX

```
/opt/Cosminexus/CC/cjmsp/bin/cjmsbroker -name MyBroker
```

Note that when you start CJMSP Broker, as and when required, change the Java heap size used by CJMSP Broker #.

3. Start the CJMSP resource adapter.

The following is an example of execution using the server management commands. In this example, the J2EE server name is 'cmx_MyWebSystem_unit1_J2EE_01', and the CJMSP resource adapter name is 'Cosminexus_JMS_Provider_RA'.

Example of execution

In Windows

```
Cosminexus-installation-directory\CC\admin\bin\cjstartrar  
"cmx_MyWebSystem_unit1_J2EE_01" -rename "Cosminexus_JMS_Provider_RA"
```

In UNIX

```
/opt/Cosminexus/CC/admin/bin/cjstartrar "cmx_MyWebSystem_unit1_J2EE_01" -  
rename "Cosminexus_JMS_Provider_RA"
```

4. Start the J2EE application that sends and receives the messages.

The following is an example of execution using the server management commands. Note that in this example, the J2EE server name is 'cmx_MyWebSystem_unit1_J2EE_01', and the J2EE application name is 'JMSSampleApp'.

Example of execution

In Windows

```
Cosminexus-installation-directory\CC\admin\bin\cjstartapp  
"cmx_MyWebSystem_unit1_J2EE_01" -name "JMSSampleApp"
```

In UNIX

```
/opt/Cosminexus/CC/admin/bin/cjstartapp "cmx_MyWebSystem_unit1_J2EE_01" -  
name "JMSSampleApp"
```

Concept of the Java heap size used by CJMSP Broker

By default, CJMSP Broker uses 192 MB of the Java heap. This value is too small to handle a large amount of messages, so increase the value as and when required.

If the memory size of the Java heap to be used is too small, the connections with the J2EE application need to be closed in order to allocate free memory, and the amount of processable messages decreases.

However, if you increase the maximum Java heap size that can be used with CJMSP Broker excessively, the physical memory of the system is affected and causes `OutOfMemoryError` in the system. If `OutOfMemoryError` occurs, apart from lowering the system performance, errors occur in CJMSP Broker in unexpected situations and affect the execution of other applications and services.

Considering all this, specify an appropriate value according to the executed environment.

You can specify the Java heap size to be used by CJMSP Broker in the `-vmargs` option of the `cjmsbroker` command when you start CJMSP Broker.

The following is an example of execution:

Example of execution

```
cjmsbroker -vmargs "-Xms256m -Xmx1024m"
```

In this example, CJMSP Broker uses 256 MB memory when starting, and the settings are specified to use maximum 1 GB of Java heap.

7.17.2 System stopping procedure

To stop a system using Cosminexus JMS Provider, use the following procedure. Note that steps 3 and 4 are in a random order.

1. Stop the J2EE application that sends and receives the messages.

The following is an example of execution using the server management commands. In this example, the J2EE server name is 'cmx_MyWebSystem_unit1_J2EE_01', and the J2EE application name is 'JMSSampleApp'.

Example of execution**In Windows**

```
Cosminexus-installation-directory\CC\admin\bin\cjstopapp
"cmx_MyWebSystem_unit1_J2EE_01" -resname "JMSSampleApp"
```

In UNIX

```
/opt/Cosminexus/CC/admin/bin/cjstopapp "cmx_MyWebSystem_unit1_J2EE_01" -
resname "JMSSampleApp"
```

2. Stop the CJMSP resource adapter.

The following is an example of execution using the server management commands. Note that in this example, the J2EE server name is 'cmx_MyWebSystem_unit1_J2EE_01', and the CJMSP resource adapter name is 'Cosminexus_JMS_Provider_RA'.

Example of execution**In Windows**

```
Cosminexus-installation-directory\CC\admin\bin\cjstoprar
"cmx_MyWebSystem_unit1_J2EE_01" -resname "Cosminexus_JMS_Provider_RA"
```

In UNIX

```
/opt/Cosminexus/CC/admin/bin/cjstoprar "cmx_MyWebSystem_unit1_J2EE_01" -
resname "Cosminexus_JMS_Provider_RA"
```

3. Stop CJMSP Broker.

The following is an example of execution for stopping CJMSP Broker 'MyBroker':

Example of execution**In Windows**

```
Cosminexus-installation-directory\CC\cjmsp\bin\cjmsicmd shutdown bkr
```

In UNIX

```
/opt/Cosminexus/CC/cjmsp/bin/cjmsicmd shutdown bkr
```

Tip

When you want to stop CJMSP Broker, identify the target CJMSP Broker using the host name and the port number. If you execute the `cjmsicmd shutdown bkr` command without specifying the port number, the stop processing will be executed for CJMSP Broker that is using the default port 7676. To stop CJMSP Broker that is using another port, specify the host name and the port number in the `-b` option. The following is an example of execution:

Example of execution (to stop CJMSP Broker that is using the port 7777)

```
cjmsicmd shutdown bkr -b localhost:7777
```

For details, see *cjmsicmd shutdown bkr (Stopping CJMSP Broker)* in the *uCosminexus Application Server Command Reference Guide*.

4. Stop each execution environment process of the J2EE applications containing the J2EE server. You can batch stop a system set up with Smart Composer. For details, see *4.1.3 System stopping procedure* in the *uCosminexus Application Server Management Portal User Guide*.

7.17.3 Checking the CJMSP Broker status

This subsection describes the status transition and status checking methods of CJMSP Broker. CJMSP Broker has three types of statuses. The status changes due to operations such as the execution of commands.

You can check the CJMSP Broker status by using the `cjmsicmd list bkr` command. For details, see *cjmsicmd list bkr (Displaying the list of CJMSP Brokers)* in the *uCosminexus Application Server Command Reference Guide*.

The following figure shows the transition of the CJMSP Broker status due to the execution of commands.

Figure 7-14: Status transition of CJMSP Broker

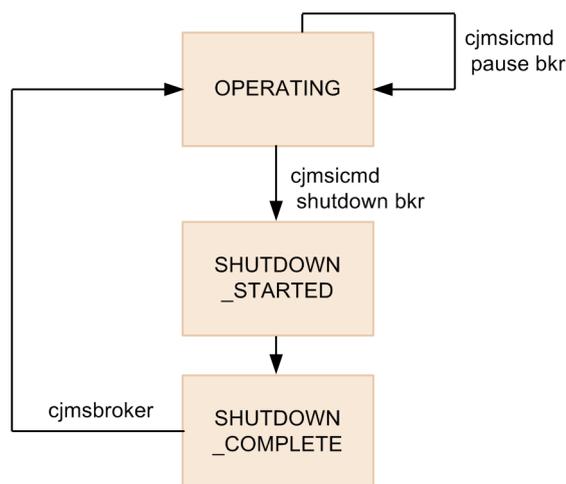


Table 7-13: CJMSP Broker status

CJMSP Broker status	Explanation
OPERATING	The status in which CJMSP Broker can process the messages. This status is reached when you start CJMSP Broker using the <code>cjmsbroker</code> command. Note that when you execute the <code>cjmsicmd pause bkr</code> command for the OPERATING CJMSP Broker, the status does not change.
SHUTDOWN_STARTED	The status in which CJMSP Broker starts the stop processing (shut down). CJMSP Broker is shut down immediately after or some minutes later. This status is reached when you execute the <code>cjmsicmd shutdown bkr</code> command.
SHUTDOWN_COMPLETE	The status in which the stop (shut down) processing of CJMSP Broker finishes. All the resources are released. Whether CJMSP Broker is shut down depends on whether any processes are being executed.

7. Cosminexus JMS Provider

CJMSP Broker status	Explanation
SHUTDOWN_COMPLETE	This status is reached when you execute the <code>cjmsicmd shutdown bkr</code> command.

7.18 Checking the troubleshooting information

This section describes the troubleshooting information to be used for errors when Cosminexus JMS Provider is used.

With Cosminexus JMS Provider, the following three components output the log:

- CJMSP resource adapter
- CJMSP Broker
- Management commands (`cjmsicmd`)

These components output the message log and exception log. The message log and exception log are output to the log files for each component.

The log format is the Hitachi Trace Common Library format.

Message log

The statuses occurring due to the operations executed from J2EE applications are output to the message log.

The output message type is set based on the log level.

The following table describes the meaning of the log levels in Cosminexus JMS Provider.

Table 7-14: Meaning of the log levels in Cosminexus JMS Provider

Log level	Meaning
ERROR	<p>A message log level that indicates the occurrence of a serious problem leading to a system error.</p> <p>This log level outputs minimum information, containing only the error and exception information.</p> <p>If this log level is specified, only the ERROR level messages are output to the log.</p>
WARNING	<p>A message log level that indicates a status in which precautions need to be taken, but not leading to a system error.</p> <p>This log level is output when the processing terminates normally and there is a problem in the status.</p> <p>If this log level is specified, the ERROR level and WARNING level messages are output to the log.</p>
INFO	<p>This log level outputs the metrics and other messages.</p> <p>This log level outputs information that indicates the flow of processing and the type of operations performed during normal processing.</p> <p>If this log level is specified, the ERROR level, WARNING level, and INFO level messages are output to the log.</p>

Note

The important messages are output to the console or the log file regardless of the log level settings.

Exception log

The information about the occurrence of errors or exceptions (exception messages) during the execution of J2EE applications is output to the exception log.

For details on the messages output to the log files, see 4. *KDAN (Messages Output by Cosminexus JMS Provider)* in the manual *uCosminexus Application Server Messages*. Also, for details on the Hitachi Trace Common Library format, see 5.2.1 *Log output format and output items in the Hitachi Trace Common Library format* in the *uCosminexus Application Server Maintenance and Migration Guide*.

Reference note

- With Cosminexus JMS Provider, you can also obtain the PRF trace. For the PRF trace collection points when Cosminexus JMS Provider is used, see 8. *Trace Collection Points and PRF Trace Collection Levels of the Trace Based Performance Analysis* in the *uCosminexus Application Server Maintenance and Migration Guide*.
- If CJMSP Broker stops with a JavaVM error, the thread dump or core file is output to the following directory:

In Windows

Cosminexus-installation-directory\CC\cjmsp\var\instances

In UNIX

/opt/Cosminexus/CC/cjmsp/var/instances

7.19 Notes on using Cosminexus JMS Provider

This section describes the notes on using Cosminexus JMS Provider.

(1) Default Cosminexus JMS Provider operations

By default, Cosminexus JMS Provider is operated as follows:

- When the number of messages or the message size managed by CJMSP Broker reaches the maximum value set for all the physical destinations, CJMSP Broker refuses to receive the latest messages in order to allocate the memory resources. Also, CJMSP Broker notifies the client that sent a message about the refusal to receive the message only when the message is a persistence message.
- The messages with expired validity periods that are sent to the dead message queue are held for 60 seconds. The messages exceeding this period cannot be recovered from the dead message queue.
- The temporary destinations that are no longer used are destroyed after 120 seconds.
- The storage of persistence data to a file is executed asynchronously. Therefore, the data might be lost if CJMSP Broker fails or stops suddenly.

(2) Notes on the priority order of message processing

The sending of messages with Cosminexus JMS Provider does not guarantee that the messages with a high priority are executed first. For example, when three messages, 'message1', 'message2', and 'message3' with the priority order set as 1, 2, and 3 are sent from the producer, the messages might be processed in a different order, 'message1', 'message3', and 'message2', with the consumer.

The differences in the priority order and processing order occur due to the following causes:

- If the producer first sends a low-priority message, and then sends a high-priority message, before the message sent later reaches the consumer, the consumer receives the low-priority message and starts the processing.
- If the validity period of a high-priority message expires before the consumer receives the message, the high-priority message is not processed.
- Even if the producer sends messages according to the priority order, the order in which the messages are sent to the consumer might vary due to causes such as delay in the internal processing.
- If a high-priority message is not a persistence message, that message is lost if an error occurs in CJMSP Broker. In such a case, only the low-priority persistence messages are processed after CJMSP Broker is restarted.

(3) Handling of messages with expired validity periods

If you change the system time of a computer on which a destination exists, the client might receive an expired message.

The validity period of the message is set by the method that sends a message to the `JMSExpiration` header field. If you change the system time settings by mistake, the messages with expired validity periods might be sent to the consumer.

(4) Order of the message properties

The processing order of the message properties is not guaranteed.

The message properties are not managed in a structure that maintains the order. Therefore, if, for example, there are properties 'property 1', 'property 2', and 'property 3', and the properties are sent to the destination in the same order, the properties are not necessarily processed in the same order. However, the properties and their values do not change before sending and when received.

(5) Handling of the byte messages in the message properties

Do not use the byte messages with the message properties.

The data of all the data types can be read with the byte messages. For example, if a message containing a `long` type value is invoked and processed with a `short` type method, the first two bytes are read successfully although the value is semantically invalid. This processing is prohibited with the other message data types.

With Cosminexus JMS Provider, the data types of the data are stored, and the data is managed so that the application of the correct transformation rules is supported.

(6) Handling of messages in the `BytesMessage` interface

When you use the `writeUTF()` method of the `BytesMessage` interface, do not write messages larger than 65,535 bytes. An exception occurs if a character string of length greater than 65,535 bytes is passed to the `writeUTF()` method.

(7) Handling of messages in the dead message queue

If rollback is repeated for a message received from the dead message queue, that message is not sent and remains in the dead message queue as is. To receive the messages remaining in the dead message queue again, restart CJMSP Broker.

8

Using JavaMail

This chapter describes the SMTP provider and SMTPS provider-specific session properties, POP3 provider-specific session properties, and notes on using JavaMail.

8.1 Organization of this chapter

This chapter describes the SMTP provider and SMTPS provider-specific session properties, and POP3 provider-specific session properties.

The following table describes the organization of this chapter.

Table 8-1: Organization of this chapter (Session properties)

Category	Title	Reference location
Implementation	Session properties	8.2
Notes	Notes on using JavaMail	8.3

Note: The functionality-specific explanation is not available for "Explanation", "Settings" and "Operations".

8.2 Session properties

With JavaMail, you specify the mail sending and receiving-related settings as session properties when the `javax.mail.Session` class is created.

The session properties include the following types:

- Session properties provided with JavaMail APIs
- Session properties provided by the SMTP provider and SMTPS provider stated in the JavaMail specifications
- Session properties provided by the POP3 provider stated in the JavaMail specifications
- SMTP provider and SMTPS provider-specific session properties
- POP3 provider-specific session properties

The values set for the session properties must be specified as character strings enclosed within " (double quotation marks).

Note that this subsection describes the SMTP provider and SMTPS provider-specific session properties and the POP3 provider-specific session properties.

8.2.1 Session properties for connecting to the SMTP server

This subsection describes the SMTP provider and SMTPS provider-specific session properties.

The following table lists and describes the SMTP provider and SMTPS provider-specific session properties. Note that if `smtps` is specified as the protocol to be used, the `smtp` part of the property name becomes `smtps`.

Table 8-2: List of the SMTP provider and SMTPS provider-specific session properties

No.	Property	Default value	Valid value	Operation when an invalid value is specified
1	<code>mail.smtp.allow8bitmime</code>	<code>false</code>	<code>true, false</code> (not case-sensitive)	The message KDJE59100-W is output during the send processing to the mail server. The default value is used.
2	<code>mail.smtp.auth</code>	<code>false</code>	<code>true, false</code> (not case-sensitive)	The message KDJE59100-W is output when connecting to the mail server. The default value is used.
3	<code>mail.smtp.connectiontimeout</code>	0 (Wait infinitely)	0 to 2147483647 (unit: millisecond)	The message KDJE59100-W is output when connecting to the mail server. The default value is used.
4	<code>mail.smtp.dsn.notify</code>	None	<ul style="list-style-type: none"> • Specify either <code>SUCCESS</code>, <code>FAILURE</code>, or <code>DELAY</code>, or multiple values demarcated by , (comma) • <code>NEVER</code> 	The specified value is used as is. If the value is considered invalid in the mail server, the <code>RCPT</code> command fails, and the sending of the mail is cancelled. The mail server processing depends on the mail server.
5	<code>mail.smtp.dsn.ret</code>	None	<code>FULL, HDRS</code>	The specified value is used as is. If the value is considered invalid in the mail server, the <code>MAIL</code> command fails, and the sending of the mail is cancelled. The mail server processing depends on the mail server.
6	<code>mail.smtp.ehlo</code>	<code>true</code>	<code>true, false</code>	The message KDJE59100-W is output when connecting to the mail server.

No.	Property	Default value	Valid value	Operation when an invalid value is specified
6	mail.smtp.ehlo	true	(not case-sensitive)	The default value is used.
7	mail.smtp.from	Value specified in <mail.from>	Character string	JavaMail or the mail server might return an exception. The cases in which the mail server returns an exception depend on the mail server.
8	mail.smtp.localhost	Return value of <code>java.net.InetAddress.getLocalHost().getHostName()</code>	Character string	The specified value is used as is.
9	mail.smtp.noop.strict	true	true, false (not case-sensitive)	The message KDJE59100-W is output when the <code>javax.mail.Transport</code> object is obtained. The default value is used.
10	mail.smtp.quitwait	false	true, false (not case-sensitive)	The message KDJE59100-W is output when the <code>javax.mail.Transport</code> object is acquired. The default value is used.
11	mail.smtp.saslrealm	None	Character string	The specified value is used as is.
12	mail.smtp.sendpartial	false	true, false (not case-sensitive)	The message KDJE59100-W is output during the send processing to the mail server. The default value is used.
13	mail.smtp.timeout	0 (Wait infinitely)	0 to 2147483647 (unit: millisecond)	The message KDJE59100-W is output when connecting to the mail server. The default value is used.

The following is a description of the properties:

(1) mail.smtp.allow8bitmime

This property specifies whether to convert the quoted-printable and base64 encoded message text to an 8-bit encoding.

If you specify `true` and if the following conditions are fulfilled, the quoted-printable and base64 encoded message text is converted to an 8-bit encoding.

- The server supports an 8-bit transmission.
- The quoted-printable and base64 encoded message text conforms to RFC2045.

If you specify `false`, the text is not converted to an 8-bit encoding.

(2) mail.smtp.auth

This property specifies whether to perform user authentication using the `AUTH` command.

If you specify `true`, the user authentication is performed by using the `AUTH` command. If you specify `false`, the user authentication is not performed using the `AUTH` command.

(3) mail.smtp.connectiontimeout

This property specifies the timeout value until a connection is established (unit: millisecond).

If you specify 0, the system continues to wait infinitely.

When this property is specified, the timeout value is monitored by using threads. If the threads cannot be created due to the policy or system constraints, the timeout value is not monitored.

If the timeout value is greater than the timeout value based on the TCP resend timer of the OS, or if the timeout is not to be monitored, the timeout value based on the TCP resend timer is enabled. Note that the timeout based on the resend timer differs depending on the OS.

(4) mail.smtp.dsn.notify

This property specifies the NOTIFY option of the RCPT command.

This option specifies the conditions for generating the DSN (Delivery Status Notification). You can combine and specify the values described in the following table, using commas to demarcate the values.

Table 8-3: Values that can be specified in mail.smtp.dsn.notify

No.	Value	Specified contents
1	NEVER	The DSN is not returned to the sender regardless of the delivery status.
2	SUCCESS	The DSN is generated when the message is delivered successfully.
3	FAILURE	The DSN is generated when the delivery of the message fails.
4	DELAY	The DSN is generated when the delivery of the message is delayed.

If no value is specified for the DSN generation, this might be interpreted as NOTIFY=FAILURE or NOTIFY=FAILURE, DELAY has been specified in RFC1891 and depends on the implementation of the server.

(5) mail.smtp.dsn.ret

This property specifies the value of the RET addition parameter of the MAIL command.

You can combine and specify the values described in the following table.

Table 8-4: Values that can be specified in mail.smtp.dsn.ret

No.	Value	Specified contents
1	FULL	The entire sent message is included in the message reporting delivery failure.
2	HDRS	Only the header of the sent message is included in the message reporting delivery failure.

If you do not specify the property, the contents to be included in the message reporting delivery failure depend on the implementation of the server.

(6) mail.smtp.ehlo

This property specifies whether to use the EHLO command.

If you specify `true`, the EHLO command will be used. If an error occurs in the EHLO command, the HELO command is used as a substitute. If you specify `false`, the EHLO command is not used.

(7) mail.smtp.from

This property specifies the email address to be used in `reverse-path` in the SMTP and SMTPS MAIL command.

(8) mail.smtp.localhost

This property specifies the domain name of the local host used in the SMTP and SMTPS HELO command or EHLO command.

This property need not be specified if the JDK and name service is defined appropriately.

(9) mail.smtp.noop.strict

This property is used to change the response that determines the connected status when the status of connection with the mail server is checked.

If you specify `true`, the connected status is assumed when the mail server response for the `NOOP` command is 250. If you specify `false`, the connected status is assumed when the mail server response for the `NOOP` command is any positive integer value except 421.

(10) mail.smtp.quitwait

This property specifies whether the system will await a response for the `QUIT` command.

If you specify `true`, the system awaits a response for the `QUIT` command. If you specify `false`, the connection is closed immediately after the `QUIT` command is sent.

(11) mail.smtp.saslrealm

This property specifies the realm to be used for the DIGEST-MD5 authentication.

If you omit this property, the first realm included in the challenge from the server is used. If the challenge does not contain a realm, the connection destination host is used as the realm.

(12) mail.smtp.sendpartial

This property specifies the operation to be performed when the email address specified in a message contains an incorrect email address.

If you specify `true`, the message is sent and then the email address with error is reported with `SendFailedException`. If you specify `false`, the message is not sent.

(13) mail.smtp.timeout

This property specifies the timeout value (unit: millisecond) for communication (read) with the SMTP server.

If you specify `0`, the system continues to wait infinitely.

If the timeout value is greater than the timeout value based on the TCP resend timer of the OS, or if the timeout is not to be monitored, the timeout value based on the TCP resend timer is enabled. Note that the timeout based on the resend timer differs depending on the OS.

8.2.2 Session properties for connecting to the POP3 server

This subsection describes the POP3 provider-specific session properties.

The following table lists and describes the POP3 provider-specific session properties.

Table 8-5: List of POP3 provider-specific session properties

No.	Property	Default value	Valid value	Operation when an invalid value is specified
1	<code>mail.pop3.connectiontimeout</code>	0 (Wait infinitely)	0 to 2147483647 (unit: millisecond)	The message KDJE59100-W is output when connecting to the mail server. The default value is used.
2	<code>mail.pop3.resetbeforequit</code>	<code>false</code>	<code>true</code> , <code>false</code> (not case-sensitive)	The message KDJE59100-W is output when the <code>javax.mail.Store</code> object is acquired. The default value is used.

No.	Property	Default value	Valid value	Operation when an invalid value is specified
3	<code>mail.pop3.timeout</code>	0 (Wait infinitely)	0 to 2147483647 (unit: millisecond)	The message KDJE59100-W is output when connecting to the mail server. The default value is used.

The following is a description of the properties:

(1) `mail.pop3.connectiontimeout`

This property specifies the timeout value until a connection is established (unit: millisecond).

If you specify 0, the system continues to wait infinitely.

If the timeout value is greater than the timeout value based on the TCP resend timer of the OS, or if the timeout is not to be monitored, the timeout value based on the TCP resend timer is enabled. Note that the timeout based on the resend timer differs depending on the OS.

(2) `mail.pop3.rsetbeforequit`

This property specifies whether to issue the `RSET` command before the `QUIT` command when you close the folder.

If you specify `true`, the `RSET` command is issued before the `QUIT` command, and the delete mark assigned to the message is removed. For servers in which the delete mark is automatically assigned, you can prevent the deletion of the message without a user request.

If you specify `false`, the `RSET` command is not issued before the `QUIT` command.

(3) `mail.pop3.timeout`

This property specifies the timeout value (unit: millisecond) for communication (read) with the POP3 server.

If you specify 0, the system continues to wait infinitely.

If the timeout value is greater than the timeout value based on the TCP resend timer of the OS, or if the timeout is not to be monitored, the timeout value based on the TCP resend timer is enabled. Note that the timeout based on the resend timer differs depending on the OS.

8.3 Notes on using JavaMail

This section describes the notes on using JavaMail.

(1) Occurrence of method exceptions

The exceptions do not occur as per the JavaMail specifications in the methods of the following classes:

No.	Class name	Method name
1	<code>javax.mail.Message</code>	<code>setFlag, setFlags</code>
2	<code>javax.mail.Folder</code>	<code>setFlags</code>

Even if you change the flag value using the methods described in the table for the `javax.mail.Message` object acquired from the `javax.mail.Folder` object for which `READ_ONLY` is specified, the flag is set correctly as in the case when `READ_WRITE` is specified.

However, in the case of the POP3 server, even if you open the folder with `READ_ONLY` and set the `DELETED` flag for a message, the mail is not actually deleted.

(2) Operating the SEEN flag in POP3

With the JavaMail specifications, if you use the `getInputStream` method and `getContent` method of `Message`, the specifications mention that the `SEEN` flag is set up, but the `SEEN` flag is not actually set up.

(3) JavaMail providers

The SMTP, SMTPS, and POP3 providers, provided by default by Application Server, are the only JavaMail providers that can be used with Application Server. The operations might not function normally if you change to another provider by using the `javamail.providers` file.

(4) Response codes

Unlike the rigorous RFC specifications, the response codes considered invalid in RFC specifications might not be considered as invalid by JavaMail. Therefore, instead of the messages (KDJE59111-E or KDJE59112-E) that indicate the receipt of a non-RFC specification invalid response, the messages (KDJE59107-E, KDJE59108-E, KDJE59109-E, or KDJE59110-E) that indicate the receipt of an error value response code might be output.

(5) Notes on connecting to the mail server and receiving mails using an application

After you connect to the mail server and receive mails using an application, if you stop or reload the application without invoking the following methods, the KDJE59106-E message might be output:

- `close` method of the `javax.mail.Store` class
- `close` method of the `javax.mail.Folder` class

In this case, check whether the respective `close` methods have been invoked.

Note that the KDJE59106-E message is output, but the stop and reload processing of the application is not affected. The JavaMail operations are also not affected.

(6) Differences with the JavaMail specifications

If only `Reply-To:` is specified in the mail header and the `Reply-To` header field, and if the address is not specified, Application Server JavaMail returns an array of the `javax.mail.Address` object with length 0 to the `getReplyTo` method of the `javax.mail.MimeMessage` class, instead of returning the return value of the `getFrom` method.

9

Using CDI with Application Server

This chapter provides an overview of the CDI functionality, and describes the implementation methods and notes on the CDI functionality.

9.1 Organization of this chapter

This chapter describes the CDI functionality. The following table describes the organization of this chapter.

Table 9-1: Organization of this chapter (Using CDI with Application Server)

Category	Title	Reference location
Explanation	Overview of the CDI	9.2
	Application development using the CDI	9.3
Implementation	Setting up the class path (Development environment settings)	9.4
	Settings for using the CDI (Changing the security settings)	9.5
	Using the debug log (Check development log)	9.6
Settings	Setting up the execution environment	9.7
Notes	Notes on using the CDI	9.8

Note: The functionality-specific explanation is not available for "Operations".

9.2 Overview of the CDI

(1) CDI

The CDI is one of the Dependency Injection (DI) specifications. CDI 1.0 is supported with Application Server.

If you use the CDI, you can manage the DI of the application objects, and the life-cycle from the generation to the destruction of the DI target objects. The following CDI functionality is provided with Application Server:

- Resolving Beans based on the class types
- Resolving Beans based on the names (character string)
- Generating and destroying instances based on the contexts specified by annotations
- Automatically injecting the other Bean instances specified using the EL (Expression Language) and annotations
- Life-cycle callback interceptor and interceptors

Tip

The CDI functionality enhances the developability during application development by implementing injections between application resources. To implement the CDI functionality, you must change the J2EE server security settings so that any application resource can be injected. For details on changing the security settings, see *9.5 Settings for using CDI (Changing the security settings)*.

(2) Integrating with Bean Validation

With Application Server, you can use the Bean Validation functionality to simplify the validation of the values entered in the CDI application.

For details on the use of Bean Validation, see *10. Using Bean Validation with Application Server*.

9.3 Application development using the CDI

This section describes the `beans.xml` files and the application versions required for application development using the CDI.

9.3.1 CDI-target applications

The CDI-target applications are the applications containing the J2EE modules that have the `beans.xml` files. The following table describes the storage destinations of the `beans.xml` files for each J2EE module type.

Table 9-2: Storage destinations of the `beans.xml` files for each J2EE module type

Types of J2EE modules	Storage destination of <code>beans.xml</code>
EJB-JAR	<code>META-INF/beans.xml</code>
Library JAR	<code>META-INF/beans.xml</code>
WAR	<code>WEB-INF/beans.xml</code>

The CDI-target J2EE modules are not managed by the CDI if the `beans.xml` files are not stored. Make sure you store the `beans.xml` files at the storage destinations described in the table.

(1) The following table describes the versions of the components that can be set as the CDI targets and the objects that can use the CDI.

Table 9-3: Versions of the components that can be set as the CDI targets and the objects that can use the CDI

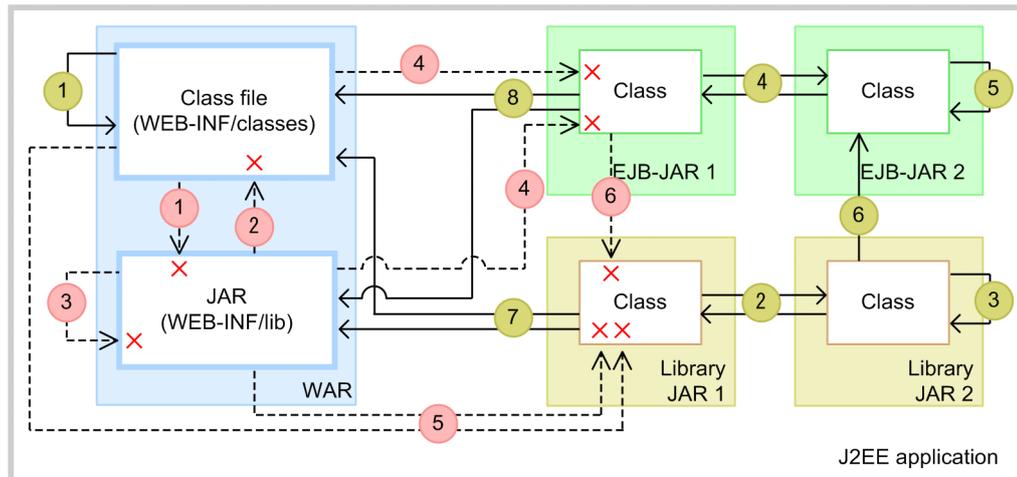
Component types	Supported component versions	Usable objects
EJB	EJB 3.1 (The <code>ejb-jar.xml</code> version is 3.1)	Session Bean
Servlet	Servlet 3.0 (The <code>web.xml</code> version is 3.0)	<ul style="list-style-type: none"> • Servlet • Listener • Filter
JSP	JSP 2.1 (The <code>web.xml</code> version is 3.0)	Objects supporting the EL syntax
JSF	JSF 2.1 (The <code>web.xml</code> version is 3.0, and JSF framework supported by Application Server)	Objects supporting the EL syntax

9.3.2 Injective relationship of the CDI-target J2EE modules

This subsection describes the injective relationship of the CDI-target J2EE modules.

The following figure shows the injective relationship of the CDI-target J2EE modules.

Figure 9-1: Injective relationship of the CDI-target J2EE modules



Legend:

— n → : Can be inserted

-- n → X : Cannot be inserted

The relationships that can be injected and cannot be injected as CDI targets are as follows:

Relationships that can be injected as CDI targets

1. Injection from WEB-INF/classes in a WAR to WEB-INF/classes in the same WAR
2. Injection from the classes included in a library JAR to the classes included in another library JAR
3. Injection from the classes included in a library JAR to the classes included in the same library JAR
4. Injection from the classes included in an EJB-JAR to the classes included in another EJB-JAR (however, an EJB cannot be injected)
5. Injection from the classes included in an EJB-JAR to the classes included in the same EJB-JAR (however, an EJB cannot be injected)
6. Injection from the classes included in a library JAR to the classes included in an EJB-JAR
7. Injection from the classes included in a library JAR to the classes included in a WAR (JAR beneath WEB-INF/classes and WEB-INF/lib)
8. Injection from the classes included in an EJB-JAR to the classes included in a WAR (JAR beneath WEB-INF/classes and WEB-INF/lib)

Relationships that cannot be injected as CDI targets

1. Injection from the classes included in WEB-INF/classes in a WAR to the classes included in a JAR beneath WEB-INF/lib in the same WAR
2. Injection from the classes included in a JAR beneath WEB-INF/lib in a WAR to the classes included in WEB-INF/classes in the same WAR
3. Injection from the classes included in a JAR beneath WEB-INF/lib in a WAR to the classes included in a JAR beneath WEB-INF/lib in the same WAR
4. Injection from the classes included in a WAR (JAR beneath WEB-INF/classes and WEB-INF/lib) to the classes included in an EJB-JAR
5. Injection from the classes included in a WAR (JAR beneath WEB-INF/classes and WEB-INF/lib) to the classes included in a library JAR
6. Injection from the classes included in an EJB-JAR to the classes included in a library JAR

9.3.3 Notes on development

Note the following when you use the CDI to develop applications:

- Handling constructors with applications that use the CDI

You cannot specify arguments for the constructors of the `EJB Bean class`, `Servlet class`, `Listener class`, and `Filter class` that use the CDI annotations.

9.4 Setting up the class path (Development environment settings)

This section describes the class path settings required for using the CDI.

9.4.1 File storage destinations

The following table describes the CDI-related files installed when Application Server is installed and the storage destinations of the files.

Table 9-4: Storage destination of the CDI library

File name	Class path
cjsf.jar	<i>Application-Server-installation-directory</i> \CC\lib\cjsf.jar
cjstl.jar	<i>Application-Server-installation-directory</i> \CC\lib\cjstl.jar

Note: In UNIX, replace *Application-Server-installation-directory* with /opt/Cosminexus, and \ with /.

9.4.2 Class path settings

To operate the applications that use the CDI with Application Server, you must set the class path in the `add.class.path` key of `usrconf.cfg` (option definition file for Java applications) for the J2EE servers.

An example is as follows. This example is for Windows.

```
add.class.path=Application-Server-installation-directory\CC\lib\cjsf.jar
add.class.path=Application-Server-installation-directory\CC\lib\cjstl.jar
```

Important note

Do not specify a different library version in the class path at the same time because if such a version is specified, the application might perform an unexpected operation. Specify a library suitable to the application version in the class path.

9.5 Settings for using the CDI (Changing the security settings)

This section describes the changing of security settings required for using the CDI.

To use the CDI, you must change the security settings with one of the following methods:

- Changing the security policy settings
- Releasing SecurityManager

The description of the methods is as follows:

(1) Changing the security policy settings

Add permission `java.security.AllPermission;` in `server.policy`. The following is an example of editing the settings:

```
// Grant minimal permissions to everything else:
// EJBs
// client implementation classes
grant {
  permission java.util.PropertyPermission "*", "read";
  permission java.lang.RuntimePermission "queuePrintJob";
  permission java.net.SocketPermission "*", "connect";
  permission java.security.AllPermission;
};
```

Specify `server.policy` after you set up the J2EE server.

(2) Releasing SecurityManager

Specify the `-nosecurity` option for the `cjstartsv` command when you start the J2EE server to release SecurityManager.

```
# cjstartsv server-name -nosecurity
```

9.6 Using the debug log (check development log)

With application development using the CDI, you can use the **check development log** as the debug log.

If there are errors or defects in the CDI application being developed, you can use the check development log to check the cause and to understand the detailed operations of the target functionality.

With the applications that use the CDI, the development-related messages are output to the check development log. The messages that require confirmation with the applications that use the CDI can be identified from the class name output to the message. The following table describes the components that output the messages and the class name that is output.

Table 9-5: Class names output to the check development log for the applications that use the CDI

Component name	Output class name	Output contents
CDI	com.hitachi.software.cdi.weld.- com.cosminexus.cc.lib.jboss_interceptor.v2_0_0_CR1.-	<ul style="list-style-type: none"> • Error-related messages • Messages indicating a settings mistake • Exception stack trace • Messages related to the operating state of the functionality • Messages related to the internal state

Note that the check development log is not output by default. Change the settings as and when required.

For details on the settings to output the check development log, and the output destination, output format, and log levels of the check development log, see *24.4 Messages output for the check development log* in the manual *uCosminexus Application Server Messages*.

Also, for details on the check development log for Bean Validation used with the CDI applications, see *10.6 Using the debug log (check development log)*.

9.7 Setting up the execution environment

The execution environment settings for the applications that use the CDI are similar to the development environment settings.

See the following locations to specify the settings:

- Class path settings
9.4 Setting up the class path (Development environment settings)
- Changing the security settings
9.5 Settings for using the CDI (Changing the security settings)

9.8 Notes on using the CDI

The notes on using the CDI are as follows:

- **CDI extension interface (Portable Extension API)**

The CDI extension interface (Portable Extension API) cannot be used with Application Server.

- **Notes on using the CDI with Application Server (differences from the standard specifications)**

- This point describes the scope of Application Server support when you combine and use the Servlet 3.0 functionality and the CDI.

The CDI annotations can be used with `Servlet`, `Listener`, and `Filter` defined in `web.xml`, and `Servlet`, `Listener`, and `Filter` defined in annotations.

The CDI annotations can only be used with `Servlet` and `Filter` dynamically defined by using the `contextInitialized` method of `javax.servlet.ServletContextListener` for the `Servlet` listener, and the following methods of `javax.servlet.ServletContext`:

- `addFilter(java.lang.String filterName, java.lang.Class<? extends Filter> filterClass)`
- `addFilter(java.lang.String filterName, java.lang.String className)`
- `addServlet(java.lang.String servletName, java.lang.Class<? extends Servlet> servletClass)`
- `addServlet(java.lang.String servletName, java.lang.String className)`

The usage of the CDI annotations in `Servlet`, `Filter`, and `Listener` dynamically defined by using the other APIs is not supported.

- If `beans.xml` is allocated to the `META-INF` directory specified in the JavaVM class path, the classes in the JavaVM class path are not managed by the CDI.
- If `beans.xml` is allocated to the `META-INF` directory of the resource adapter (RAR), the `beans.xml` file is not read and the resource adapter is not recognized as a CDI module.
- If `beans.xml` is allocated to the `META-INF` directory in the JAR allocated within `WEB-INF/lib` of the WAR, the `beans.xml` file is not read and the JAR file is not recognized as a CDI module. Note that `ManagedBean` can be injected into the JSPs included in `META-INF/resources`, existing in the JAR files stored beneath `WEB-INF/lib`.
- The usage of `@EJB`, `@Resource`, `@PersistenceContext`, and `@PersistenceUnit` is not supported with the CDI `ManagedBean`.
- You cannot inject Session Beans (including the ones where the local interface is omitted) by using the CDI annotations.
- If you use the CDI annotations to inject an EJB, the error message KDJE59316-E is output and the injection operation fails.
- You cannot inject an EJB, with `@Named` assigned, into the JSF and JSP by using the Expression Language. If you use the Expression Language to reference an EJB for which `@Named` is assigned, the error message KDJE59316-E is output and the injection operation fails.
- If the `@Produces` annotation and `@Disposes` annotation are assigned with the `@Dependent` scope object, the producer method cannot be referenced in the same class. Also, if you create such an application, recursive injection is performed and `StackOverflowException` might occur.
- The `@PostConstruct` annotation or `@PreDestroy` annotation cannot be used with Beans to which the `@Produces` annotation or `@Disposes` annotation are assigned.
- An interceptor can be used with the CDI applications. However, you cannot use the EJB interceptor and CDI interceptor for the EJB at the same time. If used at the same time, the CDI interceptor is not executed.

- Using the reload functionality with the applications that use the CDI

The CDI applications are reloaded for each application. The CDI applications cannot be reloaded for each WAR and JSP. Also, if `Facelets` is used with the JSF, the CDI applications cannot be reloaded.

- Notes on the application starting performance

If the number of JAR files of `beans.xml` included in the CDI applications increases, the deployment time increases greatly, compared to the increment in the JAR files.

- Notes on using Managed Beans

The Managed Bean instances for which the `javax.enterprise.context.SessionScoped` annotation is specified are not stored in the explicitly-managed heap.

10

Using Bean Validation with Application Server

This chapter provides an overview of the Bean Validation functionality and describes how to implement the functionality.

10.1 Organization of this chapter

This chapter describes the Bean Validation functionality.

The following table describes the organization of this chapter.

Table 10-1: Organization of this chapter (Using Bean Validation with Application Server)

Category	Title	Reference location
Explanation	Overview of Bean Validation	<i>10.2</i>
	Applicable scope of Bean Validation	<i>10.3</i>
Implementation	Bean Validation functionality and Bean Validation operations	<i>10.4</i>
	Procedures for using Bean Validation	<i>10.5</i>
	Using the debug log (check development log)	<i>10.6</i>
Operations	Output of information when the <code>validation.xml</code> contents are invalid	<i>10.7</i>
Notes	Notes on the implementation of Bean Validation	<i>10.8</i>

Note: The functionality-specific explanation is not available for "Settings".

10.2 Overview of Bean Validation

This section provides an overview of Bean Validation.

The Bean Validation function validates whether the value actually input for an application is within the scope of the property value specified with an annotation in the JavaBeans property.

The purpose of Bean Validation is to simplify the validation process of the values input in an application.

You can use Bean Validation on user applications. With the applications using JSF and CDI, you can also use the integration functionality defined in the standard specifications.

For details on JSF and CDI, see *3.2 Overview of JSF and JSTL* and *9.2 Overview of the CDI*, in the *uCosminexus Application Server Web Container Functionality Guide*.

10.3 Applicable scope of Bean Validation

JSF, CDI, and user applications can be integrated with Bean Validation.

You cannot use the reference library and the container extension library.

10.4 Bean Validation functionality and Bean Validation operations

This section describes the Bean Validation functionality and Bean Validation operations.

The following table describes the Bean Validation functionality.

Table 10-2: Bean Validation functionality

No.	Function	Overview
1	Validating the input values	This functionality validates the validation definition specifying the value set up in JavaBean.
	Group management functionality	This functionality groups the validation.
	Message management functionality	This functionality manages the messages returned when the validation result is an error.
	Payload management functionality	This functionality categorizes the validation results. [#]
2	Functionality for creating a custom validator	This functionality creates specific validation processing.
3	Bootstrap functionality	This functionality enables you to change the Bean Validation provider.

#

If Bean Validation is integrated and used with JSF, the payload management functionality cannot be used.

You can use the Bean Validation function by specifying annotations. The following table describes the annotation classes provided by the Bean Validation function, the variable types that can be specified, and the operations when annotations are specified in the variable types that cannot be specified.

Table 10-3: Annotation classes provided by the Bean Validation function and the types of variables

No.	Annotation class	Specifiable variable types	Operations when annotations are specified in the non-specifiable variable types	Remarks
1	Null	Can be specified in all types	--	--
2	NotNull	Can be specified in all types	--	--
3	AssertTrue	<ul style="list-style-type: none"> boolean/ java.lang.Boolean 	javax.validation.UnexpectedTypeException is thrown.	--
4	AssertFalse	<ul style="list-style-type: none"> boolean/ java.lang.Boolean 	javax.validation.UnexpectedTypeException is thrown.	--
5	Min	<ul style="list-style-type: none"> java.math.BigDecimal java.math.BigInteger byte/ java.lang.Byte short/ java.lang.Short int/ java.lang.Integer long/ java.lang.Long 	javax.validation.UnexpectedTypeException is thrown.	--

No.	Annotation class	Specifiable variable types	Operations when annotations are specified in the non-specifiable variable types	Remarks
5	Min	<ul style="list-style-type: none"> • float/ java.lang.Float • double/ java.lang.Double • java.lang.String 	javax.validation.UnexpectedTypeException is thrown.	--
6	Max	<ul style="list-style-type: none"> • java.math.BigDecimal • java.math.BigInteger • byte/ java.lang.Byte • short/ java.lang.Short • int/ java.lang.Integer • long/ java.lang.Long • float/ java.lang.Float • double/ java.lang.Double • java.lang.String 	javax.validation.UnexpectedTypeException is thrown.	--
7	DecimalMin	<ul style="list-style-type: none"> • java.math.BigDecimal • java.math.BigInteger • java.lang.String • byte/ java.lang.Byte • short/ java.lang.Short • int/ java.lang.Integer • long/ java.lang.Long • float/ java.lang.Float • double/ java.lang.Double 	javax.validation.UnexpectedTypeException is thrown.	If a value that cannot be parsed with java.math.BigDecimal is specified for the value attribute, javax.validation.ValidationException is thrown.
8	DecimalMax	<ul style="list-style-type: none"> • java.math.BigDecimal • java.math.BigInteger • java.lang.String • byte/ java.lang.Byte • short/ java.lang.Short • int/ java.lang.Integer 	javax.validation.UnexpectedTypeException is thrown.	If a value that cannot be parsed with java.math.BigDecimal is specified for the value attribute, javax.validation.ValidationException is thrown.

No.	Annotation class	Specifiable variable types	Operations when annotations are specified in the non-specifiable variable types	Remarks
8	DecimalMax	<ul style="list-style-type: none"> • long/ java.lang.Long • float/ java.lang.Float • double/ java.lang.Double 	javax.validation.UnexpectedTypeException is thrown.	If a value that cannot be parsed with java.math.BigDecimal is specified for the value attribute, javax.validation.ValidationException is thrown.
9	Size	<ul style="list-style-type: none"> • java.lang.String • java.util.Collection • java.util.Map • Array 	javax.validation.UnexpectedTypeException is thrown.	<p>If a negative value is specified for the max and min attributes, javax.validation.ValidationException is thrown.</p> <p>If a value greater than max is specified in the value for min, javax.validation.ValidationException is thrown.</p>
10	Digits	<ul style="list-style-type: none"> • java.math.BigDecimal • java.math.BigInteger • java.lang.String • byte/ java.lang.Byte • short/ java.lang.Short • int/ java.lang.Integer • long/ java.lang.Long • float/ java.lang.Float • double/ java.lang.Double 	javax.validation.UnexpectedTypeException is thrown.	If a negative value is specified for the integer and fraction attributes, javax.validation.ValidationException is thrown.
11	Past	<ul style="list-style-type: none"> • java.util.Date • java.util.Calendar 	javax.validation.UnexpectedTypeException is thrown.	--
12	Future	<ul style="list-style-type: none"> • java.util.Date • java.util.Calendar 	javax.validation.UnexpectedTypeException is thrown.	--
13	Pattern	<ul style="list-style-type: none"> • java.lang.String 	javax.validation.UnexpectedTypeException is thrown.	<p>If the value specified for the regexp attribute is an incorrect regular expression, the javax.validation.ValidationException is thrown.</p> <p>The validity of the regular expression depends on the java.util.regex.Pattern specifications.</p>

Legend:

--: Not applicable.

10.5 Procedures for using Bean Validation

This section describes procedures for using Bean Validation.

10.5.1 Procedures for using Bean Validation from JSF

This subsection describes procedures for using Bean Validation from JSF.

(1) Preconditions for use

The following conditions must be satisfied to perform the validation processing using Bean Validation in JSF:

- Create the Web page with Facelets.
- Use the Bean Validation annotation in the ManagedBean, and define the constraints.

(2) Validation operations

The validation operations of the Bean Validation function vary depending on the value of the context parameter provided by JSF. The operations change as described in the following table depending on the value specified for `javax.faces.validator.DISABLE_DEFAULT_BEAN_VALIDATOR` and the presence or absence of the `<f:validateBean>` tag for the ManagedBean variable that defines the annotation for Bean Validation.

Table 10-4: Relationship of the value specified for `javax.faces.validator.DISABLE_DEFAULT_BEAN_VALIDATOR` and the validation processing

Value specified for <code>javax.faces.validator.DISABLE_DEFAULT_BEAN_VALIDATOR</code>	Input value Specification of the <code><f:validateBean></code> tag (when disabled attribute is not specified)	Validation by Bean Validation
true	Yes	Yes
	No	No
false	Yes	Yes
	No	Yes
If value is not specified (default)	Same as when <code>false</code> is specified	Same as when <code>false</code> is specified

For details on how to use the `<f:validateBean>` tag, see the standard specifications for JSF. For details on how to define validation for the ManagedBean variable, see the standard specifications for Bean Validation.

(3) Example of implementation

The following is an example implementation of using Bean Validation from JSF.

The first example shows an implementation of the Facelets page that registers the information for which validation is required.

```
<f:view>
<h:form>
  Enter ID <br/>
  <h:inputText id="IDBox" value="#{personalData.id}" /><br/>
  <h:message for="IDBox"/><br/>
  <br/>
  <f:validateBean disabled="true">
  Enter name (validation processing is not performed)<br/>
  <h:inputText id="NameBox" value="#{personalData.name}" /><br/>
  </f:validateBean >
  <br/>
  Enter age<br/>
  <h:inputText id="AgeBox" value="#{personalData.age}" /><br/>
  <h:message for="AgeBox"/><br/>
  <br/>
</h:form>
</f:view>
```

```

<br/>
...
</h:form>
</f:view>

```

! Important note

The `<inputText>` tag that specifies `<f:validateBean disabled="true">` defines validation for the corresponding `ManagedBean` variable, but the validation processing is not performed.

The next example shows an implementation of the validation definition for the `ManagedBean` variable that stores the data to be validated.

```

@ManagedBean(name="personalData")
@SessionScoped
public class PersonalData
{
    @Size(min=8,max=12, message="Enter a string of 8 to 12 characters.")
    private String id = "";

    @Size(min=1,message="Enter the name.")
    private String name = "";

    @Max(value = 150, message="Check whether the age is entered correctly.")
    @Min(value = 0, message="Enter an age of at least 0 years.")
    private int age = 0;
    ...
    setter/gettter method
    ...
}

```

10.5.2 Procedures for using Bean Validation from CDI

This subsection describes procedures for using Bean Validation from CDI.

(1) Procedure required before using CDI

To perform the validation processing using Bean Validation with CDI:

1. Use the `@Inject` annotation in the user application class to inject `ValidatorFactory`.
Example: `@Inject private ValidatorFactory validatorFactory`
2. To obtain the `Validator` object, invoke the `validatorFactory.getValidator()` of the user application from Bean Validation.
3. Finally, invoke the `validator.validate()` method from the user application, and pass the Bean class to be validated.

(2) Example of implementation

The following is an example of implementation using Bean Validation from CDI.

The first example shows an implementation of the servlet that registers the information for which validation is required.

```

public class EmployeeServBv extends HttpServlet{
    @Inject private ValidatorFactory validatorFactory;
    @Inject BV_CDI bean;

    public void doGet(HttpServletRequest req, HttpServletResponse res) throws
ServletException{
    Validator validator = validatorFactory.getValidator();
    validator.validate(bean);
    }
}

```

In this example, the Bean Validation annotation is applied in the `BV_CDI` bean.

The next example shows an implementation of the validation definition for the Bean that stores the data to be validated.

```
import javax.validation.constraints.NotNull;
public class BV_CDI{
    @NotNull
    private String name;
    public String getName(){
        return name;
    }
    public void setName(String name){
        this.name = name;
    }
}
```

10.6 Using the debug log (check development log)

This section describes the log for debugging Bean Validation (check development log).

With the Bean Validation function, development-related messages are output to the check development log. The messages that must be checked for debugging Bean Validation used with applications can be determined from the class names output in the messages. The following table describes the class names in which the messages are output.

Table 10-5: Class names output in the check development log for Bean Validation

Component name	Output class name	Output contents
Bean Validation	com.hitachi.software.beanvalidation~	<ul style="list-style-type: none"> • Messages related to errors • Messages related to the operation status of the function • Messages related to the internal status

Note that the check development log is not output by default. Change the settings as and when required.

For details on the settings for the output of the check development log, and information such as the output destination, output format, and log level of the check development log, see *24.4 Messages output in the check development log* in the manual *uCosminexus Application Server Messages*.

10.7 Output of information when the validation.xml contents are invalid

This section describes the messages that are output when the `validation.xml` contents are invalid in a system integrated with Bean Validation.

The following table lists the output message ID and contents, and the output destination and action for each application integrated with the Bean Validation function.

Table 10-6: Output messages, output destinations, and actions when the validation.xml contents are invalid

Applications integrated with Bean Validation	Output message ID and contents	Output destination and action
JSF applications	The Web container outputs KDJE39056-E. The output message contains the resource name and exception name.	Output destination: web_servlet log Action: Check the <code>validation.xml</code> contents and correct the problem.
CDI applications	KDJE59312-E is output. The output message contains the exception details.	Output destination: cjexception log Action: Check the <code>validation.xml</code> contents and correct the problem.
User applications (Web applications)	The Web container outputs KDJE39045-E. In the J2EE server mode, the output message contains the J2EE application name and the context root name. In the servlet engine mode, an empty string and the context root name is output.	Output destination: web_servlet log Action: Check the <code>validation.xml</code> contents and correct the problem.
User applications (EJB applications)	--	Output destination: cjexception log Action: Check the <code>validation.xml</code> contents and correct the problem.

Legend:

--: Not applicable.

10.8 Notes on the implementation of Bean Validation

This section describes the notes on the implementation of Bean Validation.

Notes on the Bean Validation 1.0 specifications

With the Bean Validation 1.0 specifications, only one `validation.xml` can be allocated to the class path. Do not allocate multiple `validation.xml` to the class path of a J2EE application.

Notes related to the default Message Interpolator messages

The default Message Interpolator messages differ from the content mentioned in the standard specifications. The following table lists the default values of the Message Interpolator messages for Application Server.

Table 10-7: Default values of the Message Interpolator messages for Application Server

Message property key	Default value
<code>javax.validation.constraints.AssertFalse.message</code>	must be false
<code>javax.validation.constraints.AssertTrue.message</code>	must be true
<code>javax.validation.constraints.Digits.message</code>	numeric value out of bounds (<{integer} digits>.<{fraction} digits> expected)
<code>javax.validation.constraints.Future.message</code>	must be in the future
<code>javax.validation.constraints.Max.message</code>	must be less than or equal to {value}
<code>javax.validation.constraints.Min.message</code>	must be greater than or equal to {value}
<code>javax.validation.constraints.NotNull.message</code>	may not be null
<code>javax.validation.constraints.Null.message</code>	must be null
<code>javax.validation.constraints.Past.message</code>	must be in the past
<code>javax.validation.constraints.Pattern.message</code>	must match "{regexp}"
<code>javax.validation.constraints.Size.message</code>	size must be between {min} and {max}

11

Managing Application Attributes

This chapter describes the management of the application attributes. When you create an application, you must define the elements configuring the applications, such as EJB-JAR attributes, WAR attributes, and resources.

With Application Server, you define the information required for using the application in the respective property files. Also, by defining the Application Server-specific information in `cosminexus.xml` and including the information in the application, you can manage the default DD definition and the Application Server-specific definition separately. This chapter describes the property files required for setting up the applications and the operations for the applications using `cosminexus.xml`.

11.1 Organization of this chapter

The following table describes the contents explained in the management of application attributes and the reference locations.

Table 11-1: Contents described in the management of application attributes and the reference locations

Functionality	Reference location
Managing attributes	<i>11.2</i>
Applications containing <code>cosminexus.xml</code>	<i>11.3</i>
Omitting the DD	<i>11.4</i>

11.2 Managing attributes

With Application Server, you can use the file defining the DD and the Cosminexus Application Server-specific information (`cosminexus.xml`) to define an application.

The following table describes the range of the DD supported in Application Server.

Table 11-2: Range of the DD supported in Application Server

DD types	Version	Support
application.xml	1.2	C#
	1.3	C#
	1.4	Y
	5.0	Y
	6.0	Y
	No DD (5.0)	Y
	No DD (6.0)	Y
ejb-jar.xml	1.1	C
	2.0	Y
	2.1	Y
	3.0	Y
	3.1	Y
	No DD (3.0)	Y
	No DD (3.1)	Y
web.xml	2.2	C
	2.3	Y
	2.4	Y
	2.5	Y
	3.0	Y
	No DD (2.5)	Y
	No DD (3.0)	Y
ra.xml	1.0	Y
	1.5	Y

Legend:

Y: Supported.

C: Supported, but the DD version is rewritten during import.

#

If the `application.xml` version is 1.2 or 1.3, the version during import is specified as 1.4, even if `<context-root>` is not unique in the EAR file.

! Important note

Elements that can be specified in `ejb-jar.xml` version 3.0

You can specify EJB-JAR `<display-name>`, the interceptor-related elements, and the application exception-related elements. If you specify other elements and attributes, the value will be ignored.

You can typically specify the following elements:

- `/ejb-jar/display-name`
- `/ejb-jar/assembly-descriptor/interceptor-binding`, and the elements under this element
- `/ejb-jar/assembly-descriptor/application-exception`, and the elements under this element

If you specify multiple `interceptor-binding` elements with * (asterisk) in the `ejb-name` element, only the contents coded at the top are used. The second and subsequent contents are ignored.

If you specify multiple `<interceptor-binding>` tags with matching `<ejb-name>`, `<named-method>`, and the elements under them, only the contents coded at the top are used. The second and subsequent contents are ignored.

Elements that can be specified in `ejb-jar.xml` version 3.1

With `ejb-jar.xml` version 3.1, you can specify the following elements in addition to the elements that can be specified with `ejb-jar.xml` version 3.0:

- `/ejb-jar/module-name`

With Application Server, the DD and `cosminexus.xml` can be managed separately and included in the J2EE applications. By including `cosminexus.xml` in a J2EE application, the property file settings need not be specified after the application is imported. Therefore, after the applications containing `cosminexus.xml` are imported, the applications can be started and used as are.

Also, with Application Server, you can omit the DD (`application.xml`, `ejb-jar.xml`, and `web.xml`) that defines the application attributes.

The method of creating and operating the applications containing `cosminexus.xml` is described in 11.3 and later sections. Also, the omission of the DD is described in 11.4 and later sections.

11.3 Applications containing cosminexus.xml

This section gives an overview of the applications containing `cosminexus.xml`.

The following table describes the organization of this section.

Table 11-3: Organization of this section (Applications containing `cosminexus.xml`)

Category	Title	Reference location
Explanation	<code>cosminexus.xml</code>	11.3.1
	Advantages of using the applications containing <code>cosminexus.xml</code>	11.3.2
	Creating the applications containing <code>cosminexus.xml</code>	11.3.3
Implementation	Creating <code>cosminexus.xml</code>	11.3.4
	Example of creating <code>cosminexus.xml</code>	11.3.5
Operations	Operations of the applications containing <code>cosminexus.xml</code>	11.3.6
	Procedure of migrating from the applications not containing <code>cosminexus.xml</code>	11.3.7

Note: The functionality-specific explanation is not available for "Settings" and "Notes".

11.3.1 `cosminexus.xml`

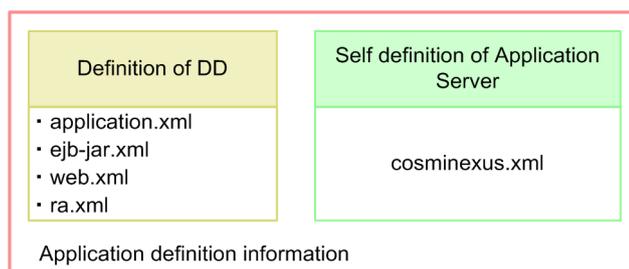
The `cosminexus.xml` file is a property file describing the Cosminexus Application Server-specific definition information related to the applications. You can create `cosminexus.xml` for each application.

By including `cosminexus.xml` in the archive-format applications and importing or re-deploying the applications, and by including `cosminexus.xml` in the exploded archive-format applications and starting the applications, you can execute the applications with the Cosminexus Application Server-specific definition information specified.

For the applications containing `cosminexus.xml`, the DD information and the Cosminexus Application Server-specific information are managed separately. Therefore, when you edit the Cosminexus Application Server-specific information, you need not edit the DD information. With the applications containing `cosminexus.xml`, the command-based application attribute settings need not be specified on the J2EE server.

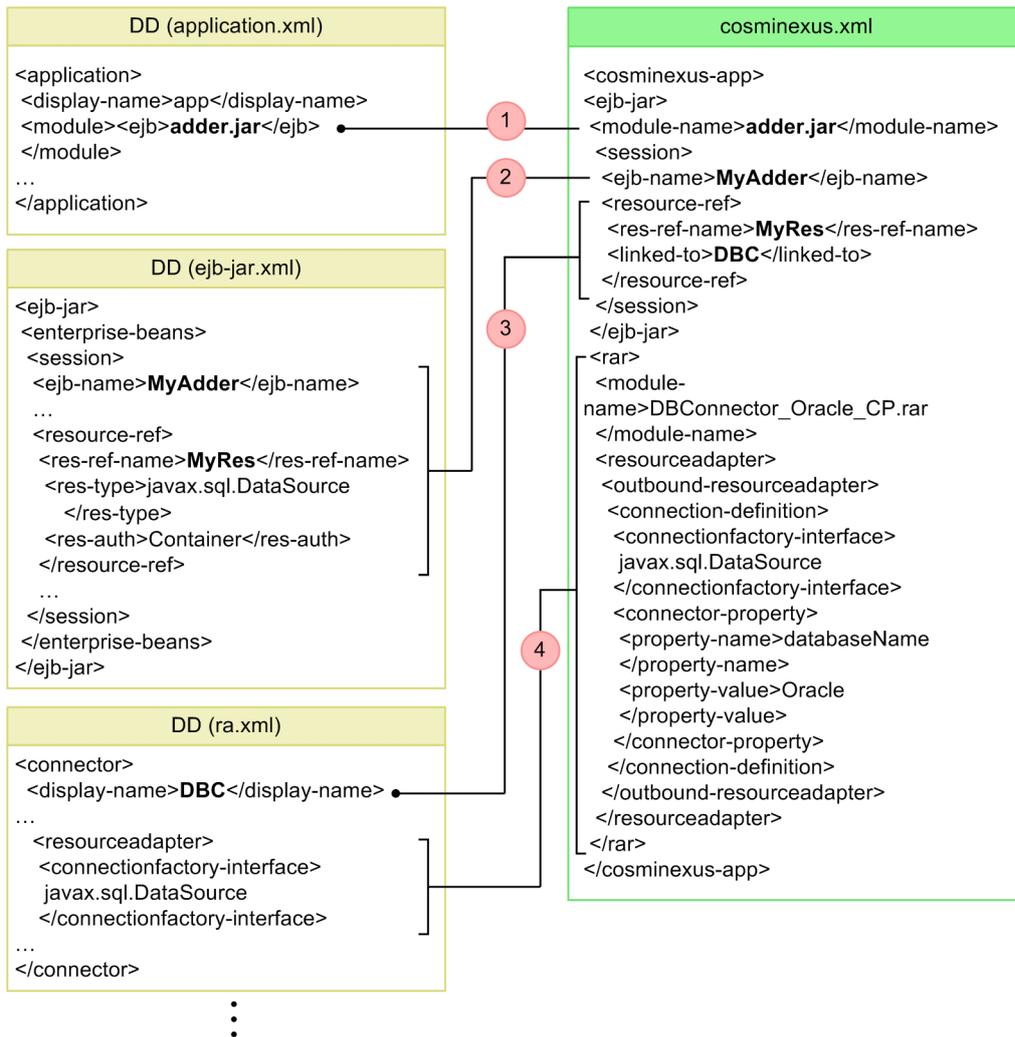
The following figure shows the concept of the applications containing `cosminexus.xml`.

Figure 11-1: Conceptual diagram of the applications containing `cosminexus.xml`



The following figure shows an example of associating the `cosminexus.xml` and DD definitions.

Figure 11-2: Example of associating the cosminexus.xml and DD definitions



The following is a description of the figure. In this example, the resource references from the EJB-JAR are resolved and the resource properties are set up using `cosminexus.xml`. Note that the numbers in the description correspond to the numbers in the figure.

1. Identify the target EJB-JAR with `<module-name>`. In this example, `adder.jar` is set as the target.
2. Associate the DD (`ejb-jar.xml`) and `cosminexus.xml` of `adder.jar`. In this example, the target Session Bean (`MyAdder`) is specified in `<ejb-name>`.
3. Resolve the resource references of the resource (`MyRef`) being referenced with the Session Bean (`MyAdder`). Specify the resource display name (`DBC`) in `<linked-to>` as the reference location for the name specified in `<res-ref-name>`.
4. Specify the properties for the resource (`DBC`). Note that the properties of the resource adapters included in the J2EE application can be specified in `cosminexus.xml`.

By including and importing this `cosminexus.xml` in the J2EE application, you need not resolve the resource references and set the resource adapter properties on the J2EE server.

11.3.2 Advantages of using the applications containing `cosminexus.xml`

This subsection describes the advantages of using the applications containing `cosminexus.xml`.

(1) Definition information can be minimized

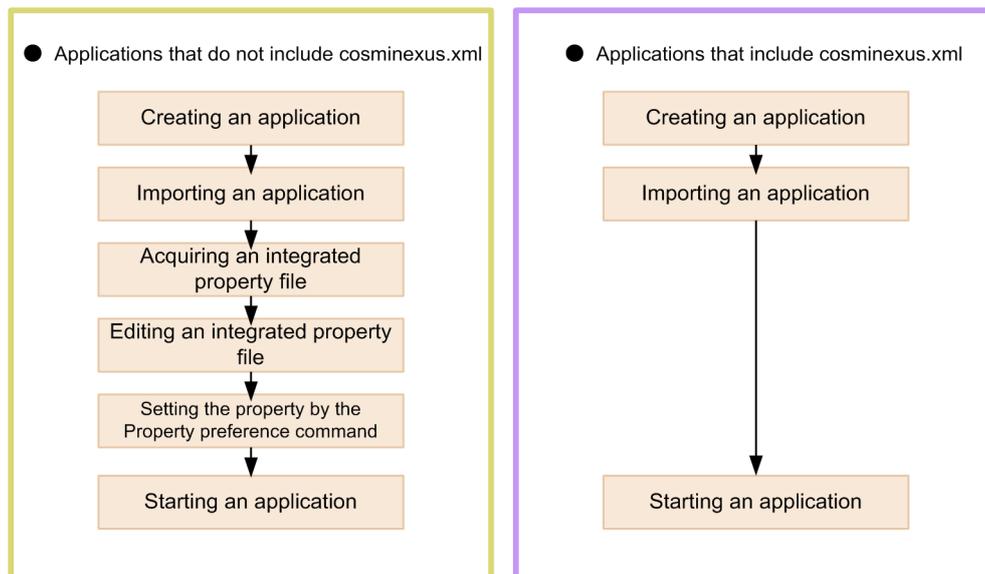
Of the Cosminexus Application Server-specific definition information, only the information that the user wants to customize is coded in the applications containing `cosminexus.xml`. If the information is omitted, the default value is used.

(2) Procedure from importing to starting the application can be simplified

After you import the application, you can execute the application without performing operations using the attribute setting commands of the application (`cjgetappprop` command and `cjsetappprop` command).

The following figure shows the flow of processes from importing to starting the applications containing `cosminexus.xml` and applications not containing `cosminexus.xml`.

Figure 11-3: Flow of processes from importing to starting the applications containing `cosminexus.xml` and applications not containing `cosminexus.xml`



Legend:  : Flow of the process

(3) Property file need not be changed when the DD is changed

For the applications not containing `cosminexus.xml`, when you change the default DD and replace the application, you must re-specify the Cosminexus Application Server-specific definitions by using the server management commands. However, for the applications containing `cosminexus.xml`, the Cosminexus Application Server-specific definitions are managed in a file separate from the DD, therefore, you need not re-specify the settings in the execution environment by using the server management commands.

However, you must change `cosminexus.xml` in the following cases:

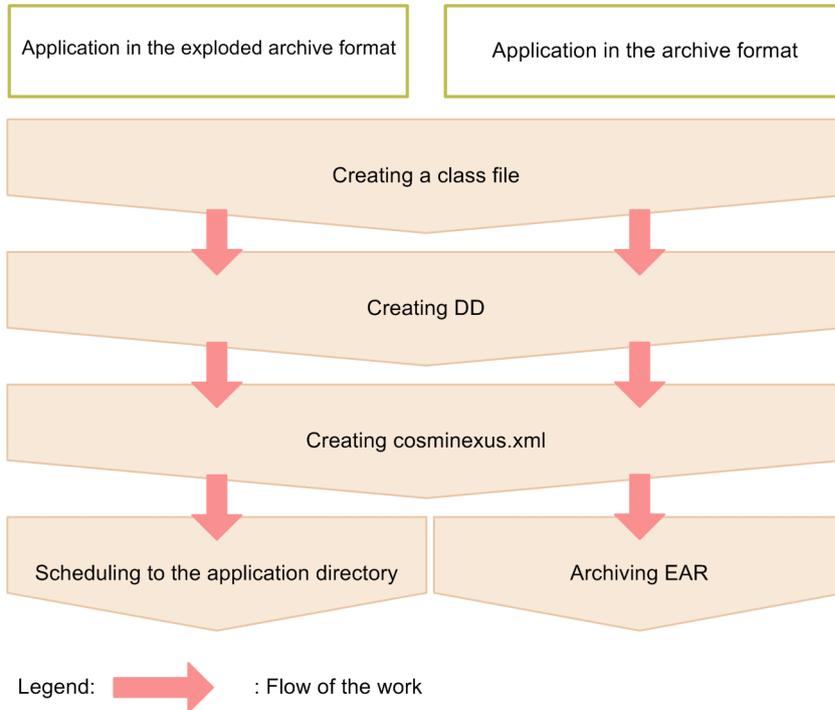
- When you change the information (such as the module name of the J2EE resource) that associates the Cosminexus Application Server-specific definition information with the definition information of the default DD and annotations
- When you add the definition information (such as resources) that requires resolution of links for starting the application

For details on the definition information that associates the definition information coded in `cosminexus.xml` and the definition information in the default DD and annotations, and the definition information that requires link resolution, see 2. *Cosminexus Application Property File (cosminexus.xml)* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

11.3.3 Creating the applications containing cosminexus.xml

This subsection describes how to create the applications containing `cosminexus.xml`. To create the applications containing `cosminexus.xml`, you must first create the class files, DD, and `cosminexus.xml`. The following figure shows the flow of operations for creating the applications containing `cosminexus.xml`.

Figure 11-4: Flow of operations for creating the applications containing `cosminexus.xml`



Note that for a WAR application that is imported by specifying a WAR file or WAR directory, Figure 11-4 is as follows. For details on the WAR applications, see *13.9 WAR applications*.

In the exploded archive format

The application is allocated in the WAR directory.

In the archive format

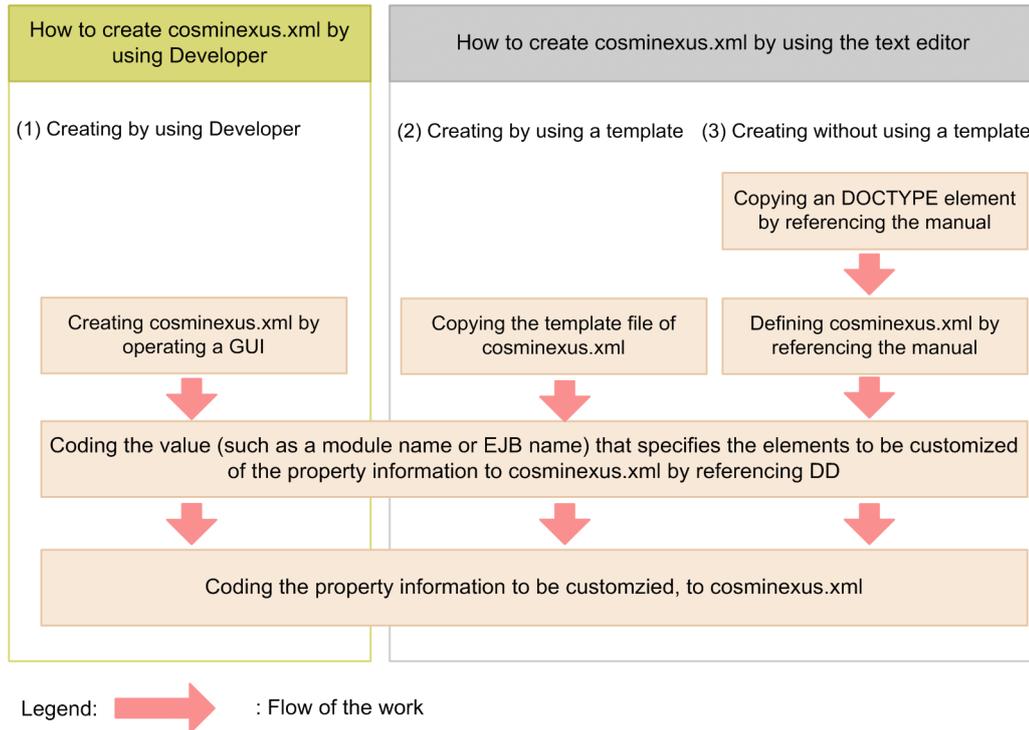
The application creates a WAR archive.

For details on creating the class files, DD, and applications containing `cosminexus.xml`, see the *uCosminexus Application Server Application Development Guide*. How to create `cosminexus.xml` is described in *11.3.4* and later sections.

11.3.4 Creating `cosminexus.xml`

To create the applications containing `cosminexus.xml`, you must first include the created `cosminexus.xml` in the application. The following figure shows how to create `cosminexus.xml`.

Figure 11-5: How to create cosminexus.xml



The methods of creating `cosminexus.xml` include the method of using WTP of Developer and the method of using the text editor. Each of these methods is described as per points (1) through (3) in the figure.

(1) Method of creating cosminexus.xml by using Developer

With Application Server, you can create `cosminexus.xml` by using WTP of Developer. For details on how to create `cosminexus.xml` by using WTP of Developer, see 5.3.1 *Creating cosminexus.xml* in the *uCosminexus Application Server Application Development Guide*.

(2) Method of creating cosminexus.xml by using a template

If you use a text editor to create `cosminexus.xml`, you can easily create `cosminexus.xml` if you use the template file.

The templates are stored in the following locations.

Template storage location

- In Windows
`Cosminexus-installation-directory\CC\admin\templates\cosminexus.xml`
- In UNIX
`/opt/Cosminexus/CC/admin/templates/cosminexus.xml`

You copy the template and store in the archive file or expansion destination directory.

The storage destination of `cosminexus.xml` is as follows:

To import an EAR file

- Storage destination in the EAR (for the archive-format applications)
`EAR-root/META-INF/cosminexus.xml`
- Storage destination in the application directory (for the exploded archive-format applications)
For details on the storage destination of `cosminexus.xml` in the exploded archive-format applications, see 13.4.2 *Configuration of the application directory*.

To import a WAR file

Specify the storage destination of `cosminexus.xml` with the `cjimportwar` command. For details on the `cjimportwar` command, see *cjimportwar (Importing WAR applications)* in the *uCosminexus Application Server Command Reference Guide*.

! Important note

The template contains the URI `file:///Cosminexus-installation-directory/CC/admin/dtds/cosminexus_8_0.dtd` that presumes that the Cosminexus installation destination is re-written as the storage destination of the schema definition file specified in the `DOCTYPE` element. The J2EE server does not reference this URI, so the URI need not be rewritten. However, if you use the URI in an editor such as an XML editor, rewrite this URI according to the environment to be used.

(3) Method of creating `cosminexus.xml` without using a template

To create `cosminexus.xml` without using a template, you use a text editor to define the Cosminexus Application Server-specific information based on the manual. For details on the `DOCTYPE` element required when you use a text editor to create `cosminexus.xml` and the contents defined in `cosminexus.xml`, see 2. *Cosminexus Application Property File (cosminexus.xml)* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

11.3.5 Example of creating `cosminexus.xml`

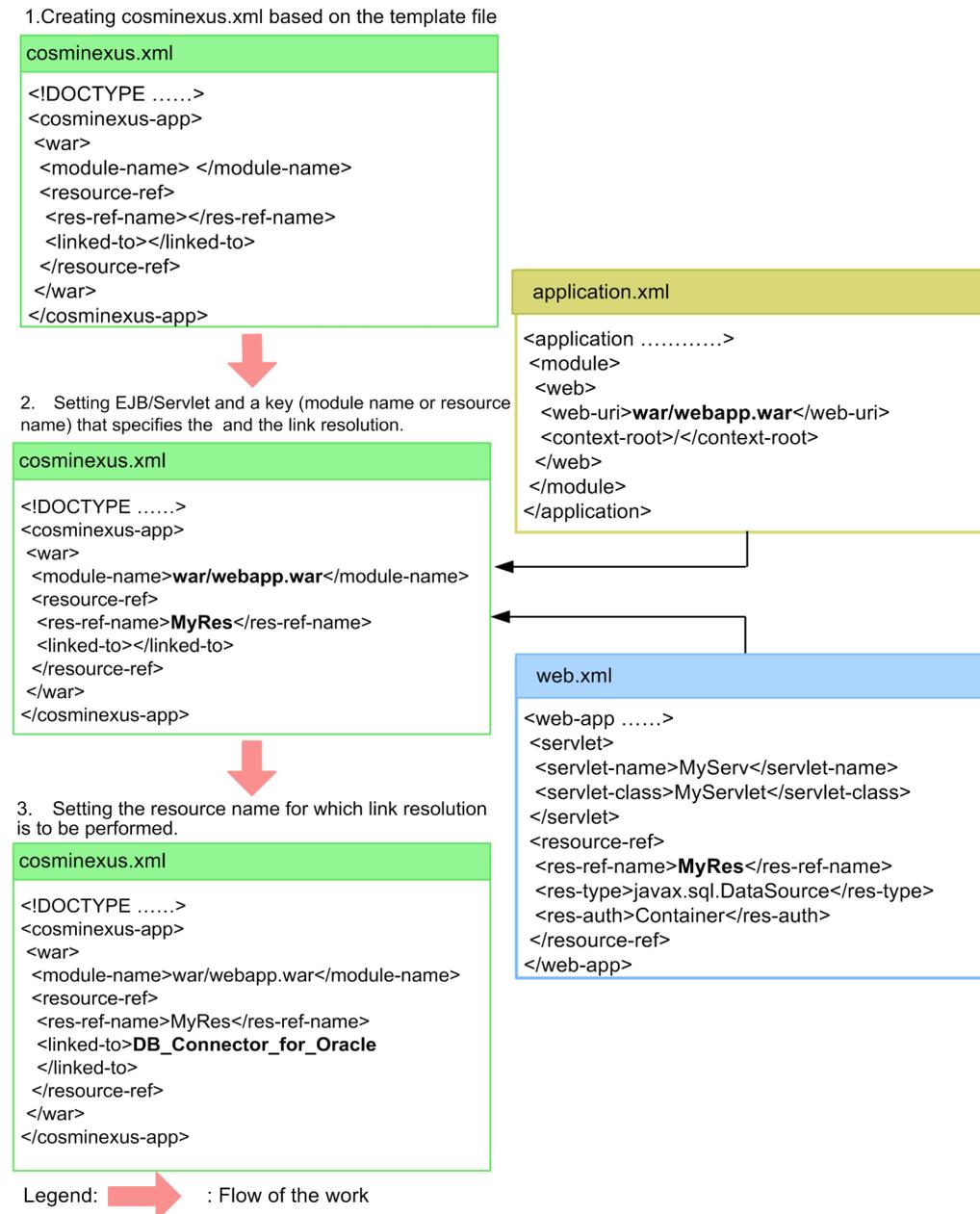
This subsection gives an example of creating `cosminexus.xml` using a text editor. In this example, `cosminexus.xml` is created on the basis of the following contents defined in the DD:

Contents defined in the default DD

- Web application module name: `war/webapp.war`
- Servlet name: `MyServ`
- Name of the resource referenced from Servlet (`MyServ`): `DB_Connector_for_Oracle`

The following figure shows an example of creating `cosminexus.xml`.

Figure 11-6: Example of creating cosminexus.xml



The following is a description of points 1 through 3 in the figure:

1. Create `cosminexus.xml` on the basis of the template file.
You copy the template file and store in the EAR or application directory. You edit the copied template file using a text editor.
2. Set up the key (module name and resource name) for identifying the EJB/Servlet and link resolution.
You set up the key associated with the information defined in the DD. In this example, the name of the Web application module and the name of the referenced resource are set up.
3. Set up the name of the resource for which the link is to be resolved.
You set up the resource that resolves the link of the key set up in step 2.

For details on the contents defined in `cosminexus.xml`, see *2.1 Contents specified in the Cosminexus application property file (cosminexus.xml)* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

11.3.6 Operations of the applications containing `cosminexus.xml`

This subsection describes the operations of the applications containing `cosminexus.xml`.

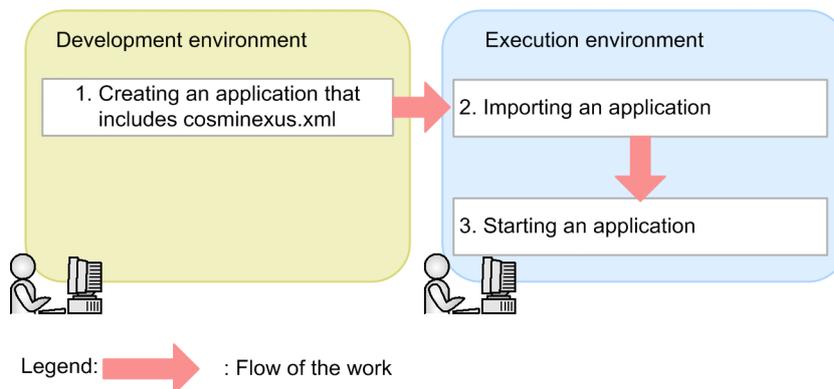
(1) Flow of creating a new application containing `cosminexus.xml`

This section describes the flow of operations for creating a new application containing `cosminexus.xml`.

You create the application containing `cosminexus.xml` in the development environment. Import the created application into the execution environment and then start the application.

The following figure shows the flow of operations from the creation of the application containing `cosminexus.xml` in the development environment to the starting of the application in the execution environment.

Figure 11-7: Flow of operations from the creation of the application containing `cosminexus.xml` to the starting of the application in the execution environment



The following is a description of points 1 through 3 in the figure:

1. Creating an application containing `cosminexus.xml`
 Create the class files, default DD, and `cosminexus.xml` and archive them according to the application format or allocate them in the application directory. For details on creating the class files, DD, and applications containing `cosminexus.xml`, see the *uCosminexus Application Server Application Development Guide*. Also, for details on creating `cosminexus.xml` using a text editor, see *11.3.4 Creating cosminexus.xml*.
2. Importing the application
 You import the application created in the development environment into the execution environment. For details on importing applications, see *(2) Importing applications*.
3. Starting the application
 You start the imported application. For details on how to start an application, see *(3) Starting applications*.

(2) Importing applications

This section describes the importing of applications containing `cosminexus.xml`.

For the applications containing `cosminexus.xml`, the `cosminexus.xml` information is read when the application is imported. To import the J2EE application, execute the `cjimportapp` command. For details on the `cjimportapp` command, see *cjimportapp (Importing J2EE applications)* in the *uCosminexus Application Server Command Reference Guide*. For the notes on importing the J2EE applications, see *8.1 Importing J2EE applications* in the *uCosminexus Application Server Application Setup Guide*.

Execute the `cjimportwar` command to import a WAR application. For details on the `cjimportwar` command, see *cjimportwar (Importing WAR applications)* in the *uCosminexus Application Server Command Reference Guide*. For details on the WAR applications, see *13.9 WAR applications*.

(3) Starting applications

This section describes the starting of the applications containing `cosminexus.xml`.

The same procedure is used for starting an application containing `cosminexus.xml` that is used for starting an application not containing `cosminexus.xml`. To start the application, you execute the `cjstartapp` command. For details on the `cjstartapp` command, see *cjstartapp (Starting the J2EE applications)* in the *uCosminexus Application Server Command Reference Guide*. For the notes on starting the J2EE applications, see *10.2.1 Starting the J2EE applications* in the *uCosminexus Application Server Application Setup Guide*.

The following is a description of the operations of an application when an application containing `cosminexus.xml` is started in the execution environment, stopped once, and then restarted:

The operations will be described separately for the archive-format applications, exploded archive-format applications, and applications not containing `cosminexus.xml` as the formats when the application is restarted.

- For the archive-format applications
When `cosminexus.xml` is included in the archive-format applications, the `cosminexus.xml` information included in the application is read. The definition information changed using the server management commands before the application is stopped is not applied to the archive-format applications.
- For the exploded archive-format
When `cosminexus.xml` is included in the exploded archive-format applications, the application-related information specific to Cosminexus Application Server and changed in the execution environment is overwritten when the application starts. In other words, for the exploded archive-format applications, the changes in the definition information using the server management commands are applied in the file.
- For the applications not containing `cosminexus.xml`
When `cosminexus.xml` is not included in an application, the application is started with the Cosminexus Application Server-specific definition content used when the application was started previously.

(4) Procedure of changing the definition information after the application starts

This section describes the procedures for changing the definition information of the application after the application imported from the development environment is started in the execution environment.

When you change the definitions for the applications containing `cosminexus.xml`, the procedure differs depending on whether the application format is the archive format or the exploded archive format.

The procedure for changing the contents defined for each application format is as follows:

Reference note

As in the case of the applications not containing `cosminexus.xml`, you can also use the server management commands (`cjgetappprop` command and `cjsetappprop` command) to change the contents defined for an application.

! Important note

The notes on changing the information defined in the applications containing `cosminexus.xml` are as follows:

- If you want to return all the Cosminexus Application Server-specific definition information to the default values, you delete all the elements except the `<cosminexus-app>` element from the `cosminexus.xml` file.
- The following paragraphs describe the notes on changing the application definitions by using the server management commands, for each application format:

For the archive format

When you change the definition information using the redeploy functionality, the application runtime information of Cosminexus Application Server is returned to the default values once, and the information defined in the replaced `cosminexus.xml` is read. Therefore, all the definition information set up with the server management commands is lost.

! Important note

For a WAR application, you cannot change the definition information with the redeploy functionality. Use the server management commands (`cjgetappprop` and `cjsetappprop` commands) to change the information.

For the exploded archive format

The contents of the application attributes changed by using the server management commands are applied as are to `cosminexus.xml` in the application directory.

Note that the `cosminexus.xml` definition information is not read with the reload functionality. If you want to apply the changed definition information, you must restart the application. For details on changing the definition information by using HITACHI Application Integrated Property File and server management commands, see 9. *Setting up the J2EE Application Properties* in the *uCosminexus Application Server Application Setup Guide*.

(a) For the archive format

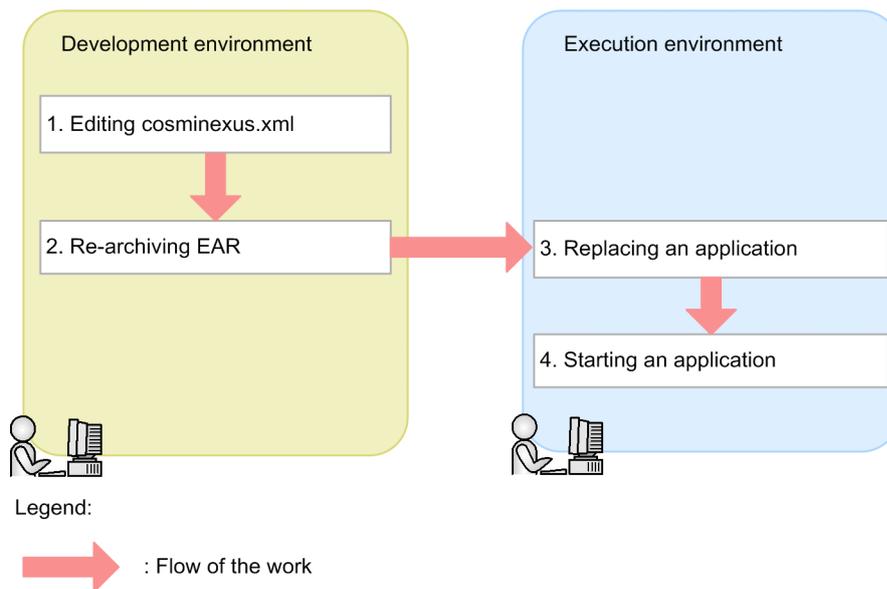
This section describes the procedure of changing the definition information when the application containing `cosminexus.xml` is an archive-format application. Note that the definition information of the applications containing `cosminexus.xml` can be changed not only with the procedure described here, but also with the procedure of using the server management commands and property files.

When you want to change the definition content after the application containing `cosminexus.xml` is started in the execution environment, change the definition content of the application in the development environment. After changing and archiving the definition content in the development environment, import the application into the execution environment again and then replace the application.

You can also export a running J2EE application by setting the application in the ZIP format with runtime information. The procedure of operating applications in the ZIP format with runtime information is the same for the applications containing `cosminexus.xml` and the applications not containing `cosminexus.xml`. For details on exporting applications, see (6) *Exporting applications*.

The following figure shows the procedure of changing the content defined in the applications containing `cosminexus.xml`.

Figure 11-8: Procedure of changing the content defined in the applications containing `cosminexus.xml` (In the archive format)



The following subsection describes the points 1 through 4 of the above figure:

1. Editing `cosminexus.xml`

To change the definition information after the application containing `cosminexus.xml` is imported into the execution environment, you edit `cosminexus.xml` in the development environment. For details on editing `cosminexus.xml`, see 5.3.2 *How to operate the cosminexus.xml editor* in the *uCosminexus Application Server Application Development Guide*.

2. Re-archiving the EAR file

You include the edited `cosminexus.xml` into the application and archive the file.

3. Replacing the application

You replace the application being used in the execution environment with the application that is re-created in the development environment. For details on replacing an application, see (5) *Replacing applications*.

4. Starting the application

You start the application. For details on how to start an application, see (3) *Starting applications*.

Tip

When you use the server management commands to change the `cosminexus.xml` definition information and when you replace an application, the `cosminexus.xml` definition information included in the replaced J2EE application is overwritten. To change the `cosminexus.xml` definition information, you edit `cosminexus.xml` in the development environment, and then redeploy or re-import the application.

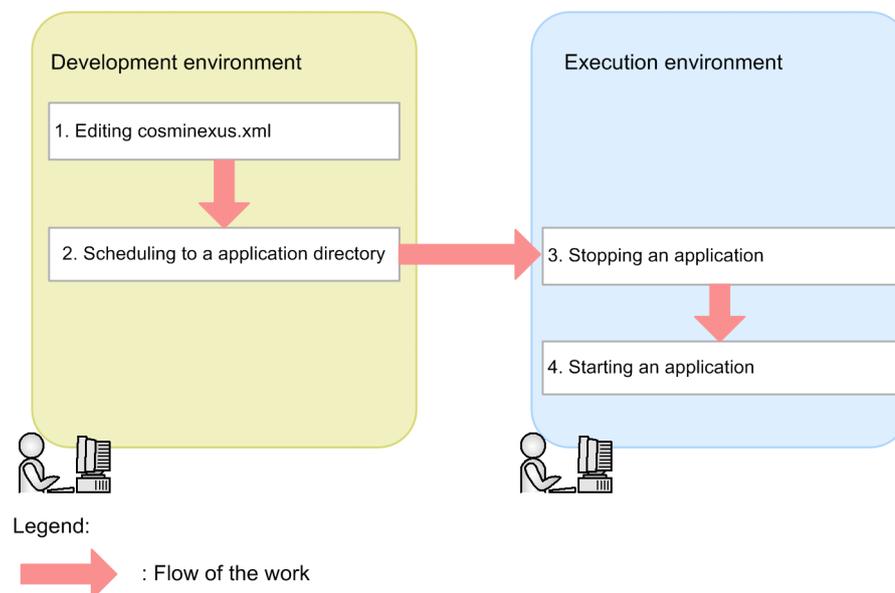
For details on how to change the definition information by using the server management commands, see 9. *Setting up the J2EE Application Properties* in the *uCosminexus Application Server Application Setup Guide*.

(b) For the exploded archive format

This subsection describes the procedure of changing the definition information when the application containing `cosminexus.xml` is an exploded archive-format application.

The following figure shows the procedure of changing the content defined in the applications containing `cosminexus.xml`.

Figure 11-9: Procedure of changing the content defined in the applications containing `cosminexus.xml` (In the exploded archive format)



The following subsection describes the points 1 through 4 of the above figure:

1. Editing `cosminexus.xml`

To change the definition information after the application containing `cosminexus.xml` is imported into the execution environment, you edit `cosminexus.xml` in the development environment. For details on editing `cosminexus.xml`, see 5.3.2 *How to operate the cosminexus.xml editor* in the *uCosminexus Application Server Application Development Guide*.

2. Allocating to the application directory

You allocate the edited `cosminexus.xml` to the application directory. For details on allocation to the application directory, see 1.4.1 *Exploded archive-format J2EE applications* in the *uCosminexus Application Server Application Development Guide*.

3. Stopping the application

You stop the application being used in the execution environment. For details on stopping an application, see 10.2.2 *Stopping the J2EE applications* in the *uCosminexus Application Server Application Setup Guide*.

4. Starting the application

You start the imported application. For details on starting applications, see (3) *Starting applications*.

(5) Replacing applications

This subsection describes the replacement of the applications containing `cosminexus.xml`.

To replace the archive-format applications, you use the redeploy functionality. When you replace an application using the redeploy functionality, the `cosminexus.xml` information included in the replaced application is read again.

You change `cosminexus.xml` in the development environment, re-archive in the EAR file, and then redeploy to change the Cosminexus Application Server-specific definition information.

! Important note

For a WAR application, you cannot change the definition information by replacing applications. Use the server management commands (`cjgetappprop` and `cjsetappprop` commands) to change the information.

Note that if the application to be replaced does not contain `cosminexus.xml`, the definition information of `cosminexus.xml` before replacement is carried over. #

To use the redeploy functionality, you use the `cjreplaceapp` command. For details on the `cjreplaceapp` command, see *cjreplaceapp (Replacing applications)* in the *uCosminexus Application Server Command Reference Guide*. For details on the notes on the redeploy functionality, see *10.5.1 Archive-format applications* in the *uCosminexus Application Server Application Setup Guide*.

#: Regardless of whether `cosminexus.xml` exists in the application to be replaced, the DD cannot be replaced.

! Important note

The notes on replacing the archive-format applications are as follows:

- For the applications containing `cosminexus.xml`, if you use the server management commands to change the Cosminexus Application Server-specific definition information, and then execute the redeploy functionality, the Cosminexus Application Server-specific information changed with the server management commands is lost.
- To return all the Cosminexus Application Server-specific information to the default values, delete all the elements except `<cosminexus-app>` from `cosminexus.xml`, and then you replace the applications containing `cosminexus.xml`.

(6) Exporting applications

This subsection describes the exporting of the applications containing `cosminexus.xml`.

To export an application, you use the `cjexportapp` command. For details on the `cjexportapp` command, see *cjexportapp (Exporting J2EE applications)* in the *uCosminexus Application Server Command Reference Guide*. For details on the notes on the exporting of applications, see *8.2 Exporting J2EE applications* in the *uCosminexus Application Server Application Setup Guide*.

! Important note

You cannot export an application for a WAR application.

When the applications containing `cosminexus.xml` are exported in the archive format, `cosminexus.xml` is included in the exported EAR file. The following table describes the combination of the applications to be exported and the exported EAR files.

Table 11-4: Combination of the applications to be exported and the exported EAR files

Formats of the applications to be exported	Formats of the exported EAR files	
	When the <code>-normal</code> option is specified in the <code>cjexportapp</code> command	When the <code>-raw</code> option is specified in the <code>cjexportapp</code> command
Applications containing <code>cosminexus.xml</code>	ZIP format with runtime information containing <code>cosminexus.xml</code>	EAR file containing <code>cosminexus.xml</code>
Applications not containing <code>cosminexus.xml</code>	ZIP format with runtime information not containing <code>cosminexus.xml</code>	EAR file not containing <code>cosminexus.xml</code>

! Important note

The notes on `cosminexus.xml` included in the exported applications are as follows:

- The comments are not stored.
- The character encoding is UTF-8.
- If you export a J2EE application containing `cosminexus.xml` in which the DTD version mentioned in `DOCTYPE` is an old version, the DTD version is changed to the latest version supported with Application Server.
- Only `cosminexus.xml` that was finally successfully read by the J2EE server is included in the exported EAR/ZIP. After the J2EE server reads `cosminexus.xml`, even if `cosminexus.xml` in the application directory is changed, the changed definition information is not included in the exported EAR/ZIP. For example, after importing the exploded archive-format application, if `cosminexus.xml` is re-written in the application directory and the application is exported in the EAR format without being started, `cosminexus.xml` used for importing is included in the EAR file. `cosminexus.xml` in the application directory that was updated after being imported is not included.

(7) Deleting J2EE resources from applications

When you delete a J2EE resource from the application, even if `cosminexus.xml` contains the information of the J2EE resource to be deleted, that information is not deleted.

If `cosminexus.xml` contains definition information for customizing an element that does not exist in the application, a warning message is issued during the execution of the server management command that reads `cosminexus.xml`.

11.3.7 Procedure of migrating from the applications not containing `cosminexus.xml`

This subsection describes the procedure for migrating from applications not containing `cosminexus.xml` to applications containing `cosminexus.xml`.

To migrate the applications:

1. Obtain HITACHI Application Integrated Property File from the application used before migration.
You use the server management command (`cjgetappprop`) to obtain the property file.
2. Create `cosminexus.xml`.
For the correspondence between `cosminexus.xml` and HITACHI Application Integrated Property File, see *2.2.1 Details of HITACHI Application Integrated Property* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.
3. Include `cosminexus.xml` created in step 2 in the EAR (for the archive format) or application directory (for the exploded archive format).
4. Replace the application.
5. Verify whether the definition information for the correct application could be migrated.
To verify, you obtain HITACHI Application Integrated Property File from the replaced application and compare the property file with the file obtained in step 1.

11.4 Omitting the DD

With Application Server, you can omit `application.xml`, `ejb-jar.xml`, and `web.xml`. However, depending on the combination of the module and DD versions, you might not be able to omit `application.xml`, `ejb-jar.xml`, or `web.xml`.

This section describes the combination of the omitted DD values in the applications running on Application Server and the operations when the DD is omitted.

The following table describes the organization of this section.

Table 11-5: Organization of this section (Applications containing `cosminexus.xml`)

Category	Title	Reference location
Explanation	Configuration of applications that can be executed with Application Server	11.4.1
	Differences in functionality depending on the presence of <code>application.xml</code>	11.4.2
	Rules for determining the modules when <code>application.xml</code> exists	11.4.3
	Rules for determining the modules when <code>application.xml</code> does not exist	11.4.4
	Operations for Web applications in which <code>web.xml</code> is omitted	11.4.5
	Display name specified when the DD is omitted	11.4.6
	Operations for creating <code>application.xml</code> when <code>application.xml</code> is omitted	11.4.7
Notes	Notes on adding resources in the J2EE applications when <code>application.xml</code> is omitted	11.4.8

Note: The functionality-specific explanation is not available for "Settings", "Implementation" and "Operations".

11.4.1 Configuration of applications that can be executed with Application Server

You can omit the following DDs with Application Server:

- `application.xml` for Java EE 5 and Java EE 6
- `ejb-jar.xml` for EJB 3.0 and EJB 3.1
- `web.xml` for Servlet 2.5 and Servlet 3.0

The following table describes the combinations of the DD version in `application.xml` and the omission of the DD, and the DD (`ejb-jar.xml`, `web.xml`, and `ra.xml`) version for each module and the omission of the DD.

Table 11-6: DD combinations

DD types		application.xml			
		1.4 or earlier #1	Java EE 5	Java EE 6	No DD
ejb-jar.xml	2.0 or earlier #1	Y	Y	Y	Y
	2.1	Y	Y	Y	Y
	3.0	N	Y	Y	Y
	3.1	N	N	Y	Y
	No DD (3.0 or later)	Y (3.0)	Y (3.0)	Y (3.1)	Y (3.1)#2

DD types		application.xml			
		1.4 or earlier #1	Java EE 5	Java EE 6	No DD
web.xml	2.3 or earlier #1	Y	Y	Y	Y
	2.4	Y	Y	Y	Y
	2.5	N	Y	Y	Y
	3.0	N	N	Y	Y
	No DD (2.5 or later)	N	Y (2.5)	Y (3.0)	Y (3.0)#2
ra.xml	1.0	Y	Y	Y	Y
	1.5	Y	Y	Y	Y

Legend:

Y: Supported

N: Not supported

#1 When the version is upgraded or when the server management commands are executed after the version is upgraded, the version is changed as described below.

#2 Even if the J2EE applications that are already imported with Application Server version 08-70 or earlier are upgraded to version 09-00 or later, the `ejb-jar.xml` version operates as 3.0 and the `web.xml` version operates as 2.5.

Table 11-7: Upgrading the DD versions

DD	Version before upgrade	Version after upgrade
application.xml	1.2	1.4
	1.3	
ejb-jar.xml	1.1	2.0
web.xml	2.2	2.3

Reference note

When you execute the following commands, the message KDJE42361-E is output and an error occurs if the configuration is not supported by the J2EE application:

- `cjimportapp`
- `cjreplaceapp`
- `cjaddapp`

11.4.2 Differences in functionality depending on the presence of application.xml

When `application.xml` is omitted, differences occur in the assemble functionality and deploy functionality when the commands are executed. The following table separately describes the differences in the server management command functionality for the J2EE applications with `application.xml` and the J2EE applications without `application.xml`.

Table 11-8: List of differences in functionality depending on the presence of application.xml

Commands operating the J2EE application	application.xml is present	application.xml is absent
cjimportapp	<p>Determining the module The module is determined using application.xml.</p> <p>Display name of the J2EE application Determined using the <display-name> tag of application.xml. If the <display-name> tag does not exist, the display name is determined using the EAR file name.</p> <p>Determining the EJB-JAR directory and WAR directory In the exploded archive format (the -a option is specified), the directory name is the name excluding the extension from the path name coded in the <module> tag of application.xml.</p>	<p>Determining the module The module is determined using the file position, extension, and file contents.</p> <p>Display name of the J2EE application Determined using the EAR file name.</p> <p>Determining the EJB-JAR directory and WAR directory In the exploded archive format, the EJB-JAR directory name ends with _jar and the WAR directory name ends with _war.</p>
cjexportapp	Exports a J2EE application with application.xml.	Exports a J2EE application without application.xml.
cjaddapp	Adds resources into the J2EE application. Adds the module information into application.xml. There are no restrictions on the resource file extensions that can be added.	Adds resources into the J2EE application. At that time, if the resource file extension does not conform to the Java EE specifications related to the omission of application.xml, an error occurs.
cjsetappprop	Sets the contents specified in the property file.	Sets the contents specified in the property file. Creates application.xml using the set contents.
cjrenameapp	Changes the J2EE application name.	Changes the J2EE application name. Creates application.xml.

Reference note

The functionality of the following server management commands does not differ when application.xml exists and when application.xml does not exist:

- cjdeleteapp
- cjstartapp
- cjstopapp
- cjchmodapp
- cjgencmpsl
- cjgetappprop
- cjgetstubsjar
- cjimportlibjar
- cjdeletelibjar
- cjlistlibjar
- cjlistapp
- cjreloadapp
- cjreplaceapp

11.4.3 Rules for determining the modules when application.xml exists

When `application.xml` exists in an application, the modules except the library JAR are determined from the contents of `application.xml` according to the Java EE specifications. However, if the `application.xml` version is 1.4 or earlier, the library JAR is determined according to the specifications unique to Application Server. The rules for determining the library JAR modules are different when the `application.xml` version is Java EE 5 or later and when the version is J2EE 1.4 or earlier. This subsection describes the rules for determining the library JAR modules separately for the cases when the `application.xml` version is Java EE 5 or later and when the version is J2EE 1.4 or earlier.

(1) When the application.xml version is Java EE 5 or later

When the `application.xml` version is Java EE 5 or later, excluding the following files, the JAR files (files with extension `.jar` in lower case) directly beneath the library directory and those directly beneath the J2EE application root are considered as library JAR:

Excluded files

- Files written in the `<module>` tag of `application.xml` directly beneath the `META-INF` directory
- `hitachi-runtime.jar` directly beneath the J2EE application root

The following table lists the rules for determining the modules when the `application.xml` version is Java EE 5 or later.

Table 11-9: Rules for determining the modules when the application.xml version is Java EE 5 or later

Value of the <code><library-directory></code> tag	Library JAR [#]				
Existing directory	<table border="1"> <tr> <td>/</td> <td><code>file-name.jar</code></td> </tr> <tr> <td>Other than /</td> <td> <ul style="list-style-type: none"> • <code>file-name.jar</code> • Value of the <code><library-directory></code> tag/ <code>file-name.jar</code> </td> </tr> </table>	/	<code>file-name.jar</code>	Other than /	<ul style="list-style-type: none"> • <code>file-name.jar</code> • Value of the <code><library-directory></code> tag/ <code>file-name.jar</code>
/	<code>file-name.jar</code>				
Other than /	<ul style="list-style-type: none"> • <code>file-name.jar</code> • Value of the <code><library-directory></code> tag/ <code>file-name.jar</code> 				
Non-existent directory	<code>file-name.jar</code>				
Existing file	KDJE42360-E is output during import and an error occurs.				
Null	<code>file-name.jar</code>				
Tag does not exist	<ul style="list-style-type: none"> • <code>file-name.jar</code> • <code>/lib (lower case)/file-name.jar</code> 				

#

Excluding the files listed in *Excluded files*.

(2) When the application.xml version is J2EE 1.2, 1.3, or 1.4

The JAR files (the files with extension `.jar` in lower case) other than the following files are considered as library JAR:

- Files written in the `<module>` tag of `application.xml` directly beneath the `META-INF` directory
- `hitachi-runtime.jar` directly beneath the J2EE application root

(3) Examples of JAR files handled as a library JAR

If `library-directory/file-name.jar` is written in the `<module>` tag of `application.xml` directly beneath the `META-INF` directory, that JAR file is not recognized as a library JAR. Also, `hitachi-runtime.jar` is also not recognized as a library JAR. The following table lists the examples of JAR files handled as a library JAR.

Table 11–10: Examples of JAR files handled as a library JAR

Path in the EAR file	application.xml version			
	J2EE1.4	Java EE 5 or later		
		<library-directory> value		
		lib	library	No tags
lib1.jar	Library JAR	Library JAR	Library JAR	Library JAR
lib/lib2.jar	Library JAR	Library JAR	--	--
library/lib3.jar	Library JAR	--	Library JAR	--

Legend:
 --: Not applicable

11.4.4 Rules for determining the modules when application.xml does not exist

This subsection describes the rules for determining the modules when `application.xml` does not exist in the J2EE application. When `application.xml` does not exist in the J2EE application, the rules for determining the library JAR differ for each application format. The rules for determining the modules for each application format are as follows:

(1) For the archive-format applications or the exploded archive-format applications imported by specifying the `-d` option

If an application not containing `application.xml` is imported in the archive format or is imported in the exploded archive format (specifying `-d` option in the `cjimportapp` command), the module is determined according to the Java EE specifications. The order in which the modules are determined is as follows:

1. All the files with the extension `.war` are considered as the Web modules.
2. All the files with the extension `.rar` are considered as the resource adapters.
3. The `lib` directory is considered as the library directory.
4. Except the files under the `lib` directory, the files with extension `.jar` are determined as follows:
 - The JAR files containing `META-INF/ejb-jar.xml` file or the EJB component annotations are considered as the EJB modules.
 - The JAR files directly beneath the J2EE application root, except `hitachi-runtime.jar`, are considered as the library JAR.

All the files that do not satisfy these rules are ignored.

Tip

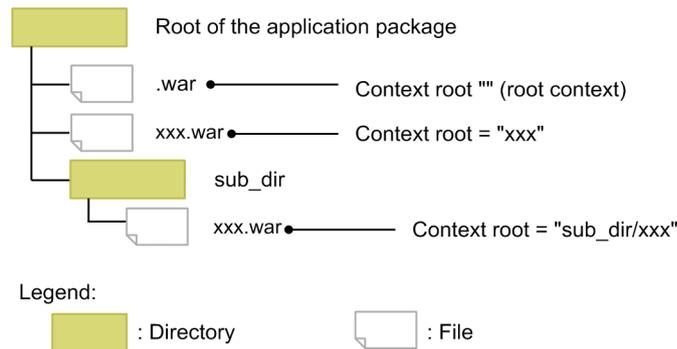
Context root of a Web module

The context root of the Web module is the string excluding the extension `.war` from the relative path from the application package root to the WAR file.

Therefore, when `application.xml` is omitted, specify the directory name of the WAR file path and the WAR file name using characters available in URI (RFC3986).

The following figure shows the Web module configuration and context root example.

Figure 11-10: Web module configuration and context root example (For the archive-format applications or for the exploded archive-format applications imported by specifying the -d option)



Note that depending on the WAR file path, the context root might be duplicated for multiple WAR files. For example, if a WAR file named 'sub/.war' and a WAR file named 'sub.war' exist, the context root for both becomes 'sub'.

(2) For the exploded archive-format applications imported by specifying the -a option

If an application not containing `application.xml` is imported in the exploded archive format (specifying `-a` option in the `cjimportapp` command), the WAR directory, EJB-JAR directory, and modules are determined sequentially according to the following rules. The order of determining the modules is as follows:

1. If a directory ending with `_war` exists in the layer below the root of the application directory, that directory is considered as the WAR directory. However, if the following conditions are fulfilled, the directory is not considered as the WAR directory:
 - Directories under the WAR directory
 - Directories under the EJB-JAR directory
 - `lib` directory
2. If a directory ending with `_jar` exists in the layer below the root of the application directory, that directory is considered as the EJB-JAR directory. However, if the following conditions are fulfilled, the directory is not considered as the EJB-JAR directory:
 - Directories under the WAR directory
 - Directories under the EJB-JAR directory
 - `lib` directory and the directories under the `lib` directory
3. Except the files under the WAR directory and the EJB-JAR directory, all the files with extension `.rar` are considered as the resource adapters.
4. The `lib` directory is considered as the library directory.
5. Except the files under the WAR directory, EJB-JAR directory, and the `lib` directory, for the files with extension `.jar`, the JAR files directly beneath the J2EE application root, excluding `hitachi-runtime.jar`, are considered as library JAR.

All the files that do not satisfy these rules are ignored.

Tip

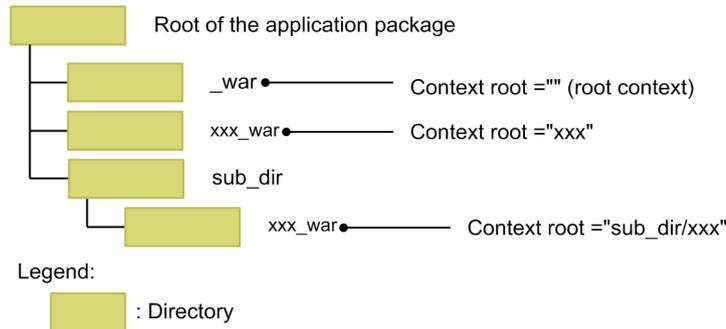
Context root of a Web module

The context root of the Web module is the string excluding the extension `_war` from behind the relative path from the application directory to the WAR directory.

Therefore, when `application.xml` is omitted, specify the directory name of the WAR directory path and the WAR directory name using characters available in URI (RFC3986).

The following figure shows the Web module configuration and context root example.

Figure 11-11: Web module configuration and context root example (For the exploded archive-format applications imported by specifying the -a option)



Note that depending on the WAR directory path, the context root might be duplicated for multiple WAR directories. For example, if a WAR directory named 'sub/_war' and a WAR directory named 'sub_war' exist, the context root for both becomes 'sub'.

(3) Method of handling the library directory in the EAR file or the application directory

This section describes the method of handling the library directory in the EAR file or the application directory when `application.xml` does not exist. The method of handling the library directory differs when a directory or file named `lib` exists and when both do not exist in the EAR file or application directory.

Table 11-11: Method of handling the library directory in the EAR file or application directory when `application.xml` does not exist

lib in the EAR file or lib in the application directory	Handling method
When the directory exists	<code>lib</code> is handled as the library directory.
When the file exists	KDJE42360-E is output during import and an error occurs.
When the directory or file do not exist	Operations are performed assuming the library directory does not exist. (<code>lib</code> existing in other locations is not handled as the library directory).

11.4.5 Operations for Web applications in which `web.xml` is omitted

When `web.xml` is omitted, the operations for the Web applications with the server management commands are limited. The following table describes whether the operations can be executed for the Web applications when `web.xml` is omitted.

Table 11-12: Operability of the Web applications when `web.xml` is omitted

Operations			Operability
Commands	Arguments	Operations	
cjaddapp	-type filter	Adding a filter	N ^{#1}
	-type war	Adding a WAR file	Y
cjcopyres	-type war	Copying a WAR file	Y
cjdeleteapp	-type filter	Deleting a filter	N ^{#2}
	-type war	Deleting a WAR file	Y
	None	Deleting an application	Y
cjdeleteres	-type war	Deleting a WAR file	Y

Operations				Operability
Commands	Arguments	Operations		
cjexportapp	-raw	Exporting applications (not containing runtime information)		Y ^{#3}
	-normal	Exporting applications (containing runtime information)		Y ^{#3}
cjgetappprop	-type war	Obtaining the WAR file attributes		Y
	-type all	Obtaining the application attributes		Y
cjgetresprop	-type war	Obtaining the WAR file attributes		Y
cjimportapp			Importing a J2EE application	Y ^{#4}
cjimportres	-type war	Importing a WAR file		Y ^{#4}
cjlistapp	-type war	Displaying the list of WAR files		Y
cjlistres	-type war	Displaying the list of WAR files		Y
cjreloadapp			Reloading an application	Y
cjreplaceapp	-replaceDD	Replacing an application (DD definition is not inherited)		Y
	None	Replacing an application (DD definition is inherited)		Y
cjsetappprop	-type war	Attribute settings	Information related to the default DD	N ^{#5}
			Runtime information	Y
	-type all	Attribute settings	Information related to the default DD	N ^{#5}
			Runtime information	Y
cjsetresprop	-type war	Attribute settings	Information related to the default DD	N ^{#5}
			Runtime information	Y
cjstartapp			Starting an application	Y
cjstopapp			Stopping an application	Y

Legend:

Y: Operations can be executed

N: Operations cannot be executed

#1: An error occurs during command execution and the message KDJE37606-E is output.

#2: A filter cannot be added in an application in which `web.xml` is omitted, therefore, the filter can also not be deleted.#3: An exported file does not include `web.xml`.

#4: An imported WAR can only use the JSP and static contents.

#5: Handled as a read-only attribute, so the settings are ignored.

11.4.6 Display name specified when the DD is omitted

With Application Server, the display name is supplemented during import when the DD is omitted. Also, the J2EE server checks the validity of the display name. This subsection describes the supplemental rules for the display names during import and the checking of the validity of the display names.

(1) Supplemental rules for the display names during import

If the `<display-name>` tag of the DD for the J2EE components EJB-JAR, WAR, RAR, or EAR is not specified, the J2EE server sets the display name. The display name is determined on the basis of the file name and directory name of the J2EE components. Even when the DD is omitted, the J2EE server sets the display name. The characters

that can be used as the file name and directory name in this case are one-byte alphanumeric characters (0 to 9, A to Z, and a to z) and underscore (_). When you want to specify the display name for EJB-JAR, WAR, RAR, or EAR, we recommend that you specify the `<display-name>` tag.

(2) Checking the validity of the display names

The characters that can be used in the display name of the lang attribute `en` of the J2EE applications, EJB-JARs, WARs, RARs, and the components (Enterprise Beans, Servlets, JSPs, Filters) are alphanumeric characters (0 to 9, A to Z, and a to z) and underscore (_). If characters other than one-byte alphanumeric characters are included in the display name, the J2EE server converts the characters to underscore (_). The J2EE server checks the validity of the display name when the J2EE applications, EJB-JARs, WARs, and RARs are imported and issues a warning message KDJE42374-W if characters that cannot be used are used in the display name.

However, if the display name is `TP1/Message Queue - Access`, a warning message is not issued.

The following is a description on the server management commands used for checking the validity of the display names:

- `cjimportapp` command
This command checks the validity of the display names of the applications and the J2EE resources included in the applications (EJB-JARs, WARs, RARs, Enterprise Beans, Servlets/JSPs, and Filters). The validity is not checked for the ZIP format with runtime information.
- `cjimportwar` command
This command checks the validity of the display names of the J2EE resources (WAR, Servlet/JSP, and Filter) included in the applications.
- `cjimportres` command
This command checks the validity of the display names of the J2EE resources (EJB-JARs, WARs, RARs, Enterprise Beans, Servlets/JSPs, and Filters). The validity is not checked for the RAR format with runtime information.

The following table lists the display names for which the validity is checked.

Table 11-13: List of display names for which the validity is checked

No.	DD elements indicating the display name	Definition types
1	<code><application><display-name></code>	<code>application.xml</code> definition
2	<code><ejb-jar><display-name></code>	<code>ejb-jar.xml</code> definition
3	<code><ejb-jar><enterprise-beans><session><display-name></code>	<code>ejb-jar.xml</code> definition
4	<code><ejb-jar><enterprise-beans><entity><display-name></code>	<code>ejb-jar.xml</code> definition
5	<code><ejb-jar><enterprise-beans><message-driven><display-name></code>	<code>ejb-jar.xml</code> definition
6	<code><web-app><display-name></code>	<code>web.xml</code> definition
7	<code><web-app><filter><display-name></code>	<code>web.xml</code> definition
8	<code><web-app><servlet><display-name></code>	<code>web.xml</code> definition
9	<code><connector><display-name></code>	<code>ra.xml</code> definition

11.4.7 Operations for creating application.xml when application.xml is omitted

If `application.xml` does not exist in the imported J2EE application, `application.xml` is created when you execute the following operations:

- When the `cjsetappprop` command is used to change one of the following tag values in the property files:

HITACHI Application Property file

```
<hitachi-application-property>/<description>
<hitachi-application-property>/<icon>/<small-icon>
<hitachi-application-property>/<icon>/<large-icon>
```

WAR property file

```
<hitachi-war-property>/<war-runtime>/<context-root>
```

- When the `cyjrenameapp` command is executed

11.4.8 Notes on adding resources in the J2EE applications when application.xml is omitted

The notes on adding the imported EJB-JAR file, WAR file, or RAR file in a J2EE application without `application.xml` are as follows:

When `application.xml` does not exist in the J2EE application, the resource file extension is determined according to the Java EE specifications as follows:

- Extension of EJB-JAR file is `.jar`
- Extension of WAR file is `.war`
- Extension of RAR file is `.rar`

If the resource file does not have the above extensions, the resource file cannot be added into the J2EE application. If you add such a resource file, the message KDJE42366-E is displayed and an error occurs.

12

Using Annotations

This chapter describes the annotation functionality.

Reference this chapter when you want to use annotations in the applications of Servlet 2.5 or later or EJB 3.0 or later.

12.1 Organization of this chapter

An *annotation* is a coding method that is used to embed the additional information for classes and methods in a source code. By using annotations, the information that was so far specified in the DD (`web.xml` or `ejb-jar.xml`) can be specified in the source code of the Servlets and Enterprise Beans.

The following table describes the organization of this chapter.

Table 12-1: Organization of this chapter (Using annotations)

Title	Reference location
Specifying annotations	<i>12.2</i>
Classes to be loaded and the class path required for loading	<i>12.3</i>
Using the DI	<i>12.4</i>
Controlling the annotation references	<i>12.5</i>
Updating the contents defined in the annotations	<i>12.6</i>
Notes on using annotations	<i>12.7</i>

12.2 Specifying annotations

This section describes the types of annotations that can be specified with Application Server and the points at which the applications specifying annotations are implemented.

The following table describes the organization of this section.

Table 12-2: Organization of this section (Specifying annotations)

Category	Title	Reference location
Explanation	Merits of using annotations and the annotations that can be specified	12.2.1
	Using the library JAR class with a declared annotation	12.2.2
Implementation	Implementing the Enterprise Beans when an annotation is specified	12.2.3

Note: The functionality-specific explanation is not available for "Settings", "Operations", and "Notes".

12.2.1 Merits of using annotations and the annotations that can be specified

This subsection describes the merits of using annotations and the annotations that can be specified.

(1) Merits of using annotations

The merits of using annotations are as follows:

- You can consolidate the information distributed across the source codes and the DDs in the source codes.
- You need not create the DDs of the Web applications and Enterprise Beans.
- You can use the DI to acquire the references to other Enterprise Beans and resources.

For details on the DI, see *12.4 Using the DI*.

(2) Annotations that can be specified in a Web application

Annotations can be specified in the Web applications for Servlet 2.4, Servlet 2.5, or Servlet 3.0. However, the annotations that can be specified depend on the Servlet version.

For details on the annotations that can be specified, see *2.1 Range of supported annotations* in the *uCosminexus Application Server API Reference Guide*.

For details on the types of resources that you can specify in the `@Resource` annotation, see *12.4.1 Types of resources that can be specified in the @Resource annotation*. Also, the `name` attribute of the `@EJB` and `@Resource` annotations corresponds to the tag elements of `web.xml`. For details on the mapping, check the standard specifications.

Tip

Note that for `web.xml` of Servlet version 2.3 or earlier, specification of annotations is disabled.

(3) Annotations that can be specified in the Enterprise Beans

Annotations can be specified in the Enterprise Beans for EJB 3.0.

For details on the annotations that can be specified, see *2.1 Range of supported annotations* in the *uCosminexus Application Server API Reference Guide*.

Note that when the home interface is specified by using the `@RemoteHome` or `@LocalHome` annotation, the return value of the `create` method of the specified home interface is considered as the component interface.

12.2.2 Using the library JAR class with a declared annotation

The classes included in the library JAR are used from each component. The annotations supported in the components that use these classes can be coded in the library JAR class.

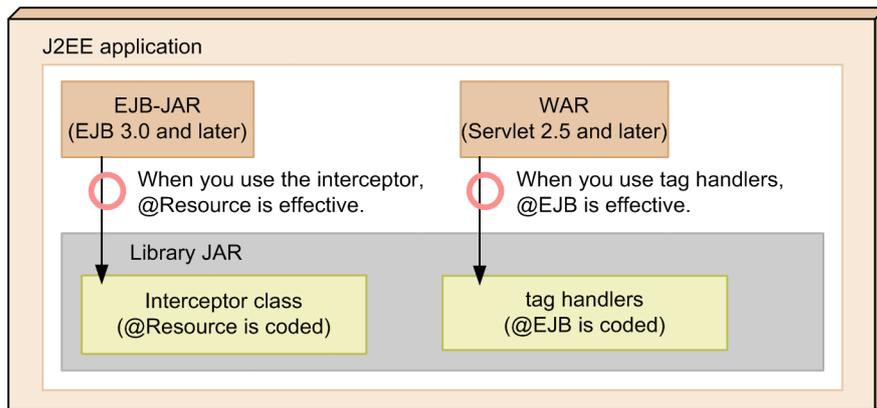
The constraints on the annotations specifiable in the library JAR are as follows:

- With the library JAR, you cannot specify annotations that declare the components. Such an annotation is ignored even if specified.
- The annotations defined in the classes that form the library JAR components are not supported.
- The annotations declared in the library JAR are ignored when referenced from the following EJB-JARs and WARs:
 - EJB-JARs containing EJB 2.1 or earlier modules
 - WARs containing Servlet 2.4 or earlier modules
- The annotations declared in the library JAR are ignored when referenced from EJB 3.0 (no DD) included in the J2EE applications of J2EE 1.4.

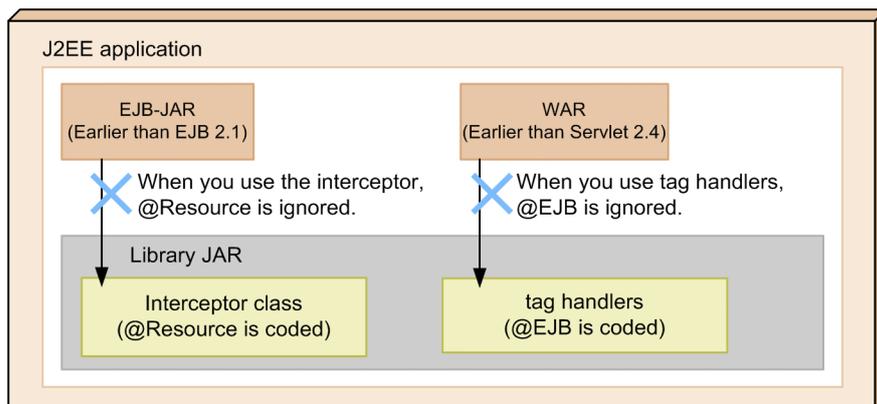
The following figure shows an example of usage of the classes coding the annotations in the library JAR.

Figure 12-1: Example of usage of the classes coding the annotations in the library JAR

- When referencing from EJB-JAR of EJB 3.0 and later, or WAR of Servlet 2.5 and later



- When referencing from EJB earlier than EJB 2.1 or WAR earlier than Servlet 2.4



When the annotation is referenced from EJB 3.0 or later or Servlet 2.5 or later, @Resource and @EJB included in the library JAR are enabled. When referenced from components of EJB 2.1 or earlier or Servlet 2.4 or earlier, @Resource and @EJB included in the library JAR are ignored. However, the J2EE application version at the reference source is presumed to be Java EE 5 or later.

12.2.3 Implementing the Enterprise Beans when an annotation is specified

For Enterprise Beans, you can specify annotations in Session Bean. This subsection describes the points for implementing Stateless Session Bean or Stateful Session Bean by specifying annotations.

(1) Implementing Stateless Session Bean

- Specify the type of Enterprise Bean (Stateless Session Bean) either in the `@Stateless` annotation or in the DD (`ejb-jar.xml`).
- You can implement Session Bean by using a business interface.
- Implementing the `javax.ejb.SessionBean` interface is not mandatory.
- Implement an interceptor class as and when required.
- You can define the following callbacks in the Stateless Session Bean:
 - `@PostConstruct`
 - `@PreDestroy`

For details on callback or the method invocation time, see *2.2.3 Life cycle of Enterprise Beans* in the *uCosminexus Application Server EJB Container Functionality Guide*.

(2) Implementing Stateful Session Bean

- Specify the type of Enterprise Bean (Stateful Session Bean) either in the `@Stateful` annotation or in the DD (`ejb-jar.xml`).
- You can implement Session Bean by using a business interface.
- Implementing the `javax.ejb.SessionBean` interface and the `java.io.Serializable` interface is not mandatory.
- Implement an interceptor class as and when required.
- You can define the following callbacks:
 - `@PostConstruct`
 - `@PreDestroy`
- You can specify the `@Init` annotation and `@Remove` annotation in the methods of the implementation class.

For details on callback or the method invocation time, see *2.2.3 Life cycle of Enterprise Beans* in the *uCosminexus Application Server EJB Container Functionality Guide*.

12.3 Classes to be loaded and the class path required for loading

The annotation information is read when the operations such as those listed below are executed:

- Adding resources (`cjaddapp`)[#]
- Importing (`cjimportapp`)
- Importing (`cjimportwar`)
- Importing (`cjimportres`)
- Importing (`cjimportlibjar`)[#]
- Starting (`cjstartapp`)
- Replacing (`cjreplaceapp`)
- Reloading (`cjreloadapp`)

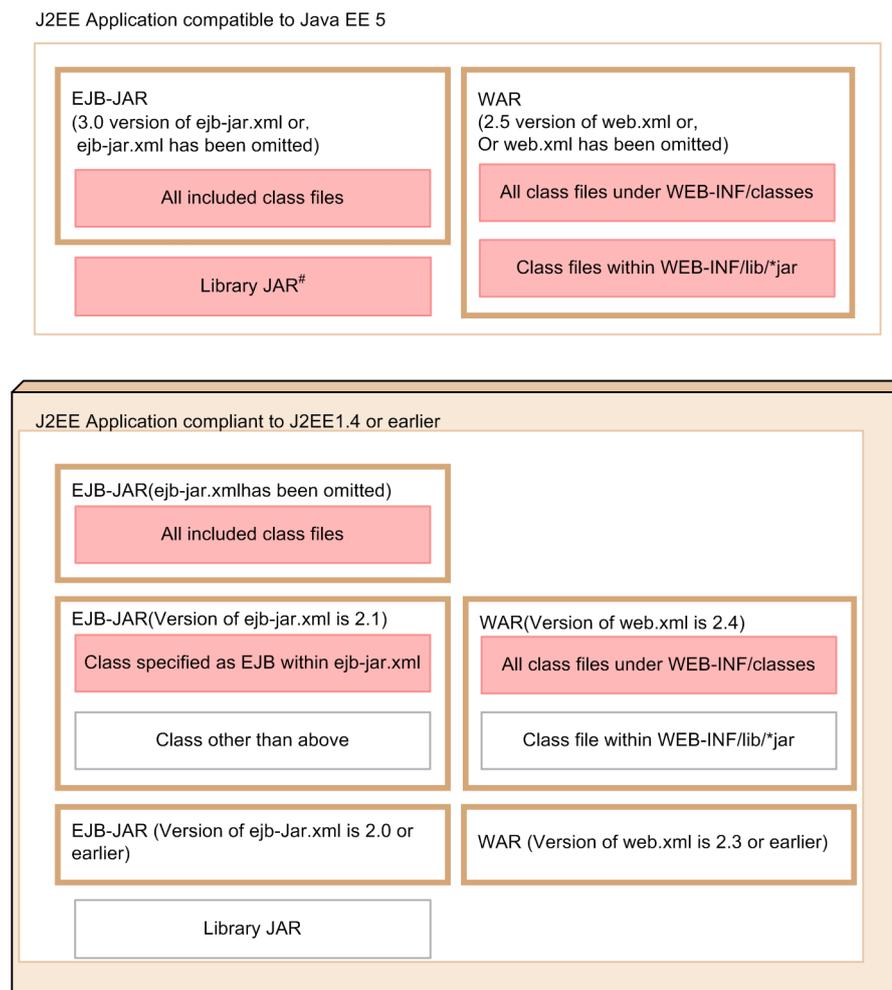
#

This operation is executed on the library JAR classes.

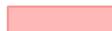
To read the annotation information, you must first load the classes.

The following figure shows the classes loaded for reading the annotation information.

Figure 12-2: Classes loaded for reading the annotation information



Legend:



: Class that becomes acquisition target of annotation information



: Class that does not become acquisition target of annotation information

#: When false is set in value of property `Ejbserver.deploy.annotations.load_libjars.enabled`, annotation information within class of Library JAR is not acquired.

Note that the annotation when the annotation information is read from the class files included in the EJB-JAR (`ejb-jar.xml` version 2.1) or from the class files included in the WAR (`web.xml` version 2.4) is a Cosminexus Application Server-specific specification. However, this functionality is used for compatibility with earlier versions. For details, see *5.4 Using annotations with EJB 2.1 and Servlet 2.4 in the uCosminexus Application Server Compatibility Guide*.

Note that in the J2EE applications compliant with J2EE 1.4 or earlier, the library JAR cannot be used from the EJB-JAR (`ejb-jar.xml` version 2.1, or `ejb-jar.xml` is omitted) and WAR (`web.xml` version 2.4). The library JAR is ignored even if specified.

The loading of a class for obtaining the annotation information depends on the version of each module and the attributes defined in the DD.

The following table lists the conditions for determining whether to obtain the annotation information when the module version is EJB-JAR (3.0) or WAR (2.5).

Table 12-3: When the module version is EJB-JAR (3.0) or WAR (2.5)

Module	metadata-complete tag of the DD		DD is omitted
	false or tag is omitted	true	
EJB-JAR (3.0)	Y	--	Y
WAR (2.5)	Y	--	Y

Legend:

- Y: Annotation information is obtained
- : Annotation information is not obtained

The following table describes whether to obtain the annotation information for the library JAR.

Table 12-4: For the library JAR

Module	J2EE application compliant with Java EE 5 or later			J2EE application compliant with J2EE 1.4 or earlier		
	metadata-complete tag of the DD		DD is omitted [#]	metadata-complete tag of the DD		DD is omitted
	false or tag is omitted [#]	true		false or tag is omitted	true	
EJB-JAR (3.0)	Y	--	Y	N	N	--
WAR (2.5)	Y	--	Y	N	N	N

Legend:

- Y: Annotation information is obtained (obtained if the J2EE application contains even one relevant module)
- : Annotation information is not obtained
- N: Combination is not supported

If you specify `false` in the value of the property `ejbserver.deploy.annotations.load_libjars.enabled`, the annotation information in the library JAR classes will not be obtained.

! Important note

If the class required for reading the annotation information is missing or if the class is invalid, an exception occurs during the process of obtaining the annotation information. The behavior when an exception occurs while a class is being loaded is as follows:

When Application Server version is 08-00 or later

The exception log and message KDJE42380-W is output and the processing continues. Note that if you want to specify an error when an exception is thrown, change the value of the property `ejbserver.deploy.annotations.load_check.enabled` from `false` to `true`.

When Application Server version is earlier than 08-00

An error occurs and the processing is cancelled.

12.4 Using the DI

Dependency Injection (*DI*) is a functionality with which the EJB container automatically sets the references to the Enterprise Beans and resources, by specifying the `@EJB` and `@Resource` annotations in the fields and the `setter` method of the EJBs and interceptor classes. If you use the DI, the references to the Enterprise Beans and resources need not be looked up by using the JNDI. This section describes the types of resources that can be specified in the `@Resource` annotation, the resolution of the resource references using the `@Resource` annotation, and the notes on the DI.

The following table describes the organization of this section.

Table 12-5: Organization of this section (Using the DI)

Category	Title	Reference location
Explanation	Types of resources that can be specified in the <code>@Resource</code> annotation	12.4.1
	Resolving the resource references using the <code>@Resource</code> annotation	12.4.2
	Operations during DI failure	12.4.3
Notes	Notes	12.4.4

Note: The functionality-specific explanation is not available for "Implementation", "Settings", and "Operations".

12.4.1 Types of resources that can be specified in the `@Resource` annotation

You can use the `@Resource` annotation to define a reference. The following table lists the types of resources that can be specified in the `@Resource` annotation.

Table 12-6: Types of resources that can be specified in the `@Resource` annotation

Types of resources	Specification
<code>javax.sql.DataSource</code> ^{#1}	Y
<code>javax.mail.Session</code>	Y
<code>java.net.URL</code>	N
<code>javax.jms.ConnectionFactory</code>	Y
<code>javax.jms.QueueConnectionFactory</code> ^{#2}	Y
<code>javax.jms.TopicConnectionFactory</code>	Y
<code>javax.jms.Queue</code> ^{#2}	Y
<code>javax.jms.Topic</code>	Y
<code>javax.resource.cci.ConnectionFactory</code> ^{#3}	Y
<code>javax.resource.cci.InteractionSpec</code>	N
<code>javax.transaction.UserTransaction</code>	Y ^{#4}
<code>org.omg.CORBA_2_3_ORB</code>	Y ^{#5}
<code>javax.xml.rpc.Service</code>	N
<code>javax.xml.ws.Service</code>	N

Types of resources	Specification
<code>javax.jws.WebService</code>	N
<code>javax.ejb.EJBContext</code>	Y#6
<code>javax.ejb.SessionContext</code>	Y#6
<code>javax.ejb.TimerService</code>	Y#6#7
Resources specific to the JavaBeans resources	Y
<code>java.lang.String</code>	Y#8
<code>java.lang.Character</code>	Y#8
<code>java.lang.Integer</code>	Y#8
<code>java.lang.Boolean</code>	Y#8
<code>java.lang.Double</code>	Y#8
<code>java.lang.Byte</code>	Y#8
<code>java.lang.Short</code>	Y#8
<code>java.lang.Long</code>	Y#8
<code>java.lang.Float</code>	Y#8

Legend:

- Y: Can be specified
- N: Cannot be specified

#1: DB Connector is applicable.

#2: TP1/Message Queue - Access and Cosminexus RM are applicable.

#3: uCosminexus TP1 Connector is applicable.

#4: Cannot be used with the Enterprise Beans or interceptors operated with the CMT.

#5: The operation is performed assuming that `true` is specified in the `shareable` attribute of ORB. Note that the injected ORB objects are the shared instances used even in other components.

#6: Cannot be used with the classes running on the Web container.

#7: Cannot be used with Stateful SessionBean and the interceptors applicable to Stateful SessionBean.

#8: In the `<env-entry-value>` tag, you cannot set a value that can be acquired with the DI or lookup.

12.4.2 Resolving the resource references using the `@Resource` annotation

There are two methods of resolving the resource references by using the `@Resource` annotation:

- Method of specifying the optional name of the resource in the `name` attribute
- Method of specifying the reference destination resource in the `mappedName` attribute

If you do not want to specify attributes in the `@Resource` annotation, the links must be resolved by using `cosminexus.xml` or property files other than `cosminexus.xml`. The following sections describe each of the methods.

(1) Method specified with the `name` attribute of the `@Resource` annotation

Specify the optional name of the resource in the `name` attribute of the `@Resource` annotation.

The resources that can be specified in the `name` attribute of the `@Resource` annotation are as follows:

- `javax.sql.DataSource`
- `javax.jms.ConnectionFactory`
- `javax.jms.QueueConnectionFactory`
- `javax.jms.TopicConnectionFactory`
- `javax.resource.cci.ConnectionFactory`

The examples of coding are as follows. In this example, the optional name `jdbc/ds` is set in the `name` attribute of the `@Resource` annotation.

```
package sample;

@Stateless public class MySessionBean implements MySession {
    @Resource(name="jdbc/ds") public DataSource customerDB;
    ...
}
```

If the optional name of the resource adapter is specified in the `name` attribute of the `@Resource` annotation, the reference resolution is implemented automatically. Therefore, the property file need not be edited. If the property file is obtained, the coding is as follows:

```
<resource-ref>
  <description xml:lang="en"></description>
  <res-ref-name>jdbc/ds</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
  <res-sharing-scope>Shareable</res-sharing-scope>
  <injection-target>
    <injection-target-class>sample.MySessionBean</injection-target-class>
    <injection-target-name>customerDB</injection-target-name>
  </injection-target>
  <linked-to></linked-to>
</resource-ref>
```

(2) Method specified with the `mappedName` attribute of the `@Resource` annotation

Specify the resource display name and queue display name in the `mappedName` attribute of the `@Resource` annotation.

The resource display name coded in the `mappedName` attribute of the `@Resource` annotation is used for reference resolution.

The examples of coding are as follows. In this example, the resource display name `DB_Connector_for_Oracle` is set in the `mappedName` attribute of the `@Resource` annotation.

```
package sample;

@Stateless public class MySessionBean implements MySession {
    @Resource(mappedName="DB_Connector_for_Oracle") public DataSource customerDB;
    ...
}
```

In this example, the `name` attribute of `@Resource` is omitted; therefore, the default value `class-name/field-name` is used. The default value in this case is `sample.MySessionBean/customerDB`. If the property file is obtained, the coding is as follows:

```
<resource-ref>
  <description xml:lang="en"></description>
  <res-ref-name>sample.MySessionBean/customerDB</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
  <res-sharing-scope>Shareable</res-sharing-scope>
  <mapped-name>DB_Connector_for_Oracle</mapped-name>
  <injection-target>
    <injection-target-class>sample.MySessionBean</injection-target-class>
    <injection-target-name>customerDB</injection-target-name>
  </injection-target>
```

```
<linked-to></linked-to>
</resource-ref>
```

(3) Method specified with `cosminexus.xml` or other property files

If the attributes of the `@Resource` annotation are not specified, the references must be resolved by using `cosminexus.xml` and the property files other than `cosminexus.xml`. Specify the resource display name in `<linked-to>` of `cosminexus.xml` or property files other than `cosminexus.xml`. If the resource at the reference destination is a queue, code the resource adapter display name and queue display name in `<linked-queue>`. If the resource at the reference destination is an Administered object, code the resource adapter display name and the Administered object name in `<linked-adminobject>`.

The examples of coding are as follows. In this example, the name attribute of the `@Resource` annotation is omitted; therefore, the default value *class name/ field name* is used as the optional name. The default value in this case is `sample.MySessionBean/customerDB`.

```
package sample;

@Stateless public class MySessionBean implements MySession {
    @Resource public DataSource customerDB;
    ...
}
```

To resolve the references with `cosminexus.xml`, specify the resource display name in `<link-to>`, `<linked-queue>`, or `<linked-adminobject>` and then import the application. To resolve the references with the property files other than `cosminexus.xml`, import the application and then obtain the property file (such as HITACHI Application Integrated Property File). In `<linked-to><linked-queue>` or `<linked-adminobject>` in the obtained property file, code the resource display name that forms the reference destination.

When the property file is obtained after specifying the attributes, the coding is as follows:

```
<resource-ref>
  <description xml:lang="en"></description>
  <res-ref-name> sample.MySessionBean/customerDB</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
  <res-sharing-scope>Shareable</res-sharing-scope>
  <injection-target>
    <injection-target-class>sample.MySessionBean</injection-target-class>
    <injection-target-name>customerDB</injection-target-name>
  </injection-target>
  <linked-to>DB_Connector_for_Oracle</linked-to>
</resource-ref>
```

(4) Priority of the reference resolution methods

When you use the `@Resource` annotation to resolve the resource references and if multiple methods are specified at the same time, the settings are enabled in the following priority order:

1. Settings using `cosminexus.xml` or property files other than `cosminexus.xml`
2. Settings in the `mappedName` attribute of the `@Resource` annotation
3. Settings in the `name` attribute of the `@Resource` annotation

The following table describes the optional name settings enabled when a different optional name is set concurrently in the J2EE resources.

Table 12-7: Reference destination settings enabled when multiple methods are set concurrently

Presence of name attribute settings #1	Presence of mappedName attribute settings	Presence of settings such as <code><linked-to></code> in the property file #2	Enabled optional name settings #3
Yes	Yes	Yes	Settings such as <code><linked-to></code> in the property file #2

Presence of name attribute settings #1	Presence of mappedName attribute settings	Presence of settings such as <linked-to> in the property file #2	Enabled optional name settings #3
Yes	Yes	No	mappedName attribute settings of the @Resource annotation
	No	Yes	Settings such as <linked-to> in the property file #2
		No	The name attribute settings of the @Resource annotation
No	<p>If the @Resource annotation is specified in a field/ method, the default value is used and the result is the same as when 'Yes' is specified.^{#4}</p> <p>Also, if the @Resource annotation is specified in a class, the name attribute of the @Resource annotation cannot be omitted. If the name attribute of the @Resource specified in the class does not exist, the reading of the annotation fails and an error occurs during import.^{#5}</p>		

Legend:

Yes: Indicates that the attribute is set in the annotation and property file.

No: The explanation for 'None' is as follows:

- Indicates that the name attribute and the mappedName attribute of the @Resource annotation are not coded or that the <linked-to> tag of cosminexus.xml or property files other than cosminexus.xml does not exist.
- Indicates that the value specified in the attribute or tag does not exist (null).

#1: The optional name of the resource can also be specified in <resource-ref> or <res-ref-name> of the default DD corresponding to the name attribute of the @Resource annotation. If the optional name is not specified in the mappedName attribute of the @Resource annotation and in <linked-to> of the property file, the resource that has the optional name specified in <resource-ref> or <res-ref-name> of the default DD becomes the reference destination.

#2: Includes cosminexus.xml and the property files other than cosminexus.xml.

#3: If the optional name is not set, the application fails to start.

#4: If the optional name is not specified in the mappedName attribute of the @Resource annotation and <linked-to> in the property file (cosminexus.xml and the property files other than cosminexus.xml) and if the resource optional name matches with the default value of the name attribute or the name attribute value of the @Resource annotation specified in the class, the resource having that optional name becomes the reference destination.

#5: If the optional name is not specified in the mappedName attribute and <linked-to> in the property file (cosminexus.xml and the property files other than cosminexus.xml) and if the resource optional name matches with the name attribute value of @Resource specified in the class, the resource having that optional name becomes the reference destination.

12.4.3 Operations during DI failure

This section describes the operations when the DI fails.

If the class specified in the <injection-target-class> tag of the DD or if the method and field specified in the <injection-target-name> tag does not exist, the DI fails. This subsection describes the behavior during application startup and DI execution in this case.

(1) When the class specified in the <injection-target-class> tag of the DD does not exist

The behavior is as follows:

When starting an application

The message (KDJE53905-W) is output. The application start processing continues.

When executing the DI

The DI cannot be executed because the class specified in the <injection-target-class> tag does not exist.

(2) When the method and field specified in the <injection-target-name> tag of the DD does not exist

The behavior is as follows:

When starting an application

The message (KDJE53905-W) is output. The application start processing continues.

When executing the DI

The message (KDJE53900-E) is output and the DI fails.

12.4.4 Notes

This subsection describes the notes on using the DI.

- The notes on obtaining the Stateful Session Bean references when the DI is used are as follows:
 - When the configuration of the J2EE application is such that Stateful Session Bean is invoked from the servlet, obtain the Stateful Session Bean references via the JNDI instead of using the DI.
 - When the configuration of the J2EE application is such that Stateful Session Bean is invoked from Stateless Session Bean or Singleton Session Bean and when the business interface of Stateful Session Bean is injected in Stateless Session Bean by using the DI, the DI is executed every time the business method of Stateless Session Bean is invoked or every time the timeout method is invoked.
- If a resource type that cannot be specified with the @Resource annotation is specified, an exception occurs during import.
- The name attribute of the @EJB and @Resource annotations corresponds to the elements of the `ejb-jar.xml` tags described in the following table. Consequently, when you define the references for the Enterprise Beans and resources using annotations, specify the settings so that the name attribute of the @EJB and @Resource annotations is not duplicated with the elements specified in the `ejb-jar.xml` tags corresponding to the name attribute. The following table describes the `ejb-jar.xml` tags corresponding to the name attribute.

Table 12-8: `ejb-jar.xml` tags corresponding to the name attribute

Annotations and attributes	<code>ejb-jar.xml</code> tags corresponding to the name attribute
@EJB name()	<ejb-ref-name> tag under the <ejb-ref> tag
	<ejb-ref-name> tag under the <ejb-local-ref> tag
@Resource name()	<env-entry-name> tag under the <env-entry> tag
	<res-ref-name> tag under the <resource-ref> tag
	<resource-env-ref-name> tag under the <resource-env-ref> tag

- Exceptions might occur when a Web application that uses DI is reloaded. The cases in which exceptions occur are as follows:
 - When the loading of the class that defines the DI target (the target in which the reference is injected by using the DI functionality) or the reference destination class fails
 - When the fields and methods corresponding to the DI target name are deleted after reloading and no longer exist

If an exception occurs, the KDJE53904-W message is output to the message log and the processing continues. Although the processing continues, you cannot execute the DI and requests for the target instances. As a result, when this message is output, modify the Web application and re-execute the reload operation.

12.5 Controlling the annotation references

This section describes the *functionality for controlling the annotation references*.

If you use the functionality for controlling the annotation references, you can control the parsing for the modules in which annotations are not used. As a result, you can prevent unnecessary overheads and the occurrence of unnecessary parsing errors.

The following table describes the organization of this section.

Table 12–9: Organization of this section (Functionality for controlling the annotation references)

Category	Title	Reference location
Explanation	Purpose and scope of the functionality for controlling the annotation references	12.5.1
	Timing for referencing annotations	12.5.2
Implementation	Definitions in the DD (module settings)	12.5.3
Operations	Changing the settings for the functionality for controlling the annotation references	12.5.4

Note: The functionality-specific explanation is not available for "Settings" and "Notes".

12.5.1 Purpose and scope of the functionality for controlling the annotation references

This subsection describes the purpose and scope of the functionality for controlling the annotation references.

(1) Purpose of the functionality for controlling the annotation references

The functionality for controlling the annotation references is used to specify whether to reference (parsing) annotations in J2EE applications. To determine the presence of annotations in a module, the application must be parsed. However, if all the applications are parsed, unnecessary overhead occurs for the applications not using annotations. Also, for the applications that reference resources and other applications, a class file reference error might occur during annotation parsing and the deployment might fail.

If you use the functionality for controlling the annotation references, you can control the parsing of annotations when an annotation is not used.

You can use the functionality for controlling the annotation references in Servlet 2.5 or later. The functionality for controlling the annotation references cannot be used in EJB 3.0 and later.

The operations in Servlet 2.5 or later are as follows:

In Servlet 2.5 or later

With Java EE 5 and later applications, annotations can be used with the standard specifications. The `metadata-complete` attribute, an attribute for controlling annotation references, is added in the DD of Servlet 2.5 and later. By using the `metadata-complete` attribute, you can control the annotation references for each module.

However, in the case of applications where the DD is omitted, the functionality for controlling the annotation references cannot be used because the annotations are necessarily referenced for reading the module definition information.

The functionality for controlling the annotation references for each module conforms to the standard specifications for Java EE 5 or later.

Reference note

With Application Server, apart from the `<display-name>` element, only the definitions related to the interceptors and application exceptions can be defined in the DD of EJB 3.0. With EJB 3.1, you can define the `<module-name>` element in addition to this, but all the other module definition information for EJB 3.0 or EJB 3.1 must be defined with annotations. Therefore, the functionality for controlling the annotation references cannot be used in EJB 3.0.

! Important note

If you change the functionality for controlling the annotation references from disable to enable after importing a J2EE application containing the annotation-coded EJB 2.1 or Servlet 2.4 modules, the following events occur:

- The annotation information is output to the property file
- When @Resource or @Resources is used, the property file cannot be set up and the J2EE application fails to start

(2) Scope

The scope of the functionality for controlling the annotation references differs according to the module type that includes the annotation.

The following table describes the scope of the functionality for controlling the annotation references.

Table 12-10: Scope of the functionality for controlling the annotation references

Module version		Presence of DD and specified contents		Scope of reference control	
				J2EE servers	Modules
EJB	1.1, 2.0	DD exists (metadata-complete attribute does not exist)		--	--
	2.1 ^{#1}	DD exists (metadata-complete attribute does not exist)		Y ^{#2}	--
	3.0 or later	No DD		N	Y
		DD exists (metadata-complete attribute exists)	Omitted	N	Y
	Specified		N	Y ^{#3}	
Servlet	2.2, 2.3	DD exists (metadata-complete attribute does not exist)		--	--
	2.4 ^{#1}	DD exists (metadata-complete attribute does not exist)		Y ^{#2}	--
	2.5 or later	No DD		N	Y
		DD exists (metadata-complete attribute exists)	Omitted	N	Y
			Specified	N	Y

Legend:

- Y: Enabled
- N: Cannot be used
- : Not applicable

#1: The use of annotations with EJB 2.1 and Servlet 2.4 is the functionality for compatibility with the earlier versions. For details, see *5.4 Using annotations with EJB 2.1 and Servlet 2.4* in the *uCosminexus Application Server Maintenance and Migration Guide*.

#2: If enabled, the J2EE application containing the EJB 2.1 or Servlet 2.4 modules with the annotations coded cannot be used.

#3: You cannot specify true as the value. If true is specified, the importing and redeploying of applications in the J2EE server fails.

Specify the module settings in the metadata-complete attribute of the DD. Specify the J2EE server settings in the ejbserver.deploy.applications.metadata_complete property. The following table describes whether the annotations are referenced when the settings for the modules and J2EE servers are combined.

Table 12-11: Combining the settings for the J2EE servers and modules

Module version	Settings for modules		Settings for J2EE servers		
			Property is not specified	ejbserver.deploy.applications.metadata_complete property	
				false	true
EJB 3.0 or later	No DD		Y	Y	Y
	DD exists (metadata-complete attribute exists)	Omitted	Y	Y	Y
		false	Y	Y	Y
		true	--#	--#	--#
Servlet 2.5 or later	No DD		Y	Y	Y
	DD exists (metadata-complete attribute exists)	Omitted	Y	Y	Y
		false	Y	Y	Y
		true	N	N	N

Legend:

Y: Referenced

N: Not referenced

--: Not applicable

#: The application containing EJB 3.0 with `true` specified in the `metadata-complete` attribute of the DD cannot be imported and redeployed.

12.5.2 Timing for referencing annotations

The following table describes the timing for referencing annotations when the functionality for controlling the application references is used and when the functionality is not used. Note that the timing for referencing annotations is the same for the reference control functionality of both the J2EE servers and modules.

Table 12-12: Timing for referencing annotations

Timing (Executed command)	Reference control functionality settings	Application format	
		Archive format	Exploded archive format
Importing (<code>cjimportapp</code>)	Used	N	N
	Not used	Y	Y
Importing (<code>cjimportwar</code>)	Used	N	N
	Not used	Y	Y
Importing (<code>cjimportres</code> ^{#1})	Used	N	--
	Not used	Y	--
Starting (<code>cjstartapp</code> ^{#2})	Used	N	N
	Not used	Y ^{#3}	Y
Replacing (<code>cjreplaceapp</code>)	Used	N	--
	Not used	NC	--
Reloading (<code>cjreloadapp</code>)	Used	--	N
	Not used	--	Y
Obtaining attributes	Used	N	N

Timing (Executed command)	Reference control functionality settings	Application format	
		Archive format	Exploded archive format
(cjgetappprop)	Not used	N	N
Setting attributes (cjsetappprop)	Used	N	N
	Not used	N	N

Legend:

- Not used: Settings for not using the functionality for controlling the application references
- Used: Settings for using the functionality for controlling the application references
- Y: Annotations are referenced
- N: Annotations are not referenced
- NC: Annotations are referenced, but the changes are not applied
- : Not applicable

- #1: When the `cjimportres` command is executed by specifying the argument `-type ejb` or `-type war`.
- #2: Also includes the restarting of the J2EE server.
- #3: The annotation is loaded only when the settings differ from when the J2EE server was started previously.

Reference note

If the annotation information is changed after importing the exploded archive format application, the changed annotation information cannot be obtained even if the property file is obtained using the `cjgetappprop` command. This is because when the property file is obtained, the annotation information is not loaded from the class file.

To obtain the changed annotation information in a property file, execute a command, such as the `cjstartapp` command, for loading the annotation information from the class file and then obtain the property file.

12.5.3 Definitions in the DD (module settings)

Define the functionality for controlling the annotation references for the modules of Servlet 2.5 or later in `web.xml`. Note that for EJB 3.0 or later, you can only specify `false` in the `metadata-complete` attribute of the `<ejb-jar>` tag.

The following table describes the definition of the functionality for controlling the annotation references in the DD.

Table 12-13: Definition of the functionality for controlling the annotation references in the DD

Items	DD types	Specified tags and attributes	Settings
Servlet 2.5 or later	<code>web.xml</code>	<code>metadata-complete</code> attribute of the <code><web-app></code> tag	Specify <code>true</code> to control the annotation reference. Specify <code>false</code> to reference the annotations.
EJB 3.0 or later	<code>ejb-jar.xml</code>	<code>metadata-complete</code> attribute of the <code><ejb-jar></code> tag	Specify <code>false</code> to reference the annotations.

An example of settings is as follows:

Example of settings (for Servlet 2.5)

```

<web-app metadata-complete="true"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/
javaee/web-app_2_5.xsd" version="2.5">
  :
</web-app>

```

The text in bold is the setting for the functionality for controlling the annotation references.

If the `metadata-complete` attribute of the DD is not specified and if the DD is also omitted, the operation is the same as when `false` is specified in the `metadata-complete` attribute.

12.5.4 Changing the settings for the functionality for controlling the annotation references

To change the module settings (the value of the `metadata-complete` attribute of the DD), you must create an application in which the `metadata-complete` attribute is changed and then re-import the application into the J2EE server.

When the `metadata-complete` attribute is changed and if an attempt is made to replace the application using the redeploy functionality (the `cjreplaceapp` command specifying the `-replaceDD` option), an error occurs and the replace processing is cancelled.

12.6 Updating the contents defined in the annotations

This section describes how to update the contents defined in the annotations.

When an annotation is updated, you can apply the changes to the application by re-importing the application or by using the reload functionality. Also, the standard Java EE 5 or later specifications determine that the information defined in the annotations can be overwritten with the standard DD. With Application Server, you can overwrite the contents defined in the annotation using the standard DD and update the definition contents using the server management commands.

The following table describes the organization of this section.

Table 12-14: Organization of this section (Updating the contents defined in the annotations)

Category	Title	Reference location
Explanation	Updating the annotations	12.6.1
	Overwriting the annotations using the DD	12.6.2
	Referencing and updating the definitions with the server management commands	12.6.3

Note: The functionality-specific explanation is not available for "Implementation", "Settings" and "Operations".

12.6.1 Updating the annotations

If the contents defined in the annotations are updated, the updated contents can be applied to the application using one of the following methods:

Method of re-importing the application

In this method, the updated contents are applied by using normal replacement. Stop and then replace the application.

Method of using the reload functionality (for the exploded archive-format applications)

In this method, the contents of a deployed application are dynamically replaced by using the update detection or by executing the `cjreloadapp` command. You can use this method for the applications in the exploded archive format.

However, the annotations that can be updated with the reload functionality are limited. The following table lists the updatability of each annotation with the reload functionality.

Table 12-15: Updatability of each annotation with the reload functionality

Annotation name	Updatability
@PostConstruct	Y
@PreDestroy	Y
@Resource	N
@Resources	N
@RunAs	N
@DeclareRoles	Y
@RolesAllowed	N
@PermitAll	Y
@DenyAll	Y
@Stateless	N
@Stateful	N

Annotation name	Updatability
@Singleton	N
@DependsOn	Y
@Startup	Y
@AccessTimeout	Y
@Lock	Y
@ConcurrencyManagement	Y
@Init	Y
@Remove	Y
@AfterBegin	Y
@BeforeCompletion	Y
@AfterCompletion	Y
@Remote	N
@Local	N
@RemoteHome	N
@LocalHome	N
@LocalBean	N
@TransactionManagement	Y
@TransactionAttribute	Y
@PostActivate	Y
@PrePassivate	Y
@Interceptors	Y
@AroundInvoke	Y
@ExcludeDefaultInterceptors	Y
@ExcludeClassInterceptors	Y
@Timeout	N
@Schedule	N
@Schedules	N
@Asynchronous	Y
@ApplicationException	Y
@EJB	N
@EJBs	N
@WebService	Y
@WebServiceProvider	Y
@PersistenceContext	Y
@PersistenceContexts	Y
@PersistenceProperty	Y

Annotation name	Updatability
@PersistenceUnit	Y
@PersistenceUnits	Y
@WebEndpoint	N
@WebServiceRef	N

Legend:

Y: Can be updated

N: Cannot be updated

12.6.2 Overwriting the annotations using the DD

You can overwrite the information defined in the annotation with the DD. The annotations defined in the modules of Servlet 2.5 or later and EJB 3.0 or later can be overwritten by the DD.

The methods of overwriting the annotations are as follows:

Method of overwriting by editing the DD

In this method, you directly edit the DD.

Method of overwriting by using the server management commands to change the attributes

In this method, you use the `cjsetappprop` command or `cjsetresprop` command to change the attributes for the application after the application is imported into the J2EE server. However, the annotations that can be overwritten with this method have some conditions. For details, see *12.6.3 Referencing and updating the definitions with the server management commands*.

Reference note

You can specify the following definitions in the DD of EJB 3.0 or later that is imported into Application Server:

- `<display-name>` element
- `<interceptor-binding>` element, and the elements beneath this element (interceptor-related definition)
- `<application-exception>` element, and the elements beneath this element (application exception-related definition)

Important note

In the name attribute of the `@WebServlet` annotation, you cannot specify the same value as another `@WebServlet` annotation. Also, in the `filtername` attribute of the `@WebFilter` annotation, you cannot specify the same value as another `@WebFilter` annotation. If you specify the same value as another annotation, the following messages are output during overwriting in the DD and the relevant annotation is ignored.

Annotation	Attribute	Message
<code>@WebServlet</code>	name attribute	KDJE42397-W
<code>@WebFilter</code>	filtername attribute	KDJE42398-W

The overwriting method differs depending on whether the target DD elements and child elements can appear multiple times.

When the DD elements appear 0 times (do not appear) or only once

The annotation information is overwritten with the DD information.

When the DD elements appear 0 or more times or 1 or more times

The overwriting rules differ with the presence or absence of elements (*key*) associating the DD and annotations and the occurrence count of the child elements in the DD elements.

This subsection describes the rules for overwriting the annotations using the DD by separating them into the following 4 patterns.

Table 12-16: Pattern of rules for overwriting the annotations using the DD

Pattern	Occurrence count of the DD elements corresponding to the annotations	Presence of keys	Occurrence count of the DD child elements	Reference location
Pattern 1	0 times or once	When the key does not exist	--	(1)
Pattern 2	0 or more times or 1 or more times	When the key does not exist	--	(2)
Pattern 3		When the key exists	0 times or once	(3)
Pattern 4			0 or more times or 1 or more times	(4)

Legend:

--: There is no difference based on the presence or absence of the key

For the mapping between the annotation and DD elements, presence or absence of keys for each element, and the key elements, see the standard specifications.

The description of each pattern is as follows:

(1) Pattern 1 (when the element appears 0 times or once)

If the occurrence count of an element is 0 times or once, whether to overwrite the definition is determined by whether the element corresponding to the annotation is defined in the DD. The key value does not make a difference.

When the element corresponding to the annotation is defined in the DD

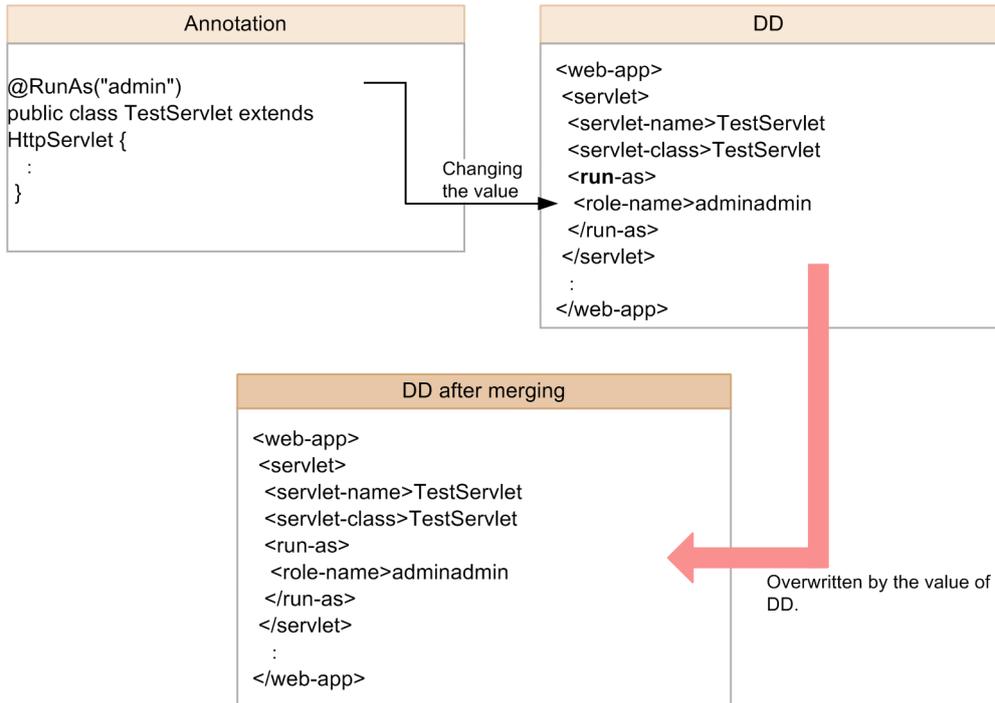
The annotation definition is overwritten by the DD definition.

When the element corresponding to the annotation is not defined in the DD

The annotation definition becomes valid.

The following figure shows an example of an element with occurrence count of 0 times or once. In this example, the key exists.

Figure 12-3: Example of an element that appears 0 times or once

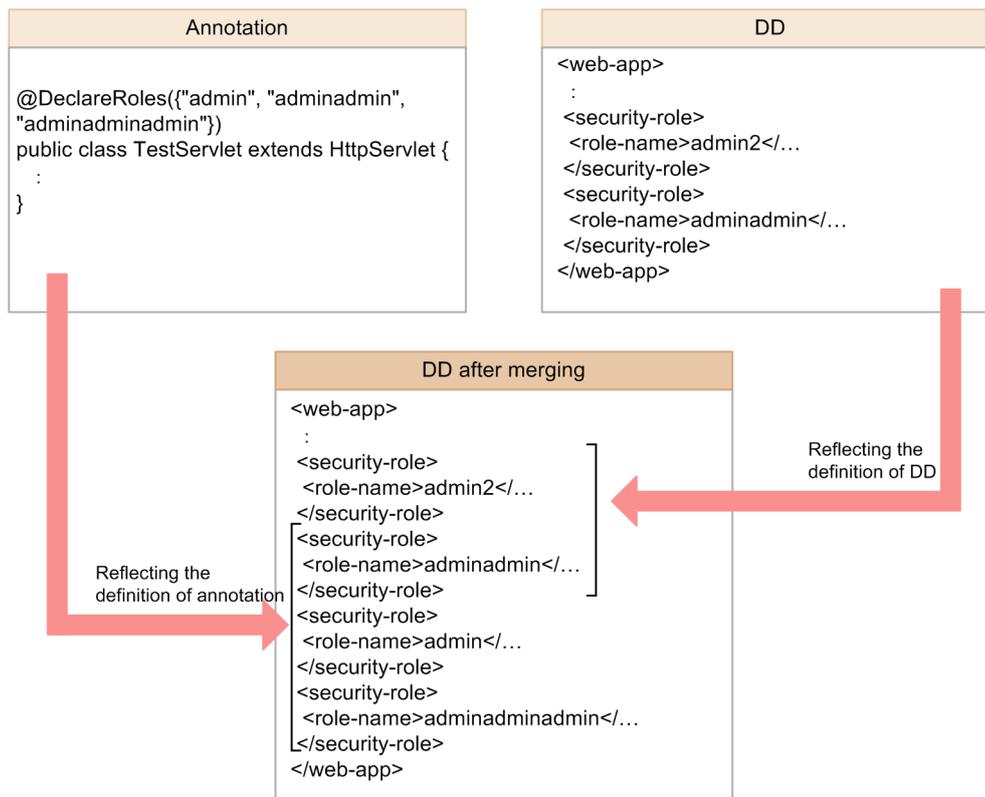


(2) Pattern 2 (when the key does not exist in an element that appears 0 or more times or 1 or more times)

The definitions of both the DD and annotations are valid.

The following figure shows an example when the key does not exist. In addition to the security role defined in @DeclareRoles, a new security role is defined in the DD.

Figure 12-4: Example when the key does not exist



In this example, three security roles, 'admin', 'adminadmin', and 'adminadminadmin', are defined in the annotation. In addition to these, 'admin2' defined in the DD is added and applied in the merged DD. Note that 'adminadmin' is a security role defined in both, annotation and the DD. The application is operated assuming the merged DD is defined.

Note that when this pattern is applicable, the annotation definition cannot be changed using the server management commands. Even if the information corresponding to the annotation is deleted or changed in the property file and the `cjsetappprop` command is executed, the information defined in the annotation is output to the property file obtained by using the `cjgetappprop` command thereafter.

(3) Pattern 3 (when the child element appears 0 times or once in the element that appears 0 or more times or 1 or more times)

When the child element within a DD element appears 0 times or once, the overwriting method differs on the basis of the key value.

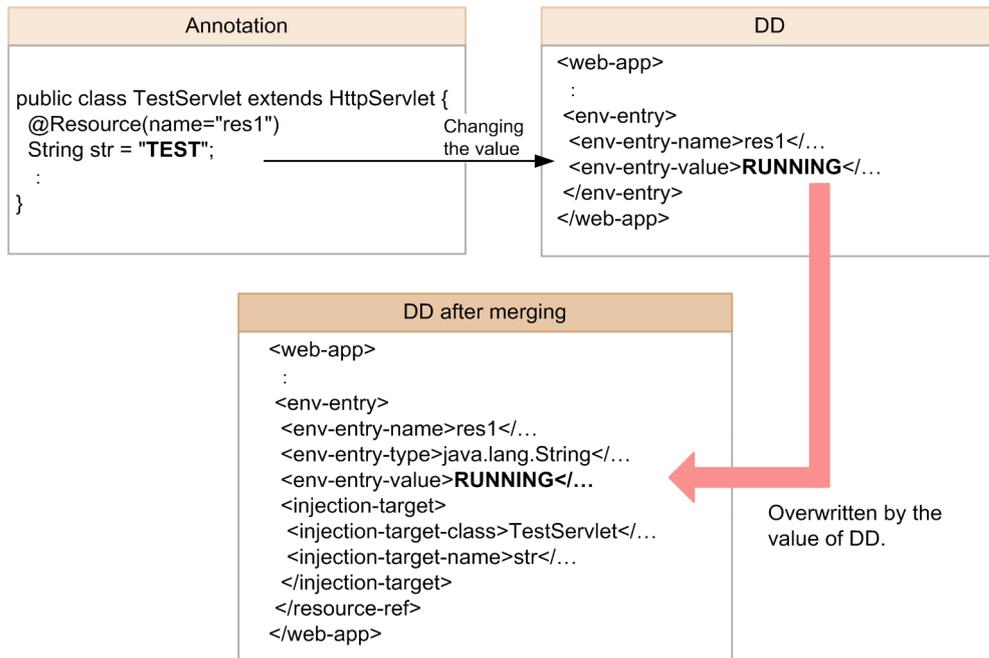
If the key value matches in the DD and annotation

The DD element overwrites the annotation definition.

If the key value does not match in the DD and annotation

The annotation is not overwritten. Both the DD and annotation definitions become valid.

Figure 12-5: Example when the child element appears 0 times or once



In this example, the value of the environment entry 'res1' is changed from 'TEST' defined in the annotation to 'RUNNING' defined in the DD. The application is operated assuming that the merged DD is defined.

(4) Pattern 4 (when the child element appears 0 or more times or 1 or more times in an element that appears 0 or more times or 1 or more times)

When the child element within a DD element appears 0 or more times or 1 or more times, the overwriting method differs on the basis of the key value.

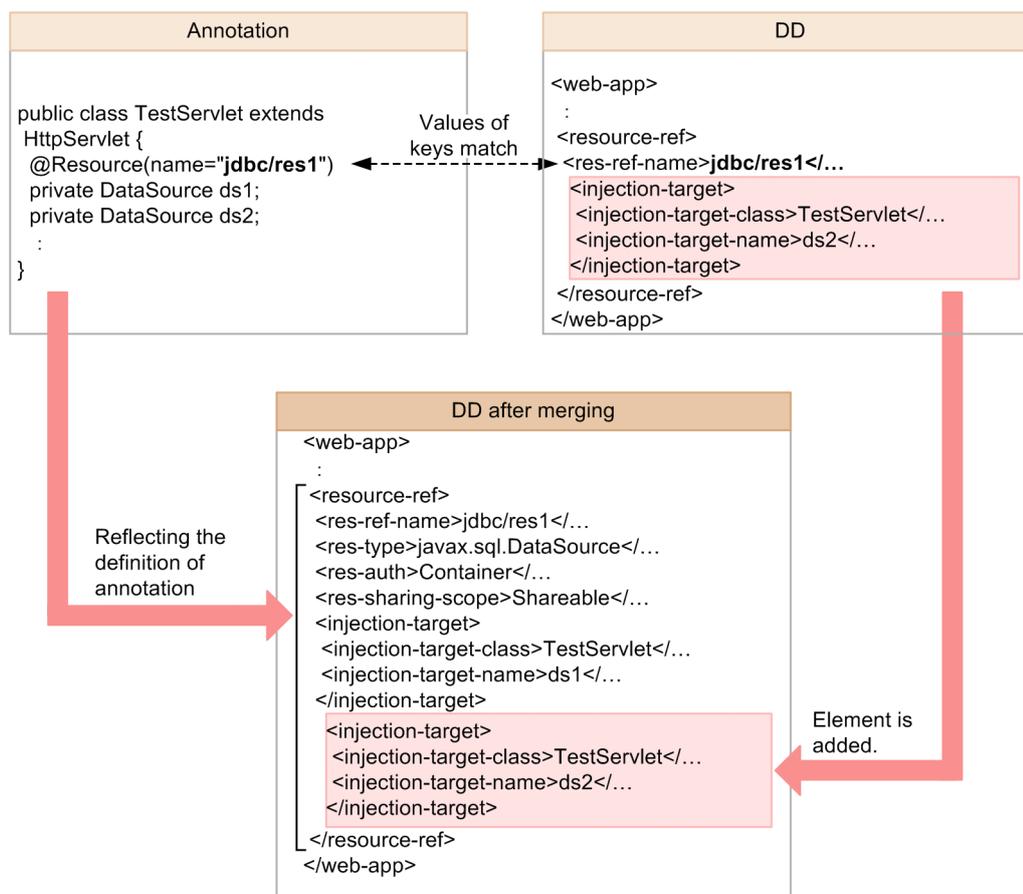
If the key value matches in the DD and annotation

The DD element is added to the annotation definition.

If the key value does not match in the DD and annotation

The annotation is not overwritten. Both the DD and annotation definitions become valid.

Figure 12-6: Example when the child element appears 0 or more times or 1 or more times



In this example, the element 'injection-target' defined in the DD is added to the annotation definition. The application is operated assuming that the merged DD is defined.

12.6.3 Referencing and updating the definitions with the server management commands

The elements defined in the annotation can be referenced by using the server management commands the `cjgetappprop` command or the `cjgetresprop` command. Also, some of the definitions can be updated by using the `cjsetappprop` command or the `cjsetresprop` command.

(1) Operations for referencing or updating the elements defined in the annotations

The following table describes the operations when the server management commands are used to reference or update the elements defined in the annotations. The operations differ when a DD exists and when a DD does not exist.

Table 12-17: Operations when the server management commands are used to reference or update the elements defined in the annotations

Element types		Reference using <code>cjgetappprop</code> or <code>cjgetresprop</code>	Update using <code>cjsetappprop</code> or <code>cjsetresprop</code>
When a DD exists	Elements defined in the DD	The contents defined in the DD are output.	The contents of the DD are updated.
	Elements defined in the annotation	The contents defined in the annotation are output.	Cannot be updated.

Element types		Reference using <code>cjsetappprop</code> or <code>cjsetresprop</code>	Update using <code>cjsetappprop</code> or <code>cjsetresprop</code>
When a DD exists	Elements in which the annotation definition is overwritten with the DD	The result of merging the definitions of the annotation and DD are output.	See (2) <i>Updating the definitions by using the server management commands.</i>
When a DD does not exist	Elements defined in the annotation	The contents defined in the annotation are output.	Cannot be updated.

(2) Updating the definitions by using the server management commands

Whether the attributes defined in the annotation by using the `cjsetappprop` command or the `cjsetresprop` command can be overwritten and updated depends on where the information to be updated is defined.

The following table describes the mapping between the definition location and overwritability.

Table 12-18: Mapping between the definition location and overwritability

Definition location	Overwritability
Elements only defined in the DD	Can be overwritten
Elements only defined in the annotation	Cannot be overwritten
Elements in which the annotation definition is overwritten by the DD	Can be overwritten #

#: When the application is exported after update, the contents defined in the annotation and the DD are merged and output to the DD (`web.xml`) in the WAR file included in the exported application. For the applications in the exploded archive format, the contents defined in the annotation and the DD are merged and output to `WAR-directory/WEB-INF/web.xml`.

■ Notes

When you use the `cjsetappprop` command or the `cjsetresprop` command to overwrite and update the elements in which the annotation definition is overwritten by the DD, the DD and the annotation information is merged and output to the DD in the application directory. In this case, when you subsequently change the annotation in the class file in order to apply the contents defined in the annotation to the DD, the definition remaining in the DD is given priority and the changed annotation definition is not enabled.

In such cases, delete the annotation definition applied to the DD and re-import the application to enable the annotation definition. Also, when you add or change the definitions in the property file, you need not re-import the application.

12.7 Notes on using annotations

This section describes the notes on using the annotations.

(1) Notes on importing

- If an annotation declared in a class references a class of an external archive file, the annotation cannot be imported individually.
 - EJB (`ejb-jar.xml` version 2.1 or later or no `ejb-jar.xml`)
 - Servlet (`web.xml` version 2.4 or later or no `web.xml`)
 - Library JAR

Implement the external reference using one of the following methods:

- Include all the classes to be referenced in the EAR and import.
- Include all the classes to be referenced in the application directory and import.
- Set the JAR files that include the classes performing external reference as the container extension library and also in the class path of the process to be imported.

The examples of settings for each of these operation methods are as follows:

Example 1: When the operations are performed using the server management commands

Specify the referencing JAR file in the key (`add.class.path`) of the option definition file for J2EE servers. Furthermore, specify the referencing JAR file in the key (`USRCONF_JVM_CLASSPATH`) of the option definition file for the server management commands.

Example 2: When operations are performed using Management Server

Specify "`add.class.path=referencing-JAR-file`" in the extension parameter in the J2EE container settings of the logical J2EE server by using the management portal.

Also, specify the referencing JAR file in the `add.class.path` key of `adminagentuser.cfg`. Furthermore, specify the referencing JAR file in the key (`USRCONF_JVM_CLASSPATH`) of the option definition file for the server management commands.

- In the following cases, an error occurs during the execution of the server management commands:
 - When Java Type not supported with Application Server is set in `elementType()` of `@Resource`.
 - When `ejb-jar.xml` does not exist in the EJB-JAR file and when a class with `@Stateless` or `@Stateful` annotations declared does not exist.
 - When annotations are declared exceeding the occurrence count specified in the DD schema definition.
- When you reference an EJB-JAR file from a WAR file, include both WAR file and EJB-JAR file and then import the application. If, however, the referenced interface is included in the WAR file, you can also import the WAR file individually.
- When a class in the library JAR is coded in an annotation attribute, import the application including the library JAR.
- If there is an EJB-JAR that references another EJB-JAR, import the application including both the EJB-JARs. If, however, the referencing EJB-JAR includes an interface of the referenced EJB-JAR, you can import the application for each EJB-JAR separately.

(2) Notes on specifying the mapped-name attribute of `@Resource`

With Application Server, only the `mapped-name` attribute specified in `@Resource` is processed. The `mapped-name` attribute specified in `@Resource` is processed as an attribute corresponding to the `<linked-to>` tag beneath the `<resource-ref>` tag, `<linked-to>` tag beneath the `<resource-env-ref>` tag, and `<linked-queue>` tag beneath the `<resource-env-ref>` tag in the property file. However, if both, the `<linked-to>` tag and `<linked-queue>` tag of the property file and the `mapped-name` attribute are specified, the value specified in the `<linked-to>` tag and `<linked-queue>` tag is given priority.

(3) Notes on the resolution of the EJB links

- `@EJB` is an annotation corresponding to the `<ejb-ref>` tag of the DD. The EJB link is resolved within the scope of the same application. If the `beanName` attribute is specified in `@EJB`, search the EJB with the corresponding EJB name to resolve the link. The EJB name indicates the `<ejb-name>` tag beneath the `<session>` tag, `<entity>` tag, and `<message-driven>` tag in the case of `ejb-jar.xml`. Also, in an annotation, the EJB name indicates the name specified in the `name` attribute of `@Stateless` and `@Stateful`.
- If the `beanName` attribute is not specified in `@EJB`, search the EJB of the type suitable to the `beanInterface` attribute of `@EJB` to resolve the link. The EJB types suitable to the `beanInterface` attribute of `@EJB` are as follows:
 - EJB with the home interface of the same class type as that specified in the `beanInterface` attribute of `@EJB`
 - EJB with the business interface of the same class type as that specified in the `beanInterface` attribute of `@EJB`

With the cases where one business interface is implemented by multiple EJBs, multiple EJBs might be suitable for `@EJB` and `@EJBs`. If the reference destinations cannot be narrowed down to one, the link is not resolved.

(4) Notes on using `@RemoteHome` or `@LocalHome`

With Application Server, the interface that forms the return value type of the `create` method of the home interface specified in `@RemoteHome` or `@LocalHome` is considered as the component interface. `@RemoteHome` and `@LocalHome` cannot be used with a combination of the home interface and business interface.

(5) Notes on using `@PostConstruct` or `@PreDestroy`

- When the Web application starts, a check is performed to confirm whether the definition of `@PostConstruct` or `@PreDestroy` is in accordance with the Java EE specifications. If the definitions are not in accordance with the Java EE specifications, an annotation-related message is output and the starting of the Web application is cancelled.
- If `<post-construct>` or `<pre-destroy>` tags are specified in `web.xml`, the method is checked when the Web application starts. If the specified class or method is not found, the message KDJE39332-E is output and an error occurs.
- If `@PostConstruct` or `@PreDestroy` is specified in two or more methods in one class, the message KDJE39327-E is output and an error occurs when the Web application starts.
- With Application Server, when you define a method specifying `@PostConstruct` or `@PreDestroy`, note the specifications described in the following table. The following table describes the specifications for defining the methods specifying `@PostConstruct` or `@PreDestroy`.

Table 12-19: Specifications for defining the methods specifying `@PostConstruct` or `@PreDestroy`

Specifications for method definition	Differences from the Java EE specifications	Explanation
Argument cannot be specified.	--	For a method with argument, the message KDJE39328-E is output and an error occurs when the Web application starts.
Checked exception can be thrown. Handled like a non-checked exception.	With the Java EE specifications, a checked exception cannot be thrown.	For a method containing the <code>throws</code> definition, the message KDJE39329-W is output when the Web application starts.
Can be used in the instance method and class (<code>static</code>) method.	With the Java EE specifications, the definition cannot be specified with the <code>static</code> method.	For the <code>static</code> method, the message KDJE39330-W is output when the Web application starts.
A method with a return value can be used.	With the Java EE specifications, the return	For a method with a return value, the message KDJE39331-W is output when the Web application starts.

Specifications for method definition	Differences from the Java EE specifications	Explanation
A method with a return value can be used.	value must be limited to <code>void</code> .	For a method with a return value, the message KDJE39331-W is output when the Web application starts.
Either <code>public</code> , <code>protected</code> , <code>package private</code> , or <code>private</code> can be specified.	--	--
Can be declared as <code>final</code> .	--	--

Legend:

--: None

- If an exception occurs in the method specifying the `@PostConstruct` annotation, the message KDJE53906-E is output and the instance where the exception occurred is destroyed.
- When a class is initialized after the Web application starts, even if the initialization fails, initialization is tried every time the class is used. For example, during the first request to the servlet, if an exception occurs in the method specifying `@PostConstruct`, the initialization of the servlet is tried even when the same servlet is accessed next time.
- If an exception occurs in the method specifying the `@PreDestroy` annotation, the message KDJE53907-W is output and the process of destroying the object from the Web container continues.

(6) Other notes

- If `@Timeout` is specified in multiple EJB methods, the target method cannot be identified.
- If the annotation information of the class specified as an interceptor cannot be obtained, an error occurs during the property file settings. To obtain the annotation information, you must specify the class path such that the information about the target class, its superclass or interface, and all the fields or methods declared on those classes can be acquired.

13

Formats and Deployment of J2EE Applications

This chapter describes the J2EE application formats that can be executed with the J2EE servers and the J2EE application deploy, un-deploy, and replace functionality that can be executed for each format.

13.1 Organization of this chapter

The following table lists the J2EE application formats, the deployment functionality, and the reference locations.

Table 13-1: J2EE application formats, deployment functionality, and reference locations

Functionality	Reference location
Executable J2EE application formats	<i>13.2</i>
Archive-format J2EE applications	<i>13.3</i>
Exploded archive-format J2EE applications	<i>13.4</i>
Deploying and un-deploying J2EE applications	<i>13.5</i>
Replacing J2EE applications	<i>13.6</i>
Redeploying J2EE applications	<i>13.7</i>
Detecting updates and reloading the J2EE applications	<i>13.8</i>
WAR applications	<i>13.9</i>

13.2 Executable J2EE application formats

The applications that can be executed on J2EE servers are called *J2EE applications*. To execute the WAR format Web applications or the EJB-JAR format EJB applications on a J2EE server, you must deploy the applications as J2EE applications. The two types of J2EE applications that can be executed on J2EE servers are as follows:

- **Archive-format J2EE applications**

An archive-format J2EE application is a J2EE application that holds the application entities, such as the EJBs and servlets, in the working directory of the J2EE server.

- **Exploded archive-format J2EE applications**

An exploded archive-format J2EE application is a J2EE application that holds the application entities, such as the EJBs and servlets, in files or directories that are outside the J2EE server and follow fixed rules.

Note that you can specify and import WAR files or WAR directories with a J2EE server. The imported application is called a *WAR application*, and can be deployed as a J2EE application. For details on the WAR applications, see *13.9 WAR applications*.

The section *13.3* and subsequent sections describe the deployment of J2EE applications in each format. As the execution functionality of J2EE applications, the sections also describe the redeployment, update detection, and reloading of J2EE applications.

Important note

In UNIX, if different values are specified for the `LANG` environment variable in the development and execution environments of the J2EE application, the `java.text.ParseException:Unparseable date` exception might be thrown when the J2EE server starts. However, the subsequent J2EE server operations are not affected.

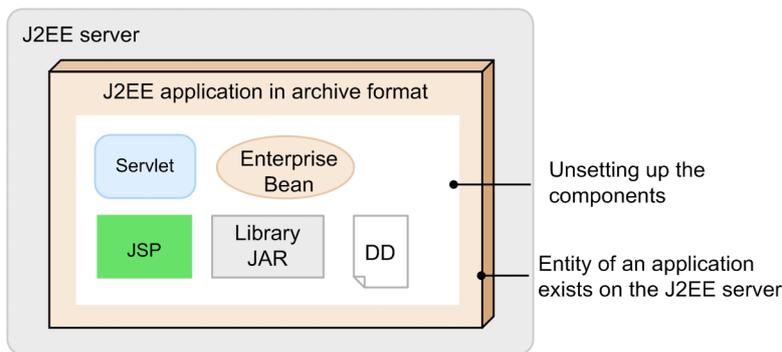
For details on the functionality to be used during the operations of J2EE applications after the system operations are started, see *5. Operations of J2EE Applications* in the *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*.

13.3 Archive-format J2EE applications

This section describes the deployment of J2EE applications using the archive format.

An *archive-format J2EE application* is one of the application formats that can be executed on a J2EE server. An archive-format J2EE application holds the application entities, such as the EJBs and servlets, in the working directory of the J2EE server. The following figure shows an archive-format J2EE application.

Figure 13-1: Archive-format J2EE application



With the archive-format J2EE applications, the application entity exists on the J2EE server and the components of the application are stored in the working directory of the J2EE server.

Normally, when you replace an archive-format J2EE application, operations, such as stopping the J2EE application that is running on the J2EE server, importing and deploying a new J2EE application, are performed. At this time, if you use the redeploy functionality, you can replace an archive-format J2EE application using fewer steps as compared to replacing a normal J2EE application.

When you replace a J2EE application using the redeploy functionality, the attribute information of the J2EE application can be inherited and the operations, such as stopping a J2EE application that is running on the J2EE server, importing and redeploying a new J2EE application, are no longer required. For details on the redeploy functionality, see *13.7 Redeploying J2EE applications*.

! Important note

If the application version is Java EE 5 or later, you cannot specify a level higher than the root of the application package for the entry name in the EAR file. If a higher level is specified, the KDJE42387-E message is displayed when the J2EE application is imported and an error occurs.

13.4 Exploded archive-format J2EE applications

This chapter describes the deployment of J2EE applications using the exploded archive format.

The following table describes the organization of this section.

Table 13-2: Organization of this section (Exploded archive-format J2EE applications)

Category	Title	Reference location
Explanation	Overview of the exploded archive format	13.4.1
	Configuration of the application directory	13.4.2
Settings	Settings for using the exploded archive-format J2EE applications (changing the security settings)	13.4.3
Notes	Notes on using the exploded archive format	13.4.4

Note: The functionality-specific explanation is not available for "Implementation" and "Operations".

13.4.1 Overview of the exploded archive format

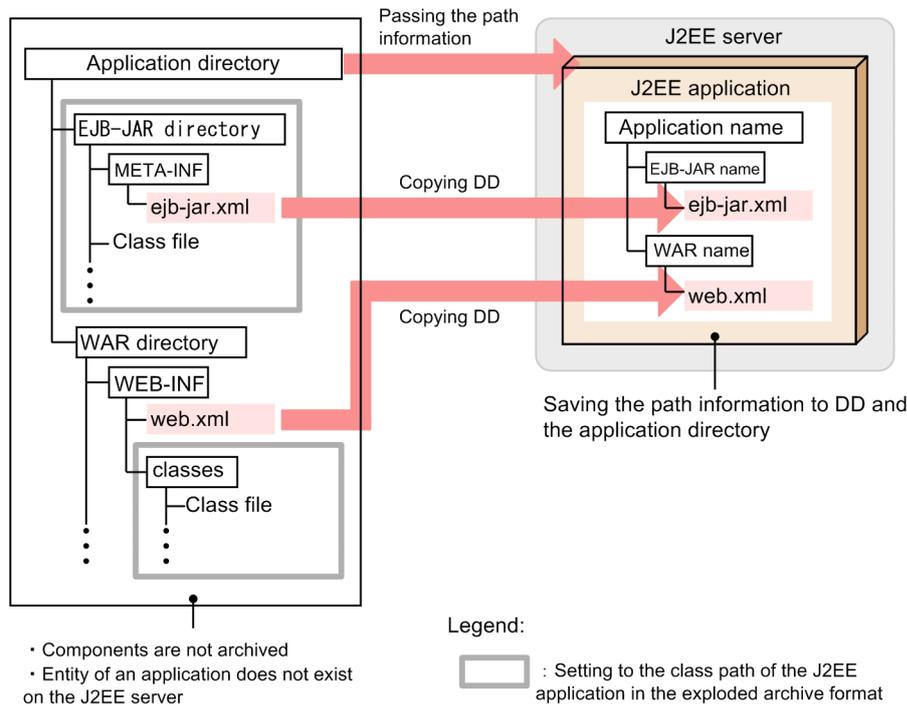
This chapter provides an overview of the exploded archive format and describes how to deploy and un-deploy the exploded archive-format J2EE applications.

(1) Exploded archive-format J2EE applications

An *exploded archive-format J2EE application* is one of the application formats that can be executed on a J2EE server. An exploded archive-format J2EE application holds the application entities, such as the EJBs and servlets, in files or directories that are outside the J2EE server and follow fixed rules. If you create a root directory called an *application directory* and store the components, such as the EJB-JAR and Web applications, in the application directory, you can operate the application as a J2EE application. For details on the application directory, see *13.4.2 Configuration of the application directory*.

The following figure shows an exploded archive-format J2EE application.

Figure 13-2: Exploded archive-format J2EE application



An exploded archive-format J2EE application stores the path information to the DD and the application directory. The components, such as the EJB-JAR or Web applications, are not stored in the working directory of the J2EE server.

This subsection describes the features of an exploded archive-format J2EE application.

(a) Simplifying the replacement of a J2EE application

With an exploded archive-format J2EE application, you need not archive the components in the EAR format. Therefore, you can replace an exploded archive-format J2EE application using fewer steps as compared to an archive-format J2EE application.

Moreover, by using the reload functionality, you dynamically replace a J2EE application using even fewer steps. The reload functionality detects the updates of the files configuring the J2EE application and reloads the updated J2EE application. By using the reload functionality, you dynamically replace the deployed servlets, JSPs, and EJB-JARs without restarting the J2EE server. For details on the reload functionality, see *13.8 Detecting updates and reloading J2EE applications*.

(b) Sharing the application directory across multiple J2EE servers

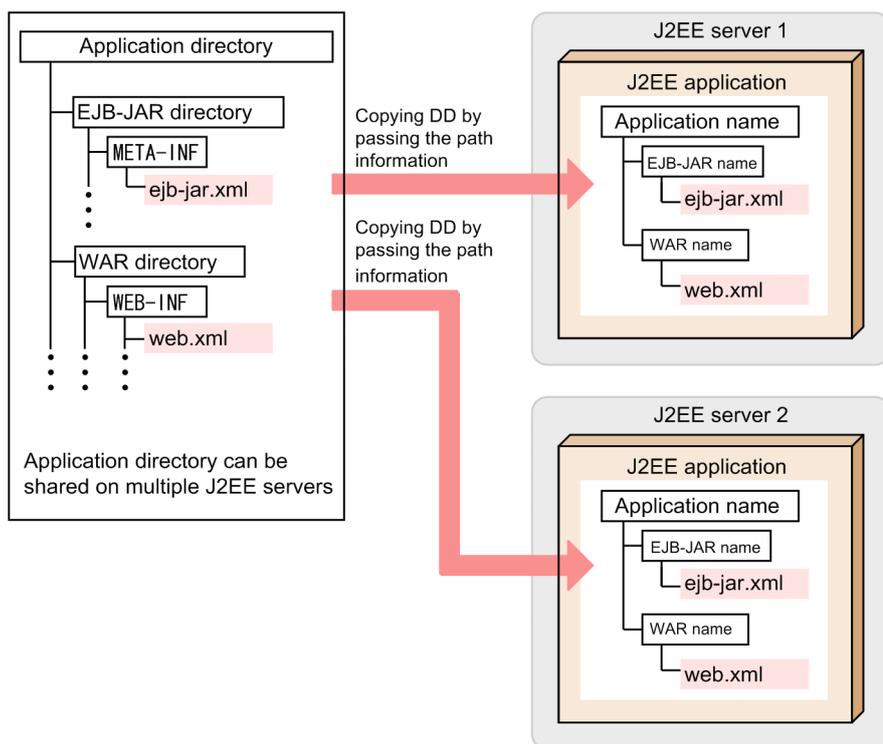
With the exploded archive-format J2EE applications, multiple J2EE servers can share the application directory.

For example, when you use an archive-format J2EE application, if you want to execute the J2EE applications with the same contents on multiple J2EE servers, you must allocate an EAR file or a ZIP file on multiple J2EE servers respectively. Also, to replace an imported J2EE application, you must re-import and redeploy the J2EE application for all those servers.

However, with the exploded archive-format J2EE applications, only the path information to the DD and the application directory is stored, so multiple J2EE servers can share the components, such as the EJB-JAR or Web applications, existing in the application directory.

For example, when J2EE servers are arranged in a cluster configuration and the same J2EE application is to be used on each J2EE server, you can specify the same directory as the application directory. The same application directory is referenced, so if any change occurs in the J2EE application, you can replace all J2EE applications by merely updating the files, such as the class files, in the application directory. The following figure shows an example of sharing an application directory.

Figure 13-3: Example of sharing an application directory



Note that when a J2EE application is shared, place the application directory in an environment (such as a shared disk device) that can be referenced from each J2EE server.

13.4.2 Configuration of the application directory

The application directory is the root directory of an exploded archive-format J2EE application. When you create an exploded archive-format J2EE application, you create the application directory and store the components, such as the EJB-JAR or Web applications, in the application directory. The following table describes the configuration of the application directory.

Table 13-3: Configuration of the application directory

Directory	Explanation of the directory
<i>Application-directory</i>	This is the root directory of the exploded archive-format J2EE applications. You can enter a name of your choice for the application directory.
META-INF	This directory stores the DD and <code>cosminexus.xml</code> of J2EE applications. The files <code>application.xml</code> and <code>cosminexus.xml</code> are stored directly beneath this directory.
<i>EJB-JAR-directory</i>	This is the root directory for EJB applications.
META-INF	This directory stores the DD of the EJB-JARs. The file <code>ejb-jar.xml</code> is stored directly beneath this directory.
<i>Package-name</i>	This directory stores the class files and property files of the EJB-JARs. Example: <code>MyEJB.class</code> <code>MyEJBHome.class</code> <code>MyEJBRemote.class</code> <code>MyResource.properties</code>
<i>WAR-directory</i>	This is the root directory for Web applications.

Directory		Explanation of the directory
<i>WAR-directory</i>		Note that the JSP files are stored directly beneath this directory. Example: index.jsp
	WEB-INF	This directory stores the DD for Web applications. The file web.xml is stored directly beneath this directory.
	classes	This directory stores the servlet class files and property files. Example: MyServlet.class
	lib	This directory stores the JAR files, such as the tag library. Example: MyTagLibrary.jar
RAR file #		This is the resource adapter used in a J2EE application.
<i>Library-directory</i>		This directory stores the library JAR. If the application version is Java EE 5 or later, the library JAR is stored directly beneath this directory.

#: Allocated when the resource adapter (RAR file) is included and used in a J2EE application. The RAR file is stored as an archive file. When the resource adapter is deployed as a J2EE resource adapter, the resource adapter need not be stored in the application directory.

Note the following points about the application directory:

- The points to remember when you allocate the *EJB-JAR-directory*, *WAR-directory*, and RAR file are described separately for cases when application.xml exists and when application.xml does not exist:
 - When application.xml exists, create the directories according to the relative path coded under the <module> tag of application.xml. Also, match the names of the EJB-JAR directory and WAR directory with the coding in application.xml. The string with the extension (.jar or .war) removed from the relative path (EJB-JAR file name or WAR file name) on the EAR coded under the <module> tag of application.xml forms the name of the EJB-JAR directory and WAR directory.
 - When application.xml does not exist, the end of the directory name is _jar or _war.
- You cannot code a relative path containing (. ./) in the values of the <module>/<ejb>, <module>/<web>, and <module>/<connector> tags of application.xml. If such a relative path is coded, an error occurs.
- If the EJB-JAR and WAR modules are declared with extensions other than .jar or .war under the <module> tag of application.xml, the application cannot be imported.
- The method of handling the library JAR differs according to the application version.
 - When the application version is J2EE 1.4 or earlier
Of the JAR files in the application directory, the JAR files not defined under the <module> tag of application.xml and with a lower case extension (.jar) are handled as library JARs.
 - When the application version is Java EE 5 or later
The JAR files placed directly beneath the library directory not containing sub-directories or directly beneath the application directory and not defined in the <module> tag of application.xml are handled as library JAR. You cannot code a path containing (. ./) in the library directory. If such a path is coded, an error occurs when the application is imported.
- To specify a JAR that is not defined in the application directory, use the reference library.
- To change the application directory, use the rules specified in (2) *Changing the application directory*.

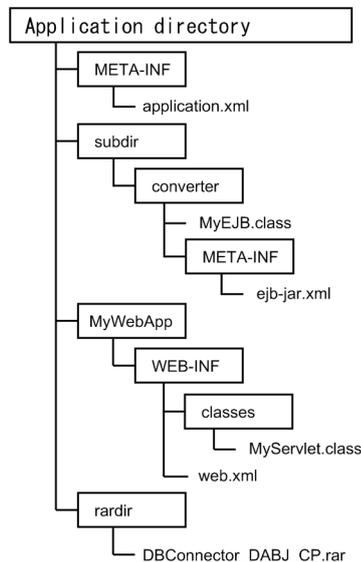
(1) Example of creating an application directory

The following is an example of creating an application directory.

This point describes an example configuration of an application directory when the EJB-JAR directory is `subdir/converter`, WAR directory is `MyWebApp`, and RAR file is `rardir/DBConnector_DABJ_CP.rar`, and an example of coding `application.xml`.

- **Example configuration of an application directory**

An example configuration of an application directory is as follows.



- **Example of coding application.xml**

`application.xml` in this example is as follows:

```

<?xml version="1.0" encoding="Shift_JIS" ?>
<application xmlns="http://java.sun.com/xml/ns/j2ee"
  version="1.4"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
  http://java.sun.com/xml/ns/j2ee/application_1_4.xsd">
  <display-name>converter</display-name>
  <module>
  <ejb>subdir/converter.jar</ejb>
  </module>
  <module>
  <web>
  <web-uri>MyWebApp.war</web-uri>
  <context-root></context-root>
  </web>
  </module>
  <module>
  <connector>rardir/DBConnector_DABJ_CP.rar</connector>
  </module>
</application>

```

(2) Changing the application directory

When you change the application directory, sometimes the application directory cannot be changed or you must re-import the application after the application directory is changed, depending on the type of the target directory or file, or the operations performed. The operations required for changing the application directory are as follows.

Example:

- **Deleting or renaming the directory**

You cannot delete the application directory, EJB-JAR directory, and WAR directory. Also, you cannot change the directory name. If you delete or rename the directory, you must re-import the application.

- **Adding, changing, or deleting JAR files**

You cannot add, delete, or rename the library JAR. Even if you add a library JAR (JAR file that is not declared in `application.xml`), that JAR file is not read. If you rename the library JAR existing when you imported the application or if you delete the library JAR, you must re-import the application.

You can add, change, or delete a JAR file with the WAR directory/`WEB-INF/lib` directory.

• **Changing the DD**

Even if the user changes the DD existing in the application directory, the J2EE server does not detect the change. To change the DD definition, change the definition in the development environment and then re-import the application or use the server management commands. You can, however, apply the updates for annotations or `cosminexus.xml` coded in the class file when the class file is re-read when you start the J2EE application the next time.

Note that the configuration of the application directory is checked when the server management commands are executed.

The following table describes whether the application directory can be changed.

Table 13-4: Changing the application directory

Target	Directory and file	Operations for the directory and file	Application status		Explanation
			Stop	Start	
J2EE application	Application directory	Add	--	--	--
		Change contents	--	--	--
		Rename	R	N	You cannot change the configuration of the application.
		Delete	R	N	You cannot change the configuration of the application.
	application.xml	Add	--	--	--
		Change contents	Y	N	You cannot compile the DD directly. Use the server management commands to update the DD definition. ^{#1}
		Rename	--	--	--
		Delete	R	N	--
	cosminexus.xml	Add	NR	Y	--
		Change contents	NR	Y	--
		Rename	--	--	--
		Delete	NR	Y	--
EJB application	EJB-JAR directory	Add	R	--	--
		Change contents	--	--	--
		Rename	R	N	You cannot change the configuration of the application.
		Delete	R	N	You cannot change the configuration of the application.
EJB application	Class files configuring EJB	Add	NR	N	After adding a class, you must edit the attributes with the server management commands.
		Change contents	NR	N	When the method information is coded in the DD (<code>container-</code>

Target	Directory and file	Operations for the directory and file	Application status		Explanation
			Stop	Start	
EJB application	Class files configuring EJB	Change contents	NR	N	transaction and method-permission), you must perform the modifications in sync with the DD information.
		Rename	R	N	If you try to change the class file name without changing the class file contents, an error occurs during the next class loader.
		Delete	NR	N	After deleting a class, you must edit the attributes with the server management commands.
	Other class files	Add	NR	N	--
		Change contents	NR	N	--
		Rename	R	N	If you try to change the class file name without changing the class file contents, an error occurs during the next class loader.
		Delete	NR	N	--
	EJB application	ejb-jar.xml	Add	--	--
Change contents			Y	N	You cannot compile the DD directly. Use the server management commands to update the DD definition.#1
Rename			--	--	--
Delete			R	N	--
Web application	WAR directory	Add	R	--	--
		Change contents	--	--	--
		Rename	R	N	You cannot change the configuration of the application.
		Delete	R	N	You cannot change the configuration of the application.
	Class files and property files under WEB-INF/classes	Add	NR	N	--
		Change contents	NR	N	--
		Rename	--	--	If you try to change the class file name without changing the class file contents, an error occurs during the next class loader.
		Delete	NR	N	--
	JAR files under WEB-INF/lib	Add	NR	N	--
		Change contents	NR	N	--

Target	Directory and file	Operations for the directory and file	Application status		Explanation
			Stop	Start	
Web application	JAR files under WEB-INF/lib	Rename	NR	N	--
		Delete	NR	N	--
	web.xml	Add	--	--	--
		Change contents	Y	N	You cannot compile the DD directly. Use the server management commands to update the DD definition. ^{#1}
		Rename	--	--	--
		Delete	R	N	--
Web application	JSP files, Tag files, Static content (such as HTML and JavaScript), files on which the JSP files or tag files depend ^{#2} , JSP compilation results	Add	NR	N	--
		Change contents	NR	N	--
		Rename	--	--	--
		Delete	NR	N	--
Library	Library JAR	Add	R	N	You cannot change the configuration of the application. The library JAR is ignored even if added.
		Change contents	NR	N	--
		Rename	R	N	You cannot change the configuration of the application.
		Delete	R	N	You cannot change the configuration of the application.
	Reference library	Add	NR	N	Change the definition using the server management command (cjsetappprop).
		Change contents	NR	N	--
		Rename	--	--	Change the definition using the server management command (cjsetappprop). You cannot rename while the reference library is open.
		Delete	Y	N	Change the definition using the server management command (cjsetappprop). You cannot delete the definition while the reference library is open.

Target	Directory and file	Operations for the directory and file	Application status		Explanation
			Stop	Start	
Library	Directory of reference libraries	Add	NR	N	--
		Change contents	NR	N	--
		Rename	NR	N	--
		Delete	NR	N	--
Resource adapter	RAR file	Add	R	N	--
		Change contents	Y	N	You cannot change the RAR file directly. Use the server management commands to change the properties.
		Rename	R	N	--
		Delete	R	N	--

Legend:

When the 'Application status' is 'Stop'

If the application is not running, the need for re-importing the application differs according to the types of the directories and files and the operation contents.

NR: Application need not be re-imported (the operation will be applied when the application is started the next time).

Y: If the conditions are satisfied, the application need not be re-imported. For conditions, see the *Explanation* column.

R: Application must be re-imported.

--: Not applicable.

When the 'Application status' is 'Start'

If the application is running and the reload functionality is disabled, the operating permissions differ according to the types of the directories and files and the operation contents.

Y: Operations can be performed.

N: Operations cannot be performed.

--: Not applicable.

#1

Even if you change the DD, the J2EE server does not detect the update and the operation is ignored.

The file is overwritten when the following commands are executed next time:

```

cjsetappprop
cjaddapp (when the -type filter option is specified)
cjdeleteapp (when the -type filter option is specified)
cjrenameapp
cjstartapp

```

#2

The dependent files are the files that are included in the `include` directive of the JSP files or tag files, and the files that are included in `<include-prelude>` or `<include-coda>` of `web.xml`.

13.4.3 Settings for using the exploded archive-format J2EE applications (changing the security settings)

This subsection describes the settings required for using the exploded archive-format J2EE applications.

To use the exploded archive-format J2EE applications, we recommend that you change the security settings using one of the following methods:

- Releasing SecurityManager
- Changing the security policy settings

A description of the methods is as follows.

(1) To release SecurityManager

When you start the J2EE server, specify the `-nosecurity` option in the `cjstartsv` command to release SecurityManager.

```
# cjstartsv server-name -nosecurity
```

! Important note

By releasing SecurityManager, you can reduce the overhead of checking the permissions when the resources are accessed, but the security is lowered because the J2EE application can now access any resource.

(2) To change the security policy settings

Edit `server.policy` as follows in order to grant permissions to access resources in the JAR files and class files under the application directory:

```
grant codeBase "file:/D:/MyApplicationDir/Web.war/-"{
  permission java.lang.RuntimePermission "loadLibrary.*";
  permission java.lang.RuntimePermission "queuePrintJob";
  permission java.net.SocketPermission "*", "connect";
  permission java.io.FilePermission "<<ALL FILES>>", "read, write";
  permission java.util.PropertyPermission "*", "read";
  permission javax.security.auth.AuthPermission "getSubject";
  permission javax.security.auth.AuthPermission "createLoginContext.*";
};
```

Specify `server.policy` after you set up the system using the Smart Composer functionality commands.

13.4.4 Notes on using the exploded archive format

This subsection describes the notes on using the exploded archive format.

- With the exploded archive-format J2EE applications, you cannot perform the following operations as compared to the archive-format J2EE applications:
 - Adding EJB-JARs/ WARs/ RARs
 - Deleting EJB-JARs/ WARs/ RARs
 - Adding library JAR
 - Deleting library JAR

To perform these operations on an exploded archive-format J2EE application:

1. Delete the exploded archive-format J2EE application from the J2EE server.
 2. Modify the application directory.
 3. Import the modified exploded archive-format J2EE application.
- To use the exploded archive-format J2EE applications, we recommend that you change the security settings so that the J2EE application can access any resource. For details on changing the security settings, see *13.4.3 Settings for using the exploded archive-format J2EE applications (changing the security settings)*.
 - When you import an exploded archive-format J2EE application, you cannot specify a path containing the UNC name as the application directory path. When a path containing the UNC name is specified, a command execution error occurs.
 - Do not add, delete, or overwrite the files or directories beneath the application directory when the J2EE server is being started and when the server management commands are being executed.
 - You cannot use the J2EE applications for which the `<alt-dd>` tag is specified in `application.xml`, in the exploded archive format. When you import an exploded archive-format J2EE application or when you import the EAR files and ZIP files exported from the J2EE server in an exploded archive format, a command execution error occurs if the `<alt-dd>` tag is specified in `application.xml` of the J2EE application.
 - In the J2EE server, multiple J2EE applications cannot specify the same directory as the application directory. When you import an exploded archive-format J2EE application or when you import the EAR files and ZIP files

exported from the J2EE server in an exploded archive format, a command execution error occurs if the specified directory already exists as the application directory of another J2EE application.

- If the J2EE server already contains J2EE applications with the following directories as the application directory or WAR directory, a command execution error occurs during the import operation:
 - Directory above the directory specified in the `-a` option or `-d` option of the `cjimportapp` command
 - Directory beneath the directory specified in the `-a` option or `-d` option of the `cjimportapp` command
- When you import an application by specifying the directory immediately beneath the Windows drive (such as `C:\`) or the UNIX root directory (`/`) in the `-a` option of the `cjimportapp` command, specify the application name in the `<display-name>` tag of the `application.xml` file.
- The JAR file name specified when you import the application is used as the directory name in the work directory. Specify the JAR file name such that the work directory path length does not reach the maximum limit for the platform. For details on estimating the work directory path length, see *Appendix C.1 Work directory of a J2EE server* in the *uCosminexus Application Server System Setup and Operation Guide*.
- We do not recommend the configuration in which the Java source files are placed under the application directory. If the class files and Java source files placed in the same directory are not synchronized, the application might fail to start or reload.
- Do not specify the following single-byte symbols in the application directory name and module name because these symbols are treated as escape characters:
! # % +
- The JAR files placed under the application directory are treated as the library JARs. Therefore, do not place stubs and interfaces obtained by the `cjgetstubsjar` command under the application directory.
- When you import the exploded archive-format J2EE applications, the library JAR is searched in all the directories beneath the application directory excluding the EJB-JAR directory and WAR directory. Therefore, if there are many files under the application directory excluding the EJB-JAR directory and WAR directory, the process of importing the applications might take some time.
- When you start an application, if the post-expansion absolute file path of the entries included in the WAR files, EJB-JAR files, or EAR files, which are expanded when the application is imported, do not exist in the respective root directories, the KDJE42389-E error message is displayed and the file expansion processing is interrupted.

13.5 Deploying and un-deploying J2EE applications

This section describes the deploying and un-deploying of J2EE applications.

The following table describes the organization of this section.

Table 13-5: Organization of this section (Deploying and un-deploying J2EE applications)

Category	Title	Reference location
Explanation	Deploying and un-deploying J2EE applications in the archive format	<i>13.5.1</i>
	Deploying and un-deploying J2EE applications in the exploded archive-format	<i>13.5.2</i>
	Deploying the EAR files and ZIP files in the exploded archive format	<i>13.5.3</i>
Settings	Setting up the J2EE application properties	<i>13.5.4</i>

Note: The functionality-specific explanation is not available for "Implementation" and "Operations".

13.5.1 Deploying and un-deploying J2EE applications in the archive format

This subsection describes the deploying and un-deploying of J2EE applications in the archive format.

(1) Deploying J2EE applications

To execute a Web application or an EJB application with a J2EE server, you must deploy the application as a J2EE application. Import the EAR file into the J2EE server that executes the J2EE application.

(2) Un-deploying J2EE applications

To delete an archive-format J2EE application, execute the un-deploy process for the J2EE application. After the un-deploy process finishes, the J2EE application is deleted from the J2EE server.

13.5.2 Deploying and un-deploying J2EE applications in the exploded archive-format

This subsection describes the deploying and un-deploying of J2EE applications in the exploded archive-format.

(1) Deploying J2EE applications

To import an exploded archive-format J2EE application into the J2EE server and to make the application executable from the client, you must deploy the application. Import the exploded archive-format J2EE application into the J2EE server that will execute the J2EE application. The class loaders generated when the application is started store and operate the application directory as a class path.

(2) Un-deploying J2EE applications

To delete an exploded archive-format J2EE application, execute the un-deploy process for the application. After the un-deploy process finishes, the registration of the J2EE application on the J2EE server is cancelled. Note that the application directory is not deleted.

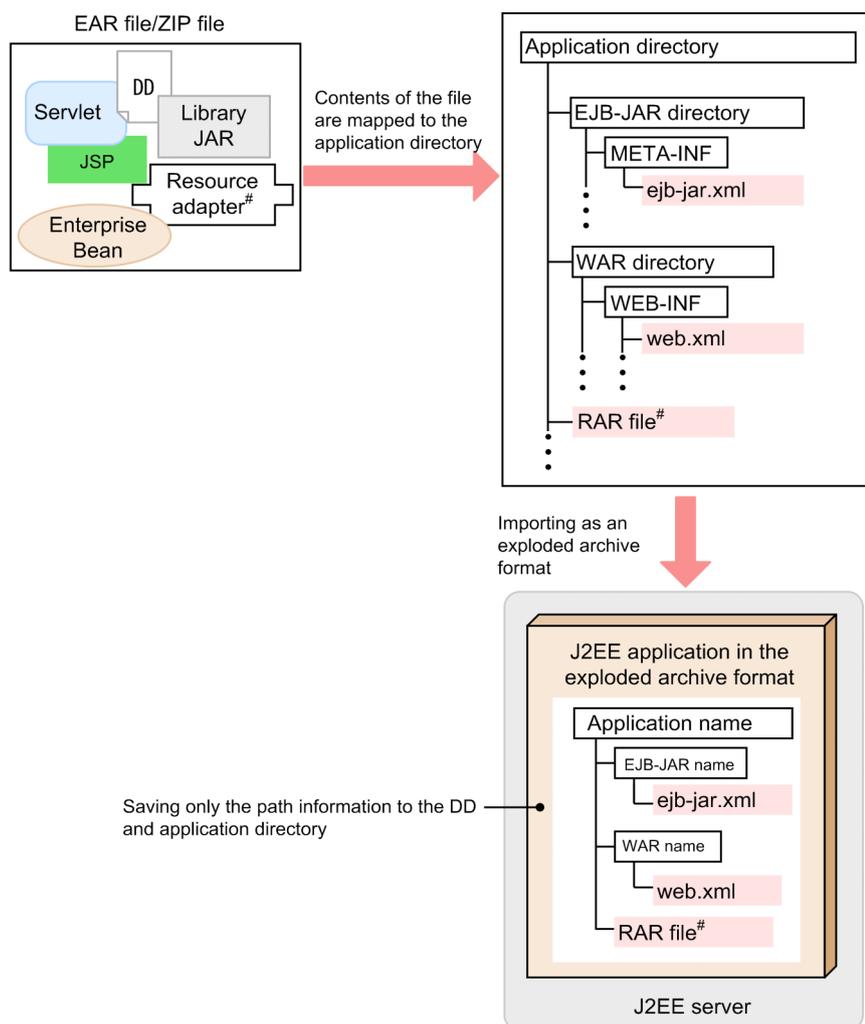
13.5.3 Deploying the EAR files and ZIP files in the exploded archive format

You can import the EAR files or the ZIP files exported from J2EE servers, in an exploded archive format into the J2EE server. By using this functionality, you can easily operate the created EAR files and ZIP files as exploded archive-format J2EE applications.

(1) Expanding the EAR files and ZIP files

Expand the contents of the EAR files and ZIP files into any directory and import the files into the J2EE server as the exploded archive-format J2EE applications. The following figure shows the expansion of the EAR files and ZIP files.

Figure 13-4: Expanding the EAR files and ZIP files



#

The resource adapter (RAR file) is included in EAR file/ZIP file and the application directory only when the resource adapter is used by including in the J2EE application. The resource adapter is not included in EAR file/ZIP file and the application directory, when it is used by deploying as a J2EE resource adapter.

When you import the files, specify the EAR file, ZIP file, and the expansion destination directory that forms the application directory, as the command arguments. The J2EE server receives the path information for the expansion destination directory from the command, generates the application directory based on the path information, and expands the contents of the EAR files and ZIP files under the application directory.

(2) Rules for generating the directory names

When you import the EAR files and ZIP files in the exploded archive format, a conflict of directory names might occur. The rules for generating the directory name differ according to the conflict pattern of the directory names. The following table describes the rules for generating the directory name for the EJB-JAR directory and WAR directory used for expanding the EAR files and ZIP files.

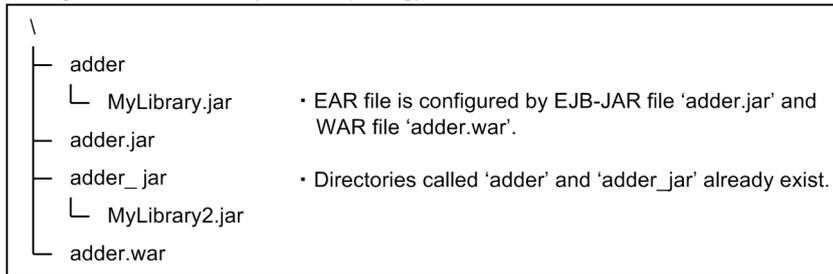
Table 13–6: Rules for generating the directory name for the EJB-JAR directory and WAR directory used for expanding the EAR files and ZIP files

Conflict pattern of the directory name	Rules for generating directory names
<p><code>application.xml</code> exists and the directory name does not conflict with the directory in the EAR file.</p>	<p>The string that remains after removing the extension (<code>.jar</code> or <code>.war</code>) from the EJB-JAR file and WAR file becomes the name of the EJB-JAR directory and WAR directory.</p> <p>If the EJB-JAR file and WAR file do not have an extension, the file name becomes the directory name as is.</p>
<ul style="list-style-type: none"> • The directory name conflicts in the EJB-JAR file and WAR file. • The directory name conflicts with the directory in the EAR file. 	<p>The string obtained by substituting the EJB-JAR file and WAR file extensions (<code>.jar</code> or <code>.war</code>) with <code>_jar</code> or <code>_war</code> respectively, becomes the name of the EJB-JAR directory and WAR directory.</p>
<p>The directory names conflict even when replaced with <code>_jar</code> or <code>_war</code>.</p>	<p>A serial number (1 to 2147483647) is added at the end of <code>_jar</code> or <code>_war</code>.</p>

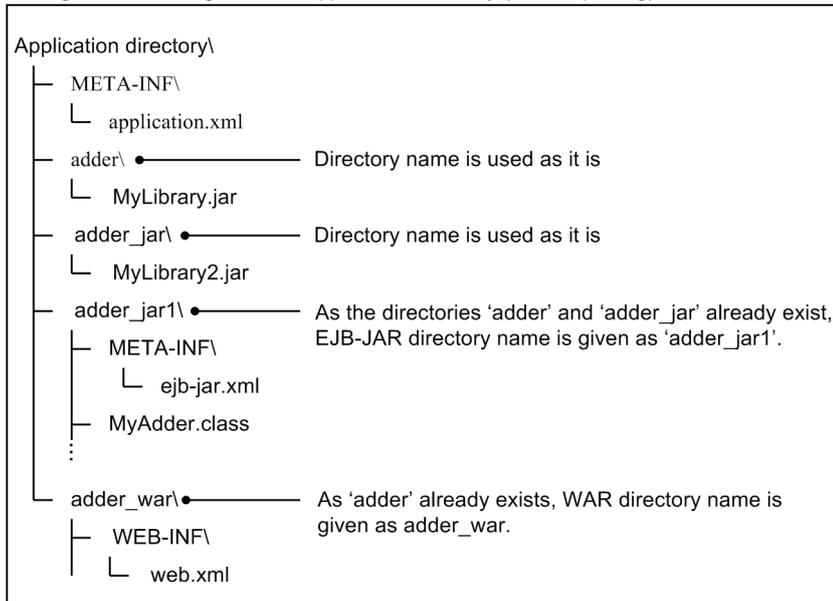
The following figure shows the EAR file configuration and the generated application directory configuration when a directory name conflict occurs.

Figure 13-5: Example of directory name conflict

Configuration of EAR file (before importing)



Configuration of the generated application directory (after importing)

**! Important note**

If the application directory, which is created by using the `-d` option of the `cjimportapp` command and expanding the EAR files and ZIP files, satisfies any of the following conditions, that application directory cannot be imported with the `-a` option of the `cjimportapp` command and used as the exploded archive-format application directory.

- An EJB-JAR module name does not end with `.jar`.
- A WAR module name does not end with `.war`.
- A module name without extension is duplicated with another module name without extension.
- A module name without extension is duplicated with a directory in the EAR file.

13.5.4 Setting up the J2EE application properties

To specify the property settings for the J2EE applications, use the server management commands.

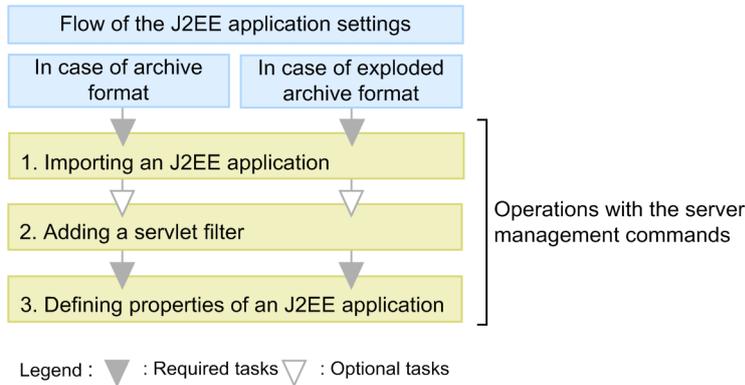
This subsection describes how to specify the settings for J2EE applications, assuming that the archive-format J2EE applications (EAR format/ ZIP format) and exploded archive-format J2EE applications are already created. Use products like WTP to create J2EE applications during application development. For details on creating an application, see *4. Developing J2EE Applications Using Eclipse* in the *uCosminexus Application Server Application Development Guide*.

Reference note

With the server management commands, you can also create an archive-format J2EE application from the created EJB-JAR file and WAR file. For details on how to create the archive-format J2EE applications using the server management commands, see *13.3 Archive-format J2EE applications*.

The following figure shows the flow of settings for a J2EE application.

Figure 13-6: Flow of J2EE application settings



A description of points 1 to 3 in the figure is as follows:

- Use the server management commands to import the J2EE application.
Use the `cjimportapp` command to import the J2EE application.
 - To import an archive-format J2EE application, specify an EAR-format/ ZIP-format file in the `-f` option.
 - To import an exploded archive-format J2EE application, specify the application directory in the `-a` option.
 - To import an EAR-format/ ZIP-format file in the exploded archive format, specify the EAR-format/ ZIP-format file in the `-f` option and specify the expansion destination directory that forms the application directory in the `-d` option.
- Use the server management commands to add a servlet filter in the J2EE application.
When you add a servlet filter, register the filter in the WAR file and then define the filter mapping. For details on adding a servlet filter, see *9.9 Filter settings* in the *uCosminexus Application Server Application Setup Guide*.
- Use the server management commands to define the J2EE application properties.
Obtain the property file using the `cjgetappprop` command and after editing the file, use the `cjsetappprop` command to apply the edited contents. For details on the property settings, see *9.1 Overview of the settings for the J2EE application properties* in the *uCosminexus Application Server Application Setup Guide*.

For details on the operations with the server management commands, see *3. Basic Operations of the Server Management Commands* in the *uCosminexus Application Server Application Setup Guide*. For details on the commands, see *2.3 Commands used with J2EE applications* in the *uCosminexus Application Server Command Reference Guide*. For details on the property files, see *3. Property files Used for Setting up J2EE Applications* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

Reference note

In the following cases, the J2EE applications can be set up efficiently by exporting or importing the J2EE applications containing the runtime information:

- When a J2EE application created in the development environment is exported and then imported and used in the operating environment
- When a J2EE application already running in the operating environment is exported and then imported and used on an additional J2EE server

Note that the J2EE applications cannot be exported or imported between hosts with different Application Server versions and platforms. To set up a J2EE application on the host that exports the J2EE application and the host with a different Application Server version and platform, create and set up a new J2EE application.

13.6 Replacing J2EE applications

A J2EE application might be replaced in order to upgrade and maintain the J2EE application, after the system operations begin.

Normally, to replace a J2EE application, you must stop the J2EE application that is running on the J2EE server, delete the application, and then import and deploy a new J2EE application.

For details on the normal J2EE application replacement procedure, see *5.6 Replacing J2EE applications* in the *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*.

13.7 Redeploying J2EE applications

Redeploy is a deployment method of replacing J2EE applications with fewer steps and a higher speed. This section describes how to replace J2EE applications by using the redeploy functionality.

The following table describes the organization of this section.

Table 13–7: Organization of this section (Redeploying J2EE applications)

Category	Title	Reference location
Explanation	Replacing J2EE applications by redeploying	13.7.1
	Statuses and replacement of J2EE applications	13.7.2
Notes	Notes on replacing J2EE applications by redeploying	13.7.3

Note: The functionality-specific explanation is not available for "Implementation", "Settings", and "Operations".

With the replacement of J2EE applications using the redeploy functionality, the information of all the J2EE application attributes can be inherited. You can use the redeploy functionality to replace J2EE applications in the following cases:

Replacing a J2EE application by modifying the logic during J2EE application testing

During testing, if you find a problem in the logic of the J2EE application and then modify the EJB-JAR files and WAR files in that J2EE application, you can use the redeploy functionality to replace the application being tested with the modified J2EE application.

Replacing a running J2EE application with a J2EE application tested in the development environment

You can use the redeploy functionality to replace a J2EE application existing in the development environment with a running J2EE application.

However, you can replace the J2EE applications that have the same application configuration in the operating environment and the development environment, and only a different logic.

When you use the redeploy functionality, the procedure required for replacing a normal J2EE application, such as stopping and deleting the J2EE application before the application is replaced, and importing and deploying the replaced J2EE application, become unnecessary. Therefore, you can replace a J2EE application using fewer steps as compared to the normal process of replacing J2EE applications.

Note that the redeploy functionality can be used to replace the archive-format J2EE applications. You cannot use the redeploy functionality with the exploded archive-format J2EE applications.

Reference note

We recommend that you execute the JSP pre-compile functionality before executing the redeploy functionality. The JSP pre-compile functionality is a functionality that compiles the JSP files included in the Web application before the files are deployed and generates the class files. The procedure until the generation of the class files is implemented in advance, so the response time when the request first reaches the JSP and the start-up time of the Web application can be reduced. For details on the JSP pre-compile functionality, see 2.5 *JSP pre-compile functionality and the storage of the compilation results* in the *uCosminexus Application Server Web Container Functionality Guide*.

13.7.1 Replacing J2EE applications by redeploying

This subsection describes how to replace J2EE applications by redeploying.

Redeploy is a deployment method of replacing the archive-format J2EE applications with fewer steps and a higher speed. You can use this functionality to replace J2EE applications where only the logic is changed. You can redeploy with the server management commands.

The following points describe the conditions in which redeploy can be executed:

Conditions in which redeploy can be executed

- Only a J2EE application that does not contain runtime information is replaced. A J2EE application containing runtime information (ZIP file) cannot be redeployed.

- The configuration of the J2EE application before and after replacement must be the same. Redeployment cannot be executed if the number of EJB-JARs, resource adapters, and WARs contained in the J2EE application are different and the file names are different. Furthermore, the name of the J2EE application must also be the same.
- The method definitions and annotation values of the home interfaces (local and remote), component interfaces (local and remote), and business interfaces (local and remote) in the EJB-JAR included in the J2EE application before and after replacement must be the same.
- When the settings for inheriting only the runtime attributes are specified, the contents defined in the DD file (`application.xml`, `ejb-jar.xml`, `ra.xml`, and `web.xml`) already set up in the application development environment, must be the same.

Furthermore, when you replace a J2EE application, if you rename and save the J2EE application before replacing, you can manage the generation of the J2EE application based on the names. This subsection also describes how to rename a J2EE application.

With redeploying, the information of a J2EE application before replacement is inherited by the replaced J2EE application. The following information is inherited:

For Application Server version 06-70 or later

By default, all the attributes of a J2EE application before replacement are inherited by the replaced J2EE application. If you want only the runtime attributes to be inherited as in the versions earlier than 06-70, you must specify an option and execute the `cjreplaceapp` command. For details on the command, see *cjreplaceapp (Replacing applications)* in the *uCosminexus Application Server Command Reference Guide*.

For Application Server version 06-70 or earlier

The runtime attributes[#] set in the property file of the J2EE application before replacement, are inherited.

[#]: You can specify the DD definitions (`application.xml`, `ejb-jar.xml`, `ra.xml`, and `web.xml`) and the property file-specific definitions in the property file. The property file-specific definitions are called *runtime attributes*.

! Important note

If the J2EE application to be replaced contains `cosminexus.xml`, the Application Server-specific definition information before replacement is returned to the default value once and then the definition information of `cosminexus.xml` is read. If the application to be replaced does not contain `cosminexus.xml`, the definition information before `cosminexus.xml` is replaced is inherited.

When you execute redeploy, the J2EE application might be either running or stopped. When a running J2EE application is replaced, the J2EE application starts automatically after being replaced. However, the J2EE application-related objects stored in the pool and cache are destroyed. When a stopped J2EE application is replaced, the replaced J2EE application is also in the stopped state. For details on the relationship between the statuses and the redeployment of J2EE applications, see *13.7.2 Statuses and replacement of J2EE applications*.

Tip

If a J2EE application is redeployed when the application is running, the application is stopped during the redeploy processing and is restarted after being replaced. At this time, if the execution time of the stop processing exceeds the timeout value set with the server management commands (`cjreplaceapp`), the J2EE application is stopped forcefully. If no timeout value is specified and if the processing exceeds the default timeout value of 60 seconds, the J2EE application is stopped forcefully. If the timeout period is further exceeded after the forced termination, the command terminates abnormally.

Note that when a replaced J2EE application is restarted, if the execution time for the start processing exceeds the timeout value specified in the `ejbserver.rmi.request.timeout` key of `usrconf.properties` for the server management commands, the command terminates abnormally.

The format and example of execution of the replacement operation is as follows.

Format of execution

```
cjreplaceapp J2EE-server-name -name J2EE-application-name -f Path-of-EAR-file-to-be-replaced
```

Example of execution

```
cjreplaceapp MyServer -name Appl -f Appl.ear
```

13.7.2 Statuses and replacement of J2EE applications

With the redeploy functionality, you can replace a J2EE application regardless of whether the application is running or stopped.

This section describes the replacement of a running application and a stopped application.

(1) When a running J2EE application is replaced

When a running J2EE application is replaced, the replaced J2EE application is also in the running state.

If you redeploy a running J2EE application, the replacement processing is implemented with the following procedure:

1. Stop the J2EE application
2. Replace the J2EE application
3. Start the J2EE application

Each processing has a timeout value.

Timeout for stopping the application

When the execution time for stopping a J2EE application exceeds the timeout value in the server management command, the J2EE application is stopped forcefully. After the forced termination, if the J2EE application does not stop even if after the timeout value set by forced termination is exceeded, the server management command terminates abnormally.

For details on the timeout settings, see *cjreplaceapp (Replacing applications)* in the *uCosminexus Application Server Command Reference Guide*.

Note that the value that could be acquired first is applied to the timeout value in the following order:

1. Timeout value set in the server management command
2. Default value (60 seconds)

Timeout for replacing and starting the application

When the execution time for replacing and starting a J2EE application exceeds the time set as the timeout value for the respective processing, the server management command terminates abnormally. Note that the time set in the timeout for RMI-IIOP communication is applied to the timeout value. For details on the RMI-IIOP communication timeout, see *2.11.5 Timeout in the RMI-IIOP communication* in the *uCosminexus Application Server EJB Container Functionality Guide*.

(2) When a stopped J2EE application is replaced

When a stopped J2EE application is replaced, the replaced J2EE application is also in the stopped state. When a stopped J2EE application is redeployed, only the replacement processing of the application is executed.

13.7.3 Notes on replacing J2EE applications by redeploying

This subsection describes the notes on replacing J2EE applications by redeploying.

- When a J2EE application is replaced, an error occurs if a J2EE application with the same name does not exist on the J2EE server. The J2EE application cannot be replaced in such a case.
- When you replace a J2EE application by redeploying, the runtime information of the J2EE application before replacement is inherited by the replaced J2EE application (new J2EE application). Therefore, if the replaced J2EE application contains the runtime information, the application cannot be replaced. The processing terminates with an error if you attempt such a replacement.
- By default, all the attributes of a J2EE application before replacement are inherited by the replaced J2EE application. If you want only the runtime attributes[#] to be inherited as in the versions earlier than 06-70, you must specify an option and execute the `cjreplaceapp` command. For details on the command, see *cjreplaceapp (Replacing applications)* in the *uCosminexus Application Server Command Reference Guide*.
- If there are differences in the following contents of the J2EE applications before and after replacement, the replacement processing terminates with an error:

- The number of EJB-JAR files, resource adaptors, and WARs is different in the J2EE application.
- The EJB-JAR file name, RAR file name, and WAR file name in the J2EE application differ before and after replacement.
- The method definitions of the home interfaces (local or remote), component interfaces (local or remote), and business interfaces (local or remote) in the EJB-JAR file are different.
- The DD definitions (`application.xml`, `ejb-jar.xml`, `ra.xml`, `web.xml`) differ when the settings are specified to inherit only the runtime attributes[#].
- The set annotation values are changed.

Also, when the `-replaceDD` option is specified to replace the DD files (`application.xml`, `ejb-jar.xml`, `ra.xml`, and `web.xml`) as well, an error occurs if the following conditions are also fulfilled:

- The DD file (`application.xml`, `ejb-jar.xml`, `ra.xml`, and `web.xml`) tags are different.
 - The DD file (`application.xml`, `ejb-jar.xml`, and `web.xml`) does not exist or a non-existent DD file comes into existence.
- When replacing a running J2EE application, the J2EE application-related objects that are stored in the pool or cache are destroyed.
 - If `application.xml` version is 1.4 or earlier and for Java EE 5 or later, the conditions for determining the library JAR are different. For details on the rules for determining the modules such as the library JAR modules, see *11.4.3 Rules for determining the modules when application.xml exists*.
 - If the replaced J2EE application contains `cosminexus.xml` and uses CMP 2.0, you must execute the `cjreplaceapp` command when the J2EE application is in a stopped state. Also, after executing this command, you must execute the `cjgencmpsql` command before deploying the application.

#

You can specify the DD definitions (`application.xml`, `ejb-jar.xml`, `ra.xml`, and `web.xml`) and the property file-specific definitions in the property file. The property file-specific definitions are called *runtime attributes*.

13.8 Detecting updates and reloading the J2EE applications

For the exploded archive-format J2EE applications, if you update the files to be configured, you can reload the updated J2EE applications. By using the reload functionality, you dynamically replace the already deployed servlets, JSPs, and EJB-JARs without restarting the J2EE server.

When you use the redeploy functionality, the procedure required for replacing a normal J2EE application, such as stopping and deleting the J2EE application before the application is replaced, and importing and deploying the replaced J2EE application, become unnecessary. As a result, the J2EE application is dynamically replaced within a few steps.

You can use the reload functionality to replace the modified J2EE applications and running J2EE applications during testing in application development and during system operations.

This section describes the detection of updates and the reloading of J2EE applications.

Note that for the applications containing `cosminexus.xml`, the `cosminexus.xml` information is not updated even if update detection and reloading is executed for J2EE applications.

The following table describes the organization of this section.

Table 13–8: Organization of this section (Detecting updates and reloading the J2EE applications)

Category	Title	Reference location
Explanation	How to reload J2EE applications	<i>13.8.1</i>
	Scope of reloading	<i>13.8.2</i>
	Class loader configuration used for reloading	<i>13.8.3</i>
	Operations when an error occurs	<i>13.8.4</i>
	Files for update detection	<i>13.8.5</i>
	Update detection interval for J2EE applications	<i>13.8.6</i>
	Interval for updating the J2EE application configuration file	<i>13.8.7</i>
	Reloading the Web applications	<i>13.8.8</i>
	Reloading the JSPs	<i>13.8.9</i>
	Relationship with the other functionality	<i>13.8.10</i>
	Reloading the J2EE applications using commands	<i>13.8.11</i>
Settings	Settings for detecting updates and reloading J2EE applications	<i>13.8.12</i>
Notes	Notes and restrictions related to reloading	<i>13.8.13</i>

Note: The functionality-specific explanation is not available for "Implementation" and "Operations".

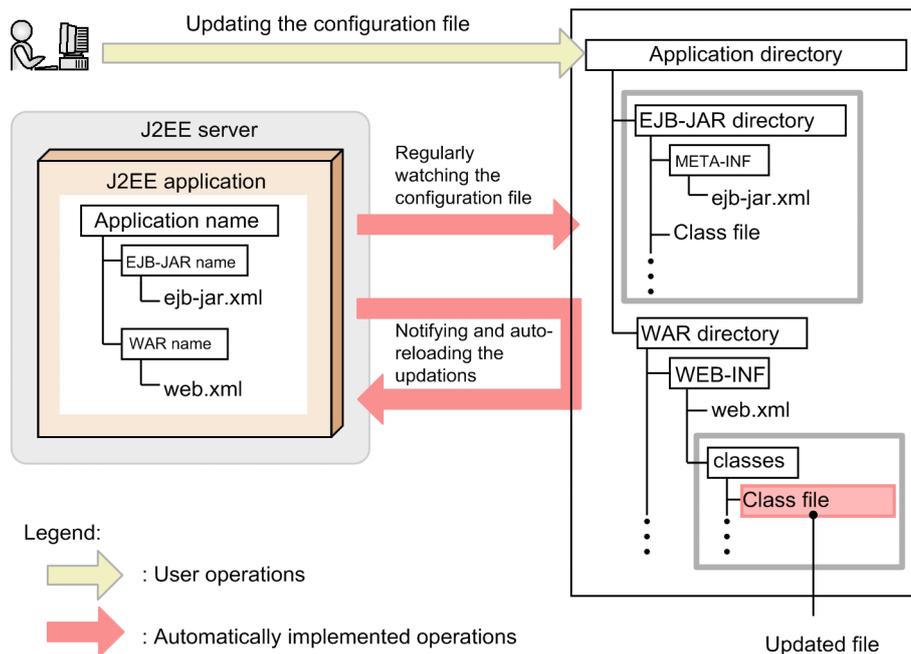
13.8.1 How to reload the J2EE applications

The methods of reloading J2EE applications include reloading by detecting updates and reloading by using commands.

(1) Reloading by detecting updates

You can use the method of reloading by detecting updates as a functionality to support testing during the development of J2EE applications. The following figure shows the process of reloading by detecting updates:

Figure 13-7: Reloading by detecting updates



When the EJB applications (EJB-JAR) and Web applications (WAR) configuring the exploded archive-format J2EE application are updated, the J2EE sever detects the J2EE application updates and the updated EJB-JAR and WAR are reloaded automatically.

The J2EE server periodically monitors the J2EE application configuration file and reloads the J2EE application if a configuration file update is detected. The following figure shows the flow of processing from the update to the reload of J2EE applications:

Figure 13-8: Process flow of update detection-based reloading

1. Updating the configuration file of a J2EE application
2. J2EE server detects the updations in a J2EE application
3. Locking the request process
4. Ending the J2EE application
5. Serializing of the session information and output of the information to the session information file
6. Creating a new class loader
7. Reading the session information from the session information file and de-serializing the session information
8. Deleting the session information file
9. Starting a J2EE application
10. Restarting the request process

Legend:
 : Procedure implemented only at the time of reloading the Web application

The following is a description of points 1 to 10 in the figure:

1. Update the J2EE application configuration file.

Update the EJB-JAR and WAR files that configure the exploded archive-format J2EE application.

2. J2EE server detects an update of the J2EE application configuration file.

- The J2EE server periodically monitors the J2EE application configuration file and detects the update if the configuration file is updated. The interval between monitoring the J2EE application configuration file and detecting an update is set as the update detection interval. For details on the update detection interval, see *13.8.6 Update detection interval for J2EE applications*.
- After an update is detected, the J2EE server reloads the updated file. Sometimes the J2EE server might start loading the file while the file is being copied and the loading might fail. To avoid this, you can set the time from the detection of configuration file update to the start of monitoring of the requests being processed, as the interval for updating the configuration files. For details on the interval for configuration file update, see *13.8.7 Interval for updating the J2EE application configuration file*.
- When the JSPs are updated, the update is detected by JSP recompilation or by monitoring the class files. For details on the reloading of JSPs, see *13.8.9 Reloading the JSPs*.

3. The processing of the request is blocked.

If the J2EE server detects a configuration file update and if the time specified in the interval for configuration file update lapses, the J2EE server locks the processing of the requests in order to reload the J2EE application.

- For the EJB applications (EJB-JAR)
An error is returned if a new request arrives. The processing continues if a request is being processed. However, by using the CTM for Stateless Session Bean, you can set a new request to a pending state.
- For the Web applications (WAR)
If a new request arrives, the request is added to the pending list. The processing continues if a request is being processed. If you use the delayed reloading functionality at this time, the reload start processing can be delayed. For details on delayed reloading, see *13.8.8(1) Delayed reloading of the Web applications*.

Reference note

If the processing of the request being processed is not complete, you can start the reload processing by executing the method timeout and method cancellation functionality for monitoring the J2EE application execution time. For details on the relationship between reloading and the monitoring of the J2EE application execution time, see *13.8.10(2) Relationship between reloading and monitoring the J2EE application execution time*.

4. Execute the termination processing of the J2EE application.

To execute the reloading process, terminate the J2EE application. The following processing is implemented during termination:

- The servlet instance loaded into the class loader used before reloading is deleted. If the servlet implements the `destroy` method, the `destroy` method is executed. Even the servlet instances generated from the JSP files are destroyed. At this time, if the JSP file implements the `jspDestroy` method, the `jspDestroy` method is executed.
- The objects registered in `javax.servlet.ServletContext` are destroyed.
- The EJB instances loaded into the class loader used before reloading are destroyed. At this time, the following EJB callback methods are executed:

Category	Executed method
For Stateless Session Bean	<code>ejbRemove</code> <code>PreDestroy</code>
For Stateful Session Bean	<code>ejbRemove</code> <code>PreDestroy</code>
For an Entity Bean	<code>unsetEntityContext</code>
For a Message-driven Bean	<code>ejbRemove</code>

5. Serialize the session information and output the information to the session information file.

When a Web application is reloaded, you can continue to use the session information that was generated before reloading, even after reloading. For details on inheriting the session information, see *13.8.8(2) Inheriting the session information when the Web applications are reloaded*.

6. Create a class loader for J2EE applications.
When a J2EE application is reloaded, the class loader for J2EE applications is created and is used to process the requests after reloading.
7. Read the session information from session information file and de-serialize.
When a Web application is reloaded, the session information output in the session information file is read in the new class loader.
8. Delete the session information file.
9. Start the J2EE application.
 - After reloading, an instance of the updated servlet is created during the first access, and the `init` method is executed.
 - With the start processing, the EJBs equal to the minimum value of the pool are generated and pooled. At this time, the following EJB callback methods are executed:

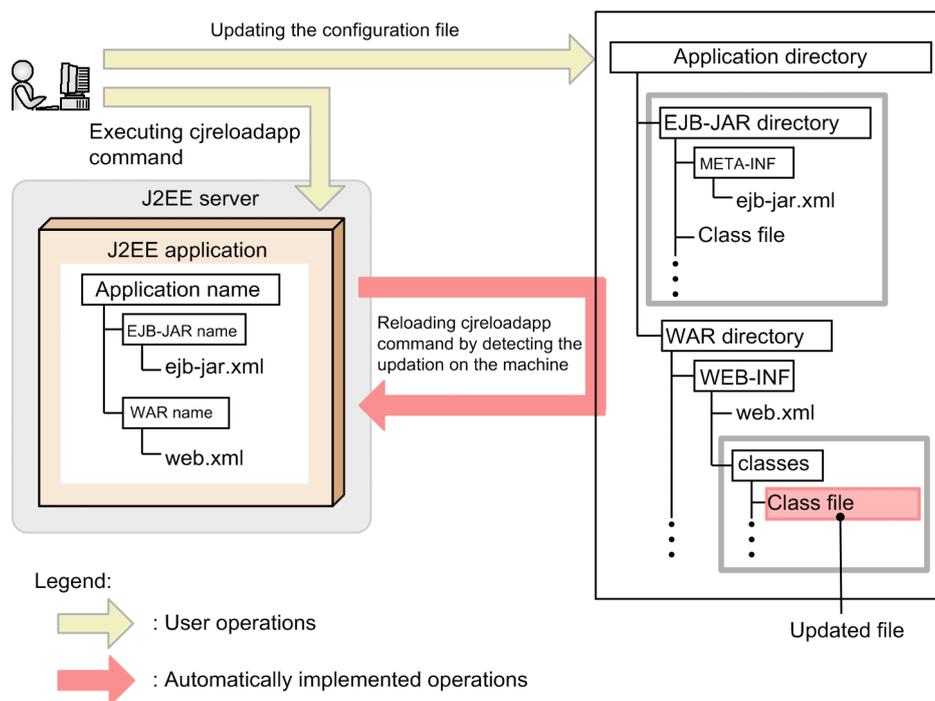
Category	Executed method
For Stateless Session Bean	<code>setSessionContext</code> <code>ejbCreate</code> <code>PostConstruct</code>
For an Entity Bean	<code>setEntityContext</code>
For a Message-driven Bean	<code>setMessageDrivenContext</code> <code>ejbCreate</code>

10. Cancel the blocking of requests and restart the processing of requests.
The processing of requests that were pending in 3 is restarted.

(2) Reloading by using commands

You can use command-based reloading as a functionality to support testing during J2EE application development or as a functionality to support the replacement of J2EE applications when the system is running. The following figure shows command-based reloading.

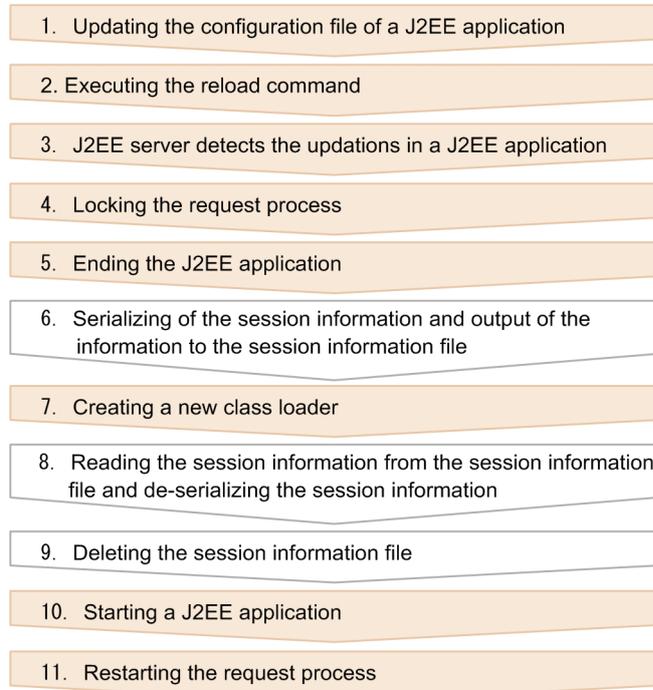
Figure 13-9: Reloading by using commands



When the EJB applications (EJB-JAR) and Web applications (WAR) configuring an exploded archive-format J2EE application are updated, the user executes the `cjreloadapp` command. The J2EE server detects a J2EE application update through the `cjreloadapp` command and automatically reloads the updated EJB-JAR and WAR.

The following figure shows the flow of processing from the update to the reload of the J2EE application.

Figure 13–10: Process flow of command-based reloading



Legend:

: Procedure implemented only at the time of reloading the Web application

The following is a description of points 1 to 4 in the figure. Point 5 and later steps are the same as that for reloading by detecting updates. For point 5 and later steps, see *(1) Reloading by detecting updates*.

1. Update the J2EE application configuration file.
Update the EJB-JAR and WAR files that configure the exploded archive-format J2EE application.
2. Execute the `cjreloadapp` command.
3. The J2EE server detects an update of the J2EE application configuration file through the `cjreloadapp` command.
4. The processing of the requests is blocked.
The request processing is blocked in order to reload the J2EE application.
 - For the EJB applications (EJB-JAR)
An error is returned if a new request arrives. The processing continues if a request is being processed. However, by using the CTM for Stateless Session Bean, you can set a new request to a pending state.
 - For the Web applications (WAR)
If a new request arrives, the request is added to the pending list. The processing continues if a request is being processed.

Reference note

If the processing of the request being processed is not complete, you can start the reload processing by executing the method timeout and method cancellation functionality for monitoring the J2EE application execution time. For details on the relationship between reloading and the monitoring of the J2EE application execution time, see *13.8.10(2) Relationship between reloading and monitoring the J2EE application execution time*.

13.8.2 Scope of reloading

The following table describes the types of applications that can be specified for reloading.

Table 13–9: Types of applications that can be specified for reloading

Types of applications		Applicability	Restrictions
EJB application (EJB-JAR)	Stateless Session Bean	R	A new request during the reload process results in an error. Note that if the CTM is used, a new request during the reload process is added to the pending list.
	Stateful Session Bean	R	A new request during the reload process results in an error. Also, if reloaded, the application state is destroyed, so the application state returns to the state before the application was started.
	Singleton Session Bean	R	If reloaded, the Bean instance is destroyed, and an instance is re-created, so the state of the instance is not stored.
	Entity Bean	R	A new request during the reload process results in an error.
	Message-driven Bean	R	
Web application (WAR)	Servlet	Y	--
	JSP	Y	--
Library JAR	--	Y	--

Legend:

- Y: Can be specified for reloading.
- R: Can be specified for reloading, there are restrictions
- : Not applicable

You can specify the scope of reloading with the following range:

- `app`: The EJB applications (EJB-JAR) and Web applications (WAR) are reloaded.
- `web`: Only the Web applications (WAR) are reloaded.
- `jsp`: Only the JSPs are reloaded.

Note: `app`, `web`, `jsp` are the values specified in the `ejbserver.deploy.context.reload_scope` key of `usrconf.properties`. Note that if you specify `none`, the reload functionality is disabled.

When `app` is specified

- If you update an EJB application, the EJB application, servlets, and JSPs are reloaded.
- If you update a servlet, the servlets and JSPs are reloaded.
- If you update a JSP, the JSP is reloaded.

When `web` is specified

- If you update a servlet, the servlets and JSPs are reloaded and if you update a JSP, the JSP is reloaded.
- If the servlets exist and the JSPs do not exist, only the servlets are reloaded. If the JSPs exist and the servlets do not exist, only the JSPs are reloaded.
- Reloading is not performed even if you update the EJB application.

When `jsp` is specified

- If you update a JSP, the JSP is reloaded.
- Reloading is not performed even if you update the EJB application or servlet.

Note that enabling or disabling of the reload functionality is decided based upon the combination of the scope of the local call optimization functionality specified in the `ejbserver.rmi.localinvocation.scope` key of

`usrconf.properties` and scope of the reload functionality. The following table describes the mapping of the scope of the local call optimization functionality and the scope of the reload functionality.

Table 13-10: Mapping of the scope of the local call optimization functionality and the scope of the reload functionality

Items		Value of the <code>ejbserver.rmi.localinvocation.scope</code> key		
		all	app	none
Scope of local call optimization		Within the same J2EE server.	Within the same application.	There is no range.
Value of the <code>ejbserver.deploy.context.reload_scope</code> key	app	N#	Y	Y
	web	Y	Y	Y
	jsp	Y	Y	Y
	none	N	N	N

Legend:

Y: Reload functionality can be used

N: Reload functionality cannot be used

#: The settings are incorrect. If you specify `ejbserver.deploy.context.reload_scope=app` for `ejbserver.rmi.localinvocation.scope=all`, a message is output when you start the J2EE server and the server fails to start.

13.8.3 Class loader configuration used for reloading

The Class loader configuration used for reloading differs according to the scope of local call optimization. The following table describes the mapping of the scope of local call optimization and the class loader configuration. For details on the class loader configuration, see *Appendix B Class loader configuration*.

Table 13-11: Mapping of the scope of local call optimization and class loader configuration

Items	Value of the <code>ejbserver.rmi.localinvocation.scope</code> key #		
	all	app	none
Scope of local call optimization	Within the same J2EE server.	Within the same application.	There is no range.
Class loader configuration	Class loader configuration used for local call optimization.	Default class loader configuration.	Default class loader configuration.

#: This key is specified in `usrconf.properties`.

With the reload functionality, the class loader beneath `ApplicationClassLoader` or `WebappClassLoader` is replaced. When an EJB-JAR is reloaded, the following files will be loaded with the default class loader configuration:

- With `ApplicationClassLoader`, the EJB-JARs, library JARs, and reference libraries included in the J2EE application will be loaded.
- With `WebappClassLoader`, the WARs included in the J2EE application will be loaded.
- With `JasperLoader`, the JSPs included in the J2EE application will be loaded.

When you replace `ApplicationClassLoader` to reload an EJB-JAR, you must also replace the lower `WebappClassLoader` and `JasperLoader`. Therefore, when the EJB-JARs, library JARs, and reference libraries are reloaded, the reloading includes the WARs.

13.8.4 Operations when an error occurs

The following table describes the operations to be performed when an error occurs while the reload functionality is being used.

Table 13–12: Operations when an error occurs in the reload functionality

Error contents	Results
When a request arrives in the server during reloading (state when no request is being processed in the application)	For the EJB-JARs, an error is returned to the client. For the WARs, the new request remains pending until the reloading is finished.
When an exception occurs during reloading (including failure in the loading of the class file)	The application is stopped without stopping the update detection process. When an update is detected, the stopped J2EE application is restarted.
When an exception occurs in processing other than reloading (during update check, during the interval for updating configuration file, during delayed reloading)	The monitoring continues with the J2EE application running and without stopping the update detection process.
When a J2EE application method does not terminate because it has hung up	Reload cannot be executed unless you cancel the method for monitoring the J2EE application execution time. Also, reload cannot be executed unless you use the <code>cjreloadapp</code> command. For details on the monitoring of the J2EE application execution time, see <i>13.8.10(2) Relationship between reloading and monitoring the J2EE application execution time</i> .

13.8.5 Files for update detection

Of the files loaded using the class loader, the J2EE server detects the updates and reloads the files when the monitored files are updated. The file updates are detected when the J2EE application starts.

However, the updates are not detected for the files that are not loaded with the class loader. The updates are not detected for the files that are in the application directory but are not loaded and the J2EE application files that do not use the application directory. Furthermore, the updates are not detected for the DD (`ejb-jar.xml` and `web.xml`).

Also, with the exploded archive format, if the date of update of an updated JSP file is old, the JSP re-compilation is not executed.

(1) Target files for update detection

The following table describes the target files for update detection.

Table 13–13: Target files for update detection

Type	Target files for update detection	Scope of reloading ^{#1}			
		app	web	jsp	
Files beneath the application directory	<ul style="list-style-type: none"> Library JARs 	Y	--	--	
EJB application files (EJB-JARs) beneath the EJB-JAR directory	<ul style="list-style-type: none"> Class files Property files loaded by <code>java.util.ResourceBundle</code> 	Y	--	--	
Web application files (WARs) beneath the WAR directory	Servlet	<ul style="list-style-type: none"> Class files JAR files Property files loaded by <code>java.util.ResourceBundle</code> 	Y	Y	--
	JSP	<ul style="list-style-type: none"> JSP files 	Y ^{#2}	Y ^{#2}	Y ^{#2}
		<ul style="list-style-type: none"> Tag files 	Y ^{#3}	Y ^{#3}	Y ^{#3}

Type		Target files for update detection	Scope of reloading ^{#1}		
			app	web	jsp
Web application files (WARs) beneath the WAR directory	JSP	• Tag library descriptors	Y ^{#3}	Y ^{#3}	Y ^{#3}
		• JSP compilation results • Tag file compilation results	Y ^{#4}	Y ^{#4}	Y ^{#4}
		• Static contents	Y ^{#5}	Y ^{#5}	Y ^{#5}
Files other than those beneath the application directory		• Reference libraries	Y	--	--

Legend:

Y: Updates are detected

N: Updates are not detected

#1: app, web, and jsp in the scope of reloading are the values specified in the `ejbserver.deploy.context.reload_scope` key of `usrconf.properties`.

#2: The updates are detected for the JSPs that are not pre-compiled and are already loaded.

#3: The updates are detected if the tag files and tag library descriptors are used with loaded JSP files or tag files when the JSPs are not pre-compiled.

#4: The updates are detected when the JSPs are pre-compiled.

#5: The updates are detected when the JSP files or tag files are dependent files. The dependent files are the files that are included in the include directive of the JSP files or tag files and the files that are included in `<include-pragma>` or `<include-coda>` of `web.xml`.

(2) Examples of target files for update detection

The following table describes the examples of files for which updates are detected from among the files beneath the application directory.

Table 13-14: Examples of target files for update detection

Directory		Files beneath the directory	Update detection target
<i>Application-directory</i>		Example of library JARs: <code>MyLibrary.jar</code>	Y
	<code>META-INF</code>	<ul style="list-style-type: none"> • DD for J2EE applications (<code>application.xml</code>) • Cosminexus application property file (<code>cosminexus.xml</code>) 	--
<i>EJB-JAR-directory</i>		--	--
	<code>META-INF</code>	DD for the EJB-JARs (<code>ejb-jar.xml</code>)	--
	<i>Package-name</i>	Enterprise Bean classes Example: <code>MyEJB.class</code>	Y
		Home interface classes Example: <code>MyEJBHome.class</code>	Y
		Component interface classes Example: <code>MyEJBRemote.class</code>	Y
		Business interface classes	Y

Directory				Files beneath the directory	Update detection target	
		<i>Package-name</i>		Example: <code>MyBusiness.class</code>	Y	
				Classes used from the EJB Example: <code>MyClass.class</code>	Y	
				Property files loaded by <code>java.util.ResourceBundle</code> Example: <code>MyResource.properties</code>	Y	
	<i>WAR-directory</i>			JSP files Example: <code>MyJsp.jsp</code>	Y	
				Static contents Example: <code>MyStatic.jspf</code>	Y	
				HTML files Example: <code>MyIndex.html</code>	--	
		META-INF		--	--	
		WEB-INF		DD for the Web applications (<code>web.xml</code>)	--	
				Tag file descriptor file Example: <code>MyTaglib.tld</code>	Y	
				Web application property file (<code>hitachi_web.properties</code>)	--	
			classes	--	--	
			<i>Package-name</i>	Property files loaded by <code>java.util.ResourceBundle</code> Example: <code>MyResource.properties</code>	Y	
				Servlet classes Example: <code>MyServlet.class</code>	Y	
			lib	JAR files Example: <code>MyLibrary.jar</code>	Y	
			tags	Tag files Example: <code>MyTag.tag</code>	Y	
			<i>JSP-working-directory</i>	JSP compilation results Example: <code>MyJsp\$jsp.class</code>	Y	
			WEB-INF	tags	Tag file compilation results Example: <code>MyTag\$tag.class</code>	Y
<i>Directory-other-than-the-application-directory</i>				Reference libraries Example: <code>MyRef.jar</code>	Y	

Legend:

Y: Updates are detected

N: Updates are not detected

(3) Operations that are reloaded

When the files to be monitored are changed, whether updates are detected and reload is executed differs depending upon the file operations. The following table describes the operations that are reloaded.

Table 13-15: Operations that are reloaded

Target files for update detection		File operations			Notes
		Add	Delete	Update	
Files beneath the application directory	Library JAR	N	N	Y	<ul style="list-style-type: none"> Annotations are also read.^{#1} You cannot change the names of the library JARs. You can change the library JAR configuration.
	Cosminexus application property file	N	N	N	<ul style="list-style-type: none"> If the contents of <code>cosminexus.xml</code> are changed or deleted when the reload process is executed and, the definition information is not read again.
EJB application files (EJB-JARs) beneath the EJB-JAR directory	Class files configuring the EJB	N	N	Y	<ul style="list-style-type: none"> Annotations are also read.^{#1} You cannot change the class name. If the class that was not referenced so far is updated so that the class is now referenced, the reference destination class file is also reloaded.
	Other class files	N	N	Y	<ul style="list-style-type: none"> Annotations are also read.^{#1} If the class that was not referenced so far is updated so that the class is now referenced, the reference destination class file is also reloaded.
Web application files (WARs) beneath the WAR directory	Class files	R	Y	Y	--
	JAR files	Y	Y	Y	--
	Property files	R	Y	Y	--
	JSP files	R	Y ^{#2}	Y ^{#2}	--
	Tag files	R	Y ^{#2}	Y ^{#2}	<ul style="list-style-type: none"> Excludes the tag files in the JAR files.
	Files on which the JSP files or tag files depend	R	Y ^{#2}	Y ^{#2}	--
	Static contents	R	N	N	<ul style="list-style-type: none"> When reloading functionality is used, the cache is disabled.
	JSP compilation results (class files generated by compiling the JSPs and tag files)	R	Y	Y	<ul style="list-style-type: none"> The JSP pre-compile functionality must be enabled.
Files other than those beneath the application directory	Reference libraries	N	N	Y	<ul style="list-style-type: none"> You cannot change the name of the reference library.

Target files for update detection		File operations			Notes
		Add	Delete	Update	
Files other than those beneath the application directory	Reference libraries	N	N	Y	<ul style="list-style-type: none"> You can update the configuration of the JAR files and the files in the class path that is specified by the reference library. If the stub and container-generated classes must be re-generated, the classes are regenerated.

Legend:

Y: Reloading process is executed

R: Operations can be performed, but the reloading process is not executed

N: Reloading process is not executed

--: Not applicable

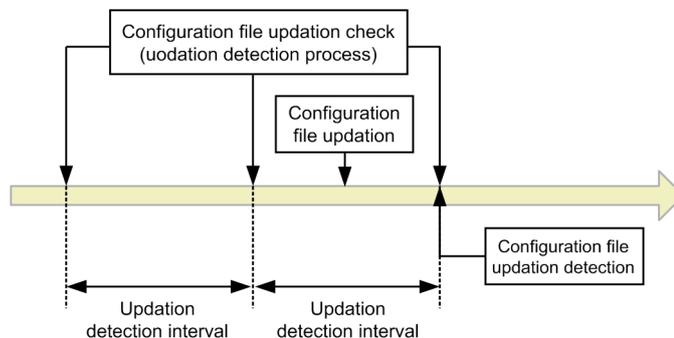
#1: Some annotations can be updated and some cannot be updated. For details on annotations, see *12.6.1 Updating the annotations*.

#2: When the JSP pre-compile functionality is enabled, R.

13.8.6 Update detection interval for J2EE applications

The J2EE server periodically monitors the J2EE application configuration file and detects an update if the configuration file is updated. The interval between monitoring the J2EE application configuration file and detecting an update is set as the update detection interval. The following figure shows the update detection interval.

Figure 13-11: Update detection interval



If the value of the update detection interval is increased, the interval for monitoring the configuration file becomes prolonged and the reloading of the updated file is applied with a delay. Also, if the value is decreased, reloading is applied faster.

Tip

If the number of files for which updates are to be detected increase, the overheads of update detection along with the CPU usage increases. In such cases, you can reduce the impact on the performance by changing the update detection interval. We recommend that you increase the value of the update detection interval.

You can set the update detection intervals for J2EE applications, Web applications, and JSPs respectively. The relationship between the set values is as follows:

For EJB applications

The update detection interval for J2EE applications (`ejbserver.deploy.context.check_interval`) is used. Note that when 0 is specified in the update detection interval for J2EE applications, the updates cannot be detected for EJB applications.

For servlets

The update detection interval for the Web applications or J2EE applications is used. The priority is as follows:

1. Update detection interval for the Web applications (`webserver.context.check_interval`)
2. Update detection interval for J2EE applications (`ejbserver.deploy.context.check_interval`)

Note that when the update detection interval for the Web applications is not specified, the update detection interval for J2EE applications is used.

Also, when 0 is specified in the update detection interval for the Web applications, the updates are not detected for the servlets.

For JSPs

The update detection interval for the JSPs or J2EE applications is used. The priority is as follows:

1. Update detection interval for the JSPs (`webserver.jsp.check_interval`)
2. Update detection interval for the J2EE applications (`ejbserver.deploy.context.check_interval`)

Note that when the update detection interval for the JSPs is not specified, the update detection interval for J2EE applications is used.

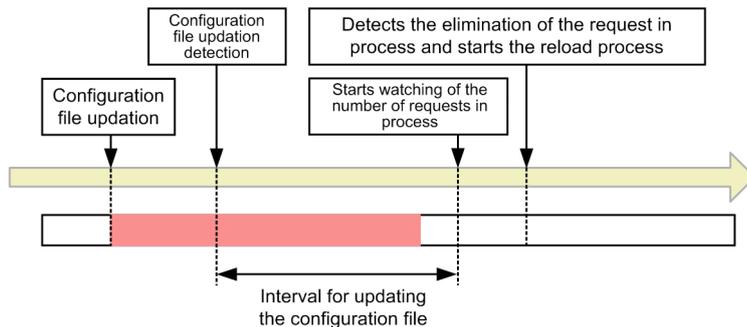
Also, when 0 is specified in the update detection interval for the JSPs, the updates are not detected for the JSPs.

13.8.7 Interval for updating the J2EE application configuration file

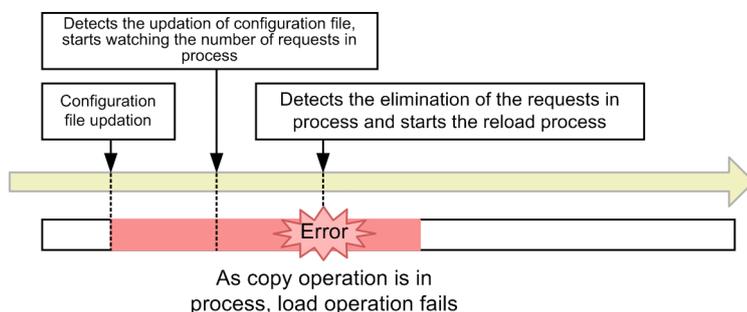
When the J2EE server detects a configuration file update, the J2EE server starts monitoring the number of requests being processed, and loads the file when the processing of the requests is complete. If the file size is large, when the file passes through a network, and when there are multiple files, the processing time to copy the files might be longer. At this time, if the request processing finishes before the copying of the file is complete, the loading process starts while the file is being copied, and hence, the loading of the file might fail. To avoid this, you can consider the time taken to copy the files and set the interval for configuration file update in such a way that the loading starts after the copy operation is complete. With the interval for configuration file update, you set the time from the detection of a configuration file update to the start of monitoring for the number of requests being processed. The following figure shows the interval for configuration file update.

Figure 13-12: interval for configuration file update

When setting up an interval for updating the configuration file



When interval for updating the configuration file is not set



Legend:

: Time required for copying the configuration file

Tip

- We recommend that you set the interval for configuration file update considering some leeway in the actual time required for copying.
- When you update multiple files concurrently, calculate the time taken to update multiple files and then set the interval for configuration file update.

You can set the interval for configuration file update for J2EE applications, Web applications, and JSPs respectively. The relationship between the set values is as follows:

For EJB applications

The interval for updating the J2EE application configuration file (`ejbserver.deploy.context.update.interval`) is used.

For the servlets

The interval for updating the configuration files of the Web applications or J2EE applications is used. The priority is as follows:

1. Interval for updating the Web application configuration file (`webserver.context.update.interval`)
2. Interval for updating the J2EE application configuration file (`ejbserver.deploy.context.update.interval`)

When interval for updating the Web application configuration file is not specified, the interval for updating the J2EE application configuration file is used.

For the JSPs

The interval for updating the configuration files of the JSPs or J2EE applications is used. The priority is as follows:

1. Interval for updating the JSP configuration file (`webserver.jsp.update.interval`)

2. Interval for updating the J2EE application configuration file
(`ejbserver.deploy.context.update.interval`)

When the interval for updating the JSP configuration file is not specified, the interval for updating the J2EE application configuration file is used.

13.8.8 Reloading the Web applications

When the Web applications are reloaded, you can use the delayed reloading functionality and minimize the period for which the J2EE application service is stopped. Also, with the reloading of the Web applications, the session information generated before the application is reloaded is inherited and can be used even after reloading. This subsection describes the delayed reloading of the Web applications and the inheritance of the session information.

(1) Delayed reloading of the Web applications

By default, if a configuration file update is detected, the reload processing is executed as follows:

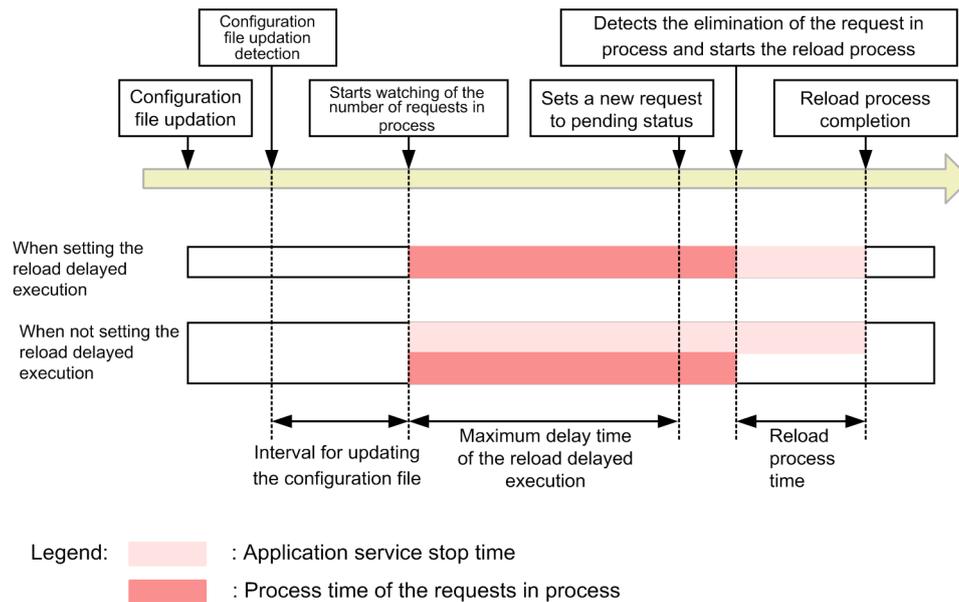
1. The request processing of the Web application is blocked.
If a configuration file update is detected and the time specified in the interval for configuration file update lapses, the monitoring of the requests being processed starts and the requests are processed as follows:
 - New request: Pending.
 - Request that is being processed: Processing continues.
2. When the processing of the requests being processed is complete, the reload process is executed.

In this case, the longer the processing time of requests being processed, the greater is the delay in starting the processing of new requests, and the period during which the application service is stopped for new requests becomes longer. To avoid this, we recommend that you set up delayed reloading. If you use the delayed reloading functionality, the pending time of requests being processed reduces and the period for which the application service is stopped is minimized.

With the delayed reloading functionality, you set whether this functionality is to be used, and the time until the reload processing is started (maximum delay time). If you use the delayed reloading functionality, new requests can be received until the processing of the request being processed is complete. When the number of requests being processed becomes 0, the reload processing starts and the new requests are added to the pending list. However, whether the number of requests being processed becomes 0 depends on the access state. When the requests are continuous, the reload processing cannot be started. In such cases, specify the maximum delay time from the detection of a configuration file update until a new request is added to the pending list. If the maximum delay time lapses, the subsequent new requests are added to the pending list and the reloading starts after the processing of the request being processed is complete.

The following figure shows the difference in the periods for which the application service is stopped when delayed reloading is set and when delayed reloading is not set.

Figure 13-13: Differences in the periods for which the application service is stopped



When the maximum delay time is set for delayed reloading, if the configuration file update is detected and the time specified for the interval for configuration file update lapses, the monitoring of the requests that are being processed and the counting of the maximum delay time for delayed reloading starts. For example, when the maximum delay time is set as five minutes, the time when the reload processing starts differs depending upon the access state as follows:

- When the requests being processed become 0 before the maximum delay time lapses
When the requests being processed become 0 three minutes after the start of the counting of the maximum delay time, the reload processing starts at that point. Also, any new request is added to the pending list.
- When the maximum delay time lapses
When the number of requests being processed does not become 0 because there are continuous requests, any new request is added to the pending list five minutes after the start of the counting of the maximum delay time. The reload processing starts after the processing of the requests being processed is complete.

Reference note

If the processing of the request being processed is not complete, you can start the reload processing by executing the method `timeout` and method `cancellation` functionality for monitoring the J2EE application execution time. For details on the monitoring of the J2EE application execution time, see [13.8.10\(2\) Relationship between reloading and monitoring the J2EE application execution time](#).

(2) Inheriting the session information when the Web applications are reloaded

When the Web applications are reloaded, the session information generated before the execution of reloading is inherited and used even after reloading.

(a) Session information file

When the Web applications are reloaded, the session information (objects registered in the `javax.servlet.http.HttpSession` object) is serialized and is output to the *session information file*. The session information is read from the session information file and is de-serialized on the class loader used after the reloading process. Note that the time required to serialize and de-serialize depends on factors such as the number of sessions generated on the Web application and the size of objects registered in the `javax.servlet.http.HttpSession` object.

! Important note

When the classes of the objects registered in the session to be serialized, and the classes of the objects referenced from such classes are updated in an un-deserializable configuration and reloading is executed, the deserialization of the session fails. If deserialization fails, all the session information on the Web application is destroyed.

The session information file is created when the reload processing starts and is deleted when the reload processing ends. The session information file is created even if the session does not exist when the reloading process is executed.

The session information file contains information that is auto-generated by Web container (2 kilobytes) and the session information. The session information depends on the number of sessions and the session information (objects) registered in the session.

Check the size of the session information file from the serialize completion message that is output during reloading and calculate the size based on the number of sessions. An example of a serialize completion message is as follows:

```
KDJE39168-I The serialization of session objects has finished. (J2EE application = appl,
context root = /examples, number of sessions = 10, session information file size(byte) =
12345)
```

In this example of output, there are 10 sessions for context root name 'examples' on the Web application 'app1' and the size of session information file is 12,345 bytes.

(b) Notes on inheriting the session information

- If you are using the session inheritance functionality for reloading, the session information registered in `javax.servlet.http.HttpSession` must be a serializable object.
- The objects that are referenced from the objects registered in the `HttpSession` object must either be declared as `transient` or must be serializable. The same applies to the objects to be referenced hereafter. For details on the serializable objects, see the J2SE specifications.
- An error occurs when an object that cannot be serialized is registered as the session information. The error details differ depending upon the object.
 - **For objects registered in `javax.servlet.http.HttpSession`**
If the objects registered in the `HttpSession` object cannot be serialized, only the relevant object is destroyed. The destroyed objects cannot be used after the reloading process is complete. Even if that object implements the `javax.servlet.http.HttpSessionBindingListener` interface, the Web container unbind event is not reported.
 - **For the objects referenced from the objects registered in `javax.servlet.http.HttpSession`**
If the objects referenced from the objects registered in `HttpSession` cannot be serialized, all the session information is destroyed. Similarly, all the session information is destroyed when the objects referenced from the objects registered in `HttpSession` are the objects registered in `HttpSession`.

13.8.9 Reloading the JSPs

If a JSP is updated, the update is detected when JSP is recompiled or through the monitoring of the class files and the JSP is reloaded. The JSP reloading methods are as follows:

- **Reloading by re-compiling JSPs**
The Web container checks whether the loaded JSP files, tag files, or the files on which the JSP or tag files depend are updated, and if the date and time of update is different from the loading time, the Web container recompiles and reloads the JSPs.
After an update is detected and the interval for detecting the update of the JSP configuration files lapses, the files are re-compiled. When compilation is complete and when the Web container detects that there are no more requests being processed, the reload processing starts.
- **Reloading by monitoring the class files**
The Web container checks whether the class files generated from the JSP files loaded in the Web container are updated, and if the date and time of update is different from the loading time, the Web container reloads the JSPs.
After an update is detected and the interval for updating the JSP configuration file lapses, the reload processing starts when the Web container detects that there are no more requests being processed.
The class files are updated in one of the following cases:
 - When the JSP pre-compile command is executed and the class files are generated by compiling the JSP files
 - When the class files compiled by the JSP pre-compile functionality are copied and over-written in the JSP working directory

For details on the JSP pre-compile functionality, see *2.5 JSP pre-compile functionality and the storage of the compilation results* in the *uCosminexus Application Server Web Container Functionality Guide*.

The following table describes the differences between the JSP reloading methods.

Table 13–16: Differences between the JSP reloading methods

Items	For reloading by re-compiling JSPs	For reloading by monitoring the class files
Usage of the JSP pre-compile functionality	The JSP pre-compile functionality is disabled.	The JSP pre-compile functionality is enabled.
Target files for update detection [#]	<ul style="list-style-type: none"> • JSP files • Tag files • Files on which the JSP files or tag files depend 	<ul style="list-style-type: none"> • Class files generated by the JSP pre-compile functionality
Processing after the update detection	The files are compiled and reloaded.	The class files are reloaded.

[#]: The updates are not detected even if you update the files that are not loaded into the Web container or the files for which updates are not to be detected.

In both the reloading methods, you can specify the update detection interval for JSPs and the interval for updating the JSP configuration file to detect the file updates. For details on the update detection interval, see *13.8.6 Update detection interval for J2EE applications*. For details on the interval for configuration file update, see *13.8.7 Interval for updating the J2EE application configuration file*.

! Important note

Time at which the loaded JSPs are destroyed and the monitoring targets of the JSP reloading functionality

If you execute the Web application reloading functionality when the Web application reloading functionality is used together with the JSP reloading functionality, the loaded JSPs are destroyed. Only the JSP files, loaded when the Web application reloading functionality is executed, are monitored for the JSP reloading functionality.

When you restart the Web container and Web applications, the loaded JSPs are destroyed. Also, only the JSP files loaded after the Web application starts are monitored for the JSP reloading functionality.

13.8.10 Relationship with the other functionality

This section describes the relationship between the reloading functionality and the following functionality:

- Controlling the number of concurrently executed threads
- Monitoring the J2EE application execution time

(1) Relationship between reloading and controlling the concurrently executed threads

This subsection describes the relationship between the addition of the new requests to the pending list and the controlling of the concurrently executed threads in the Web application reloading functionality. For details on controlling the concurrently executed threads, see *2.15 Overview of controlling the concurrently executed threads* in the *uCosminexus Application Server Web Container Functionality Guide*.

(a) When the concurrently executed threads are controlled for the Web applications or URL groups

The new requests that are pending with the reloading functionality are controlled separately from the pending queue for controlling the concurrently executed threads for the Web applications or URL groups.

• **When the concurrently executed threads are controlled for the Web applications or URL groups**

When reloading the Web applications in which the concurrently executed threads are controlled for the Web applications or URL groups, the new requests pending with the reloading functionality are controlled as follows:

- The new requests that are pending with the reloading functionality are not registered in the pending queue for the Web applications or URL groups. The new request remains pending regardless of the maximum pending queue size.

- When there are requests registered in the pending queue for the Web applications or URL groups, the reloading process starts after the processing of all the requests registered in the pending queue is complete.
- After the reload processing is complete, the processing of the new requests pending with the reloading functionality starts, but the requests exceeding the number of concurrently executed threads are registered in the pending queue for the Web applications or URL groups. The requests that exceed the maximum pending queue size return the status code 503 (Service Unavailable) error.
- **When the concurrently executed threads are not controlled for the Web applications**
When reloading the Web applications in which the concurrently executed threads are not controlled for the Web applications, the new requests pending with the reloading functionality are controlled as follows:
 - The new requests that are pending with the reloading functionality are not registered in the default pending queue. The new request remains pending regardless of the maximum size of the default pending queue.
 - When there are requests registered in the default pending queue, the reloading process starts after the processing of all the requests registered in the pending queue is complete.
 - After the reload processing is complete, the processing of the new requests pending with the reloading functionality starts, but the requests exceeding the number of concurrently executed threads are registered in the default pending queue. The requests that exceed the maximum size of the default pending queue return the status code 503 (Service Unavailable) error.

(b) When the concurrently executed threads are controlled for the Web containers

When the concurrently executed threads are controlled for Web containers, the processing threads of the new requests that can now be executed using the concurrently executed thread control for Web containers, are added to the pending list, after the reload processing starts.

Therefore, when the number of new requests pending with the reloading functionality reaches the maximum number of concurrently executed threads for the Web containers, the requests to the Web applications, other than the Web applications for which the reload processing was started, are also added to the pending list by the functionality for controlling the concurrently executed threads for Web containers.

(2) Relationship between reloading and monitoring the J2EE application execution time

When the processing of the requests being processed does not finish, you can specify the `-t` option with the `cjreloadapp` command to forcibly terminate the requests being processed and start the reload processing. However, the requests being processed can be forcibly terminated only by those methods in which the method timeout for monitoring the J2EE application execution time is applicable. Also, when method cancellation is set to operate after the method timeout occurs, the requests being processed are cancelled because of the timeout and the reloading process starts.

For details on the monitoring of the J2EE application execution time, see *5.3 Monitoring and cancelling the J2EE application execution time* in the *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*. For details on the settings for monitoring the J2EE application execution time, see *13.8.12 Settings for detecting updates and reloading the J2EE applications*.

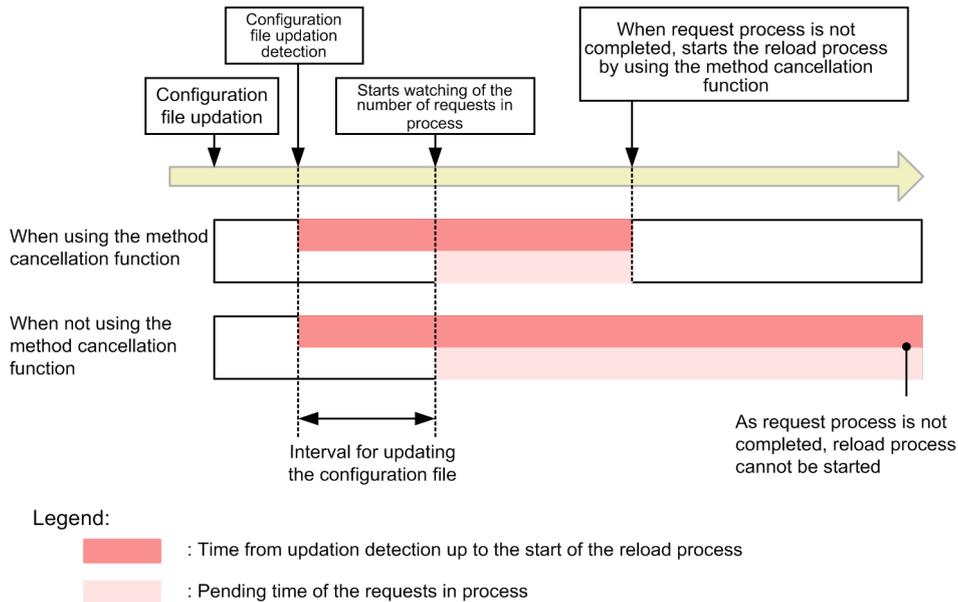
(a) For the EJB applications

This point describes the reloading of the EJB applications.

If the interval for configuration file update lapses, the J2EE server waits for the processing of the requests being processed to finish and then starts the reload processing. If the processing of requests being processed is not over at this point, the method processes not terminated in a fixed time are reported as timed out during the monitoring of the J2EE application execution time, the method processing is cancelled, and then the reload processing can be started.

The following figure shows the process of reloading with the method cancellation functionality.

Figure 13-14: Reloading with the method cancellation functionality (For the EJB applications)



You obtain the time from the detection of updates in the configuration file to the start of the reload processing with the following formula:

$$Interval-for-configuration-file-update-(seconds) + Pending-time-of-requests-being-processed-(seconds)$$

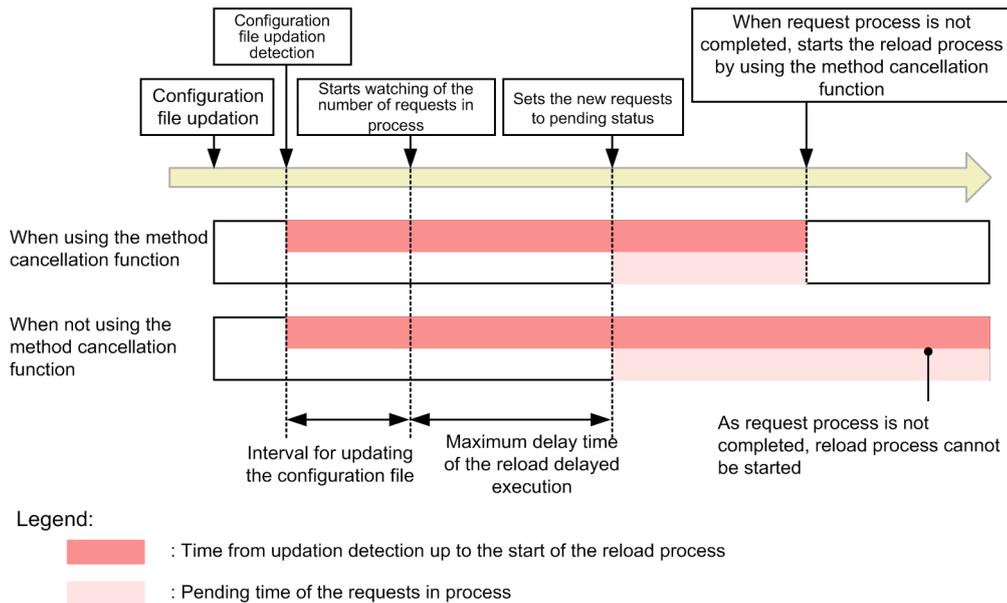
(b) For the Web applications

This point describes the reloading of the Web applications.

If delayed reloading is specified and if the maximum specified delay time lapses, the J2EE server sets the new requests to a pending state, waits for the processing of the requests being processed to finish, and then starts the reload processing. If the processing of requests being processed is not over at this point, the method processes not terminated in a fixed time are reported as timed out during the monitoring of the J2EE application execution time, the method processing is cancelled, and then the reload processing can be started.

The following figure shows the process of reloading with the method cancellation functionality.

Figure 13-15: Reloading with the method cancellation functionality (For the Web applications)



You obtain the time from the detection of updates in the configuration file to the start of the reload processing with the following formula:

$$\text{Interval-for-configuration-file-update-(seconds)} + \text{Maximum-delay-time-for-delayed-reloading-(seconds)} + \text{Pending-time-of-requests-being-processed-(seconds)}$$

13.8.11 Reloading the J2EE applications using commands

This subsection describes how to replace the J2EE applications by reloading.

Reload is a functionality with which you can replace an exploded archive-format J2EE application with fewer steps. With the replacement of J2EE applications by reloading, operations such as stopping and deleting an existing J2EE application, and archiving, importing, and restarting the replaced J2EE application, become unnecessary. This functionality is especially effective in system operations requiring frequent maintenance because you can update J2EE applications only by updating and reloading the class files.

Note that settings must be specified in advance for replacing J2EE applications by reloading. For details on how to specify the settings, see *13.8.12 Settings for detecting updates and reloading J2EE applications*.

You can use the server management commands (`cjreloadapp` command) to replace J2EE applications by reloading. For details on the `cjreloadapp` command, see *10.5.2 Applications in the exploded archive format in the uCosminexus Application Server Application Setup Guide*.

To reload:

1. Edit or create a Java source file according to the maintenance contents, and compile into a class file.
2. Reload the J2EE application.

Execute the `cjreloadapp` command. The format and example of execution is as follows.

Format of execution

```
cjreloadapp J2EE-server-name -name J2EE-application-name
```

Example of execution

```
cjreloadapp MyServer -name Appl
```

! Important note

To delete an application that could not be reloaded, stop and delete the application after it is successfully reloaded, or delete the application after restarting the J2EE server.

13.8.12 Settings for detecting updates and reloading the J2EE applications

If you want to replace a modified J2EE application and a running J2EE application when testing during application development and during system operations, you can replace the applications by using the reload functionality. When a file configuring an exploded archive-format J2EE application is updated, you can reload the updated J2EE application by using update detection and command execution. By using the reload functionality, the J2EE application is replaced dynamically using fewer steps.

This subsection describes the settings for detecting updates and reloading the exploded archive-format J2EE applications. For details on the commands and file keys, see the *uCosminexus Application Server Command Reference Guide* and *uCosminexus Application Server Definition Reference Guide*.

The settings required for reloading an exploded archive-format J2EE application differ depending on whether the application is reloaded by detecting the configuration file updates or whether the application is reloaded by executing commands. The following table describes the settings required for reloading the exploded archive-format J2EE applications.

Table 13-17: Settings required for reloading

Settings	To reload by detecting updates	To reload by using commands
Setting the scope for the reload functionality	Y	Y
Setting the update detection interval	Y	--
Setting the interval for configuration file update	Y	--
Settings for delayed reloading	Y	--
Changing the directory for storing the session information file	R	R
Settings for JSP pre-compilation	R	R
Settings for monitoring the J2EE application execution time	R	R

Legend:

- Y: Settings are specified
- R: Settings are specified as and when required
- : Not applicable

(1) Setting the scope for the reload functionality

Specify the settings for the scope of the reload functionality in the `<configuration>` tag of the logical J2EE server (`j2ee-server`) in the Easy Setup definition file.

- `ejbserver.deploy.context.reload_scope`
Specifies whether the reload functionality will be used and the reload targets.
 - `app`: The EJB applications (EJB-JARs) and Web applications (WARs) are reloaded.
 - `web`: Only the Web applications (WARs) are reloaded.
 - `jsp`: Only the JSPs are reloaded.
 - `none`: The reload functionality is not used.

By default, the reload functionality is disabled. To use the reload functionality, you must enable the reload functionality and set the scope.

Note that the enabling or disabling of the reload functionality is decided based upon the combination of the scope of the local call optimization functionality specified in the `ejbserver.rmi.localinvocation.scope` parameter of `usrconf.properties` and the scope of the reload functionality. For details on the mapping between the scope of the local call optimization functionality and the scope of the reload functionality, see *13.8.2 Scope of reloading*.

(2) Setting the update detection interval

Specify the settings for the update detection interval in the `<configuration>` tag of the logical J2EE server (`j2ee-server`) in the Easy Setup definition file. The following table describes the settings for the update detection interval specified in the Easy Setup definition file.

Table 13-18: Settings for the update detection interval in the Easy Setup definition file

Specified parameters	Settings
<code>ejbserver.deploy.context.check_interval</code>	Specify the interval (seconds) for monitoring the J2EE application configuration file and detecting the updates.
<code>webserver.context.check_interval</code>	Specify the interval (seconds) for monitoring the Web application configuration file and detecting the updates.
<code>webserver.jsp.check_interval</code>	Specify the interval (seconds) for monitoring the JSP configuration file and detecting the updates.

The relationship between the setup values of the update detection interval is as follows:

For EJB applications

The value in `ejbserver.deploy.context.check_interval` is used. Note that if 0 is specified in `ejbserver.deploy.context.check_interval`, the updates are not detected for EJB applications.

For the servlets

The value in `webserver.context.check_interval` or `ejbserver.deploy.context.check_interval` is used. The priority is as follows:

1. Value in `webserver.context.check_interval`
2. Value in `ejbserver.deploy.context.check_interval`

Note that if `webserver.context.check_interval` is not specified, the value in `ejbserver.deploy.context.check_interval` is used.

Also, if 0 is specified in `webserver.context.check_interval`, the updates are not detected for the servlets.

For the JSPs

The value in `webserver.jsp.check_interval` or `ejbserver.deploy.context.check_interval` is used. The priority is as follows:

1. Value in `webserver.jsp.check_interval`
2. Value in `ejbserver.deploy.context.check_interval`

Note that if `webserver.jsp.check_interval` is not specified, the value in `ejbserver.deploy.context.check_interval` is used.

Also, if 0 is specified in `webserver.jsp.check_interval`, the updates are not detected for the JSPs.

(3) Setting the interval for configuration file update

Specify the interval for configuration file update in the `<configuration>` tag of the logical J2EE server (`j2ee-server`) in the Easy Setup definition file. The following table describes the settings for the configuration file update interval specified in the Easy Setup definition file.

Table 13-19: Settings for the configuration file update interval in the Easy Setup definition file

Specified parameters	Settings
<code>ejbserver.deploy.context.update.interval</code>	Specify the time (seconds) required for copying the J2EE application configuration file.
<code>webserver.context.update.interval</code>	Specify the time (seconds) required for copying the Web application configuration file.
<code>webserver.jsp.update.interval</code>	Specify the time (seconds) required for copying the JSP configuration file.

The relationship between the setup values of the interval for configuration file update is as follows:

For EJB applications

The value in `ejbserver.deploy.context.update.interval` is used.

For the servlets

The value in `webserver.context.update.interval` or `ejbserver.deploy.context.update.interval` is used. The priority is as follows:

1. Value in `webserver.context.update.interval`
2. Value in `ejbserver.deploy.context.update.interval`

If `webserver.context.update.interval` is not specified, the value in `ejbserver.deploy.context.update.interval` is used.

For the JSPs

The value in `webserver.jsp.update.interval` or `ejbserver.deploy.context.update.interval` is used. The priority is as follows:

1. Value in `webserver.jsp.update.interval`
2. Value in `ejbserver.deploy.context.update.interval`

If `webserver.jsp.update.interval` is not specified, the value in `ejbserver.deploy.context.update.interval` is used.

(4) Settings for delayed reloading

Specify the settings for delayed reloading in the following parameter in the `<configuration>` tag of the logical J2EE server (`j2ee-server`) in the Easy Setup definition file:

- `webserver.context.reload_delay_timeout`
Specify whether delayed reloading will be used when the Web application is reloaded. Also, when delayed reloading is used, specify the maximum delay time (seconds) when you specify the time until the reload processing starts.

(5) Changing the output destination of the session information file

When the Web applications are reloaded, the session information generated before reloading is inherited and used even after reloading. The session information is output to the session information file.

Output destination of the session information file

By default the session information file is output to the following locations:

- In Windows
`Cosminexus-installation-directory\CC\server\repository\server-name\web\context-root-name\cjwebsession.dat`
- In UNIX
`/opt/Cosminexus/CC/server/repository/server-name/web/context-root-name/cjwebsession.dat`

Directory name for the Web applications

The name of the directory for the Web applications follows the rules based on the context root name. If the context root name contains a forward slash (/), dollar sign (\$), percent sign (%), and plus sign (+), these signs are converted to the following characters:

Character before conversion	Character after conversion
/	\$2f
\$	\$24
%	\$25
+	\$2b

The context root name directory that forms the output destination is created when the Web application starts. However, if the context root is the root context, the context root of the output destination directory is created as \$2f. The created directory is deleted when the Web application is terminated.

To change the output destination of the session information file, specify the settings in the Easy Setup definition file. Specify the following parameter in the `<configuration>` tag of the logical J2EE server (`j2ee-server`):

- `ejbserver.deploy.session.work.directory`
Specify the output destination of the session information file for the Web applications.
A directory named `web\context-root-name` is created beneath the directory specified in the `ejbserver.deploy.session.work.directory` parameter and the session information file is output beneath that directory.

(Example) Example of settings for the output destination of the session information file

```
<configuration>
<logical-server-type>j2ee-server</logical-server-type>
<param>
<param-name>ejbserver.deploy.session.work.directory</param-name>
<param-value>C:\tmp\session_work</param-value>
</param>
:
</configuration>
```

In this example, the session information file of a Web application with the context root name 'examples' is output to the following location:

```
C:\tmp\session_work\web\examples\cjwebsession.dat
```

(6) Settings for JSP pre-compilation

Even when a class file generated from a JSP file using JSP pre-compile is updated, the update of the J2EE application configuration file is detected and the reload functionality is executed.

When you execute the JSP pre-compile functionality by specifying the `-jspc` option in the `cjstartapp` command, you must set up the JSP working directory using `usrconf.properties`. For details on the JSP pre-compile settings, see *2.5.8 Settings in the execution environment (J2EE server settings)* in the *uCosminexus Application Server Web Container Functionality Guide*.

Note that when you use the `cjjspc` command to execute JSP pre-compile during application development, specify the details such as the JSP working directory in the command options. For details on using the `cjjspc` command for JSP pre-compilation, see *2.5 JSP pre-compile functionality and the storage of the compilation results* in the *uCosminexus Application Server Web Container Functionality Guide*.

(7) Settings for monitoring the J2EE application execution time

When a configuration file update is detected, the reload processing starts when the processing of the requests being processed is complete, but if the processing of requests being processed is not over at this point, you can start the reload processing by implementing the method timeout and method cancellation functionality for the monitoring of the J2EE application execution time.

As and when required, you specify the settings for monitoring the J2EE application execution time. For details on the settings for monitoring the J2EE application execution time, see *5.3.9 Settings in the execution environment* in the *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*.

13.8.13 Notes and restrictions related to reloading

The notes and restrictions related to reloading are as follows:

(1) Notes related to the usage of the reload functionality

- **Deleting the J2EE application**

To delete a J2EE application that has failed to reload, use one of the following methods:

- Re-execute reload using update detection or through the `cjreloadapp` command. If the reloading process is successful, stop and then delete the J2EE application.
- Delete the J2EE application after the J2EE server is restarted.

Re-import the J2EE application after deletion.

- **When the memory is insufficient**

With reloading, a considerable amount of memory is consumed (the volume of memory consumed depends on the implementation of J2EE applications) because the class is loaded again after re-creating the class loader. Therefore, for example, if the reload processing is repeated simultaneously with multiple Web applications, memory might become insufficient. Also, the time when the resources that are no longer required due to reloading are released from JavaVM, depends on the garbage collection time of JavaVM. If you repeat reloading during high load, such as when the requests are at a peak, there is no time for garbage collection and the insufficient memory problem is more likely to occur.

Therefore, avoid updating the resources when the load is high, such as at the peak of requests.

- **Memory size of the Permanent area**

When you use the reloading function, if you try to load a new class to the Permanent area with no free space, a `java.lang.OutOfMemoryError` occurs. Therefore, before reloading, be sure to reserve more free space in the Permanent area than is needed by the application for the update. If there is not enough free space, the Permanent area might be too small. Reconfigure the maximum value for the Permanent area.

When reloading, the application class loader is discarded and a new one is generated. However, depending on the specifications of JavaVM and the implementation of the class loader, the discarded class loader might not be released immediately. As a result, after reloading, a larger area of the Permanent area than actually required by the application is temporarily occupied. Therefore, when estimating the Permanent area required by the reloading function, we recommend assuming *Permanent-area-required-by-application* (see (4) below) that is three times larger than the actual size.

- Estimating the Permanent area required by the reloading function

Permanent-area-required-by-J2EE-server (1) is estimated by the following expression:

$$\begin{aligned} & \textit{Permanent-area-required-by-J2EE-server} \text{ (1)} \\ &= \textit{Permanent-area-required-by-container} \text{ (2)} \\ &+ \textit{Permanent-area-required-by-container-extension-library} \text{ (3)} \\ &+ \textit{Permanent-area-required-by-application} \text{ (4)} \times 3 \\ &+ \textit{Permanent-area-required-by-JDK} \text{ (7)} \end{aligned}$$

Permanent-area-required-by-container (2) is estimated at 120 MB.

Permanent-area-required-by-container-extension-library (3) is estimated by the following expression:

$$\begin{aligned} & \textit{Permanent-area-required-by-container-extension-library} \text{ (3)} \\ &= \textit{sum-of-sizes-of-.class-files-in-container-extension-library} \end{aligned}$$

Permanent-area-required-by-JDK (7) is estimated at 90 MB.

Permanent-area-required-by-application (4) is the size of the `.class` files for the classes loaded into the class loader.

However, when the `.class` files are compressed into a JAR file, the size must be estimated using the size after decompression. Additionally, if the application includes JSP, you must compile the JSP in advance and create the `.class` files in order to estimate the size.

$$\begin{aligned} & \textit{Permanent-area-required-by-application} \text{ (4)} \\ &= \textit{Permanent-area-of-class-created-by-container} \text{ (5)} \\ &+ \textit{size-of-J2EE-application-.class-file-created-by-user} \text{ (6)} \\ &+ \textit{size-of-library-JAR-and-reference-library-.class-file-created-by-user} \text{ (8)} \end{aligned}$$

Permanent-area-of-class-created-by-container (3) is estimated by the following expression:

$$\begin{aligned} & \textit{Permanent-area-of-class-created-by-container} \text{ (5)} \\ &= \textit{Permanent-area-after-starting-application} \\ &- \textit{Permanent-area-before-registering-application} \end{aligned}$$

Note: To perform these calculations, you must actually start the J2EE server and check the Permanent areas.

size-of-J2EE-application-.class-file-created-by-user (6) is estimated by the following expression:

$$\begin{aligned} & \textit{size-of-J2EE-application-.class-file-created-by-user} \text{ (6)} \\ &= \textit{size-of-.class-files-comprising-J2EE-application} \\ &+ \textit{size-of-.class-files-in-JAR-files} \\ &+ \textit{size-of-.class-files-created-by-prior-JSP-compilations} \end{aligned}$$

Note: For this calculation, you do not need to start the J2EE server because this J2EE application is created by the user.

size-of-library-JAR-and-reference-library-.class-file-created-by-user (8) is estimated by the following expression:

$$\begin{aligned} & \textit{size-of-library-JAR-and-reference-library-.class-file-created-by-user} \text{ (8)} \\ &= \textit{size-of-.class-files-in-library-JAR-file} \\ &+ \textit{size-of-.class-files-in-reference-library-files} \end{aligned}$$

- Example of estimating the Permanent area required by the reloading function

As an example, here you change the maximum values of the Permanent area. Assume the following conditions:

Permanent-area-required-by-container (2) is 120 MB.

Permanent-area-required-by-container-extension-library (3) is 16 MB.

Permanent-area-required-by-application (4) is 32 MB.

Permanent-area-required-by-JDK (7) is 90 MB.

Then, *Permanent-area-required-by-J2EE-server* (1) is:

$$120 + 16 + (32 \times 3) + 90 = 322 \text{ MB}$$

Therefore, if you use the reloading function, you must change the default maximum value of the assigned Permanent area as follows:

```
add.jvm.arg=-XX:MaxPermSize=322m
```

How to check the Permanent area

To check the memory size of the Permanent area, either output the JavaVM log file, or use the resource depletion monitoring functionality.

- **Executing method cancellation**

If the processing of the request being processed is not complete, you can start the reload processing by executing the method timeout and method cancellation functionality for monitoring the J2EE application execution time. If the processing cannot be cancelled even after method cancellation is executed, restart the J2EE server.

- **Restarting the processing of pending requests**

If an EJB application cannot be started or stopped with the reloading of the J2EE application, the new requests, after the reloading of the Web applications starts, remain pending. To restart the pending request processing, update and reload the EJB application and implement the reload processing successfully. Reloading is not executed even if the Web application is updated.

- **RAR files included in J2EE applications**

If the application directory contains a RAR file, the reload functionality is not executed for the RAR file. If the RAR file is updated, replace the J2EE application containing the updated RAR file by using the server management commands. For details on the procedure of replacing J2EE applications by using the server management commands, see *5.6 Replacing J2EE applications* in the *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*.

- **Reloading the J2EE applications that use the CDI functionality**

If the CDI functionality is included in the J2EE application components (EJB-JAR, Library JAR, and WAR), the J2EE applications cannot be reloaded by update detection. To execute reloading, use the `cjreloadapp` command. If the settings for reloading by detecting updates are enabled, the KDJE42393-W message is output when the J2EE application using the CDI functionality is started, and reports that update detection-based reloading is disabled.

To reload the J2EE applications that use the CDI functionality, specify `app` in the `ejbserver.deploy.context.reload_scope` parameter in the `<configuration>` tag of the logical J2EE server (`j2ee-server`) in the Easy Setup definition file. If the reloading process is executed, all the J2EE applications are reloaded (the reloading process cannot be executed for the WARs or JSPs) even if the updated file is a WAR file or JSP file. At this time, if the updated file is a WAR file or JSP file, KDJE42395-I is output reporting that all the J2EE applications are reloaded.

When you execute the `cjreloadapp` command to reload the J2EE applications that use the CDI functionality, if `web` or `jsp` is specified in the `ejbserver.deploy.context.reload_scope` parameter in the `<configuration>` tag of the logical J2EE server (`j2ee-server`) in the Easy Setup definition file, the KDJE42394-E message is output and the reloading process fails.

The following table lists the execution results when the `cjreloadapp` command is executed to reload the J2EE applications that use the CDI functionality.

Table 13-20: Execution results of the `cjreloadapp` command (J2EE applications that use the CDI functionality)

Components containing the updated file	Value of the <code>ejbserver.deploy.context.reload_scope</code> parameter	Execution results of the <code>cjreloadapp</code> command
EJB-JAR or Library JAR	<code>app</code>	All the J2EE applications are reloaded.
	<code>web</code>	KDJE42394-E is output and the command execution fails.
	<code>jsp</code>	
WAR	<code>app</code>	KDJE42395-I is output and all the J2EE applications are reloaded.
	<code>web</code>	KDJE42394-E is output and the command execution fails.
	<code>jsp</code>	

Components containing the updated file	Value of the ejbserver.deploy.context.reload.scope parameter	Execution results of the cjreloadapp command
JSP	app	KDJE42395-I is output and all the J2EE applications are reloaded.
	web	KDJE42394-E is output and the command execution fails.
	jsp	

- **Inbound resource adapters**

Do not send messages from the Inbound resource adapter while an application is being reloaded.

- **Reloading the Message-driven Beans**

To reload the Message-driven Beans when you are using the TP1 inbound integrated function, stop the issuing of the RPC from OpenTP1 and then perform the reloading process. If the RPC is issued during the reloading process, the Message-driven Beans are considered as terminated and an error response is returned to the RPC issuing source.

(2) Notes related to EJB applications (EJB-JARs)

- To invoke the EJBs after reloading, you must obtain the EJBHome and business interface again with the client-side source code. When the EJB is accessed through the interface that is obtained before reloading, an exception is returned. An exception is also returned when the EJB is accessed during the reloading process.
- When a remote interface or remote business interface is replaced, you must obtain the stub with the client side again.
- If reloading is executed, the state of the session with the Stateful Session Bean client is lost.
- Note the following points when you use the asynchronous method with the reload functionality for the exploded archive-format J2EE applications:
 - When the asynchronous method is running, the reload processing stands by until the processing of the asynchronous method is complete.
 - The cleanup processing is executed with the reload processing; therefore, operations can no longer be executed using the `Future<V>` object obtained before the reload processing.
- Note the following points when you reload the exploded archive-format J2EE applications containing Singleton Session Bean:
 - Singleton Session Bean, for which the `@Startup` annotation is specified, is initialized during reloading, same as the initialization during application startup.
 - During the reload processing, the Singleton Session Bean instances in the J2EE application to be reloaded are destroyed by the container.
 - By specifying the definition in the `@DependsOn` annotation, you can change the dependency of Singleton Session Bean to be reloaded. However, an error does not occur during the reload processing even if there is a circular dependency. Therefore, in order to avoid a circular dependency with Singleton Session Bean to be reloaded, we recommend that you avoid using the `@DependsOn` annotation. The operations might not function properly if you use the `@DependsOn` annotation and there is a circular dependency.
 - The processing of threads waiting to obtain the `Write` lock is not cancelled by the reload processing.

(3) Restrictions related to EJB applications (EJB-JARs)

- Reloading cannot be performed by making changes to implement the `javax.ejb.TimedObject` interface with the EJB that has not implemented the `javax.ejb.TimedObject` interface so far. If the reloading process is performed, the following occurs:
 - J2EE application stops without terminating the update detection thread.
 - The stopped J2EE application is restarted when an update is detected.
- The applications using the CMR cannot be reloaded.

- If you change the value set for the @DependsOn annotation in Singleton Session Bean and then reload the bean, the bean cannot be reloaded if the specified dependency becomes a circular dependency.

(4) Notes related to the Web applications (WARs)

- The changes in the listener tag of the tag library descriptor file are not applied after reloading.
- The servlets and JSPs available for reloading can be used in the same state even after reloading. When the servlets and JSPs are permanently unavailable, the servlets and JSPs are permanently unavailable even after reloading. When the servlets and JSPs are temporarily unavailable, regardless of reload execution, the servlets and JSPs become available after a specified time period.
- When a file monitored for reloading in a Web application is deleted, the file is no longer monitored. Consequently, reloading cannot be executed by detecting updates even if the file is added once again. However, when a JAR file is added beneath /WEB-INF/lib, you can perform command-based reloading or reloading by update detection, regardless of whether that JAR file is added once again after deletion.
- When a J2EE application is stopped during the JSP reload processing or when a Web application is reloaded, the JSP reload processing is cancelled.
- If the log output level is Error when the JSP reload processing starts, the updated file names and the number of JSP files to be reloaded are output to the message KDJE39310-I. If the log output level is Warning, the names of all the JSP files to be reloaded are output to the message KDJE39312-I.
- When an exception occurs during the loading of a listener class or during instance generation, the Web container disables the listener in which the exception was thrown and continues the reload processing. To enable the listener in which the exception occurred, modify that listener and reload once again.
- When an exception occurs during the loading of a filter class, during instance generation, or during the execution of the init method, the Web container disables the filter in which the exception was thrown and continues the reload processing. To enable the filter in which the exception occurred, modify that filter and reload once again.
- The following table describes the methods of listener interface that are executed when the Web applications are reloaded.

Table 13-21: Listener methods executed when the Web applications are reloaded

Interface name	Method Name
javax.servlet.ServletContextListener	contextDestroyed()
	contextInitialized()
javax.servlet.ServletContextAttributeListener	attributeRemoved()
	attributeAdded()
javax.servlet.http.HttpSessionActivationListener [#]	sessionWillPassivate()
	sessionDidActivate()

[#]: This interface is not executed when the javax.servlet.http.HttpSession object does not exist.

- The reload processing time greatly depends on the implementation of the Web application. Therefore, note the following points:
 - The size of the session information greatly affects the processing time for serialization and deserialization. Specify the required size for the session information to be serialized.
 - With the reload processing, the termination and initialization processing of the Web application is executed. Note that when the termination and initialization processing is long, the reload processing time also increases and affects the period during which the Web application service is stopped. However, only those servlets and JSPs with the <load-on-startup> tag specified in web.xml, are initialized during the reload processing. The servlets and JSPs for which the <load-on-startup> tag is not specified are initialized during the first access after the completion of reloading.
- The Web application reloading functionality replaces the Web applications while the server is running by replacing the class loaders for the Web applications. Consequently, after reloading, a memory leak occurs if the class loader for the Web applications prior to reloading, or the classes loaded with the class loader for the Web applications are referenced.

The relevant cases are as follows. When you use the reload functionality, do not perform the following operations in the Web application:

- **Generating threads**

The threads store the context class loader, but, by default, store the context class loader of the parent thread. The threads for Web application execution use the class loader for the Web applications as the context class loader, so the generated threads use the class loader for the Web applications as the context class loader. Consequently, if the corresponding thread exists even after reloading, a memory leak occurs because the class loader for the Web applications prior to reloading is not released.

- **Generating the instances of the `java.lang.Thread` class or its derived classes**

When an instance of the `java.lang.Thread` class or its derived class (called *thread class* hereafter) is created, the instances of that thread class are referenced from the thread group that it belongs to. The reference from the thread group is released when the execution of the thread ends (the `run` method finishes). Due to this, when a thread class instance is generated inside the Web application and the `run` method is not executed, the instance of that thread class is not released. Also, the thread stores the class loader of the Web applications as the context class loader, so the class loader of the Web applications is not released and a memory leak occurs.

- **Using thread local variables**

The threads that process the requests are pooled with the Web container and are not terminated until Web container is terminated. Therefore, if an instance of a class included in the Web application is stored in a thread local variable, a memory leak occurs.

- If the JSP pre-compile functionality is used, the updates are detected for the files on which the JSP file or tag file depend and then excluded from update detection when one of the following conditions is fulfilled:
 - When the JSP files or tag files are updated in the class files that are changed so as not to use the dependent files, and there are no other JSP files or tag files using the dependent files
 - When either the JSP files or the class file generated from the files on which the JSP file depends are updated, and there are no other JSP files or tag files using the dependent files when an error occurs during the loading of the classes generated from the JSP file

If the relevant file is excluded from update detection targets, KDJE39319-I is output to the message log.

Note that if you execute the `cjjspc` command, the JSP files or tag files using the dependent files are also re-compiled. When you create the class files in a location other than beneath the exploded archive and update the files, update all the class files generated by using the `cjjspc` command.

- If the JSP pre-compile functionality is not used, the updates are detected for the files on which the JSP file or tag file depend and then excluded from update detection when one of the following conditions is fulfilled:
 - When JSP files or tag files are modified so as not to use the dependent files and when there are no other JSP files or tag files using the dependent files after successful compilation
 - When either the JSP files or the files on which the JSP files depend are updated, and there are no other JSP files or tag files using the dependent files when an error occurs during the JSP file compilation

If the relevant file is excluded from update detection targets, KDJE39318-I is output to the message log. Also, when the JSP file compilation error occurs and one of the following conditions is fulfilled, you can reload the JSPs without modifying the JSP file with the compilation error by modifying only the file excluded from the update detection target:

- When the dependent file is a tag file

The coded contents of the tag file and the coded contents of the file that uses the tag file are inconsistent. One of the following tag file attributes is invalid:

Attributes defined in the `tag` directive: `body-content` or `dynamic-attributes`

Attributes defined in the `attribute` directive: `name`, `required`, `fragment`, `rtexprvalue`, or `type`

Attributes defined in the `variable` directive: `name-given`, `variable-class`, `declare`, or `scope`

- When the dependent file is a file that is included implicitly in `<include-pragma>` or `<include-coda>` of `web.xml`

The coded contents of the implicitly-included file conflict with the coded contents of the JSP file to be included.
- When the dependent file is a file specified in the `include` directive

The coded contents of the file specified in the `include` directive conflict with the contents of the JSP file or tag file that specifies the `include` directive.

- When the dependent file is tag library descriptor (TLD) file

The coded contents of the TLD file conflict with the coded contents of the JSP file.

In this case, updates are no longer detected for the relevant file, so you must update the dependent files and then reload the JSPs by executing one of the following operations:

- Use a browser to access the JSP files using the dependent files.
- Update the date and time of the JSP files or tag files that use the dependent files.
- The updates are not detected for `implicit.tld`. The `implicit.tld` file is a TLD file indicating the tag file version defined in the Servlet 2.5 specifications. The `implicit.tld` file is re-read at the same time as when the JSP or tag file is reloaded.
- The annotation information is not read with the reloading of the Web applications. When you update only the annotation information defined in the Web application class, to apply that updated information, stop and then restart the J2EE application once.

13.9 WAR applications

You can specify and import WAR files or WAR directories with a J2EE server. The imported application is called a *WAR application*. You can deploy a WAR application as a J2EE application.

The following table lists the WAR application functionality and the reference locations.

Table 13-22: Organization of this section (WAR applications)

Functionality	Reference location
Support range of the operation commands for WAR applications	13.9.1
Executable WAR application formats	13.9.2
Replacing WAR applications	13.9.3
Names of WAR applications	13.9.4
Determining the context root	13.9.5
Reading the <code>cosminexus.xml</code> file	13.9.6

13.9.1 Support range of the operation commands for WAR applications

Among the operation commands that can be executed for a J2EE application, the operation commands that can be executed for a WAR application are limited.

The following table describes the support range of the operation commands for WAR applications.

Table 13-23: Support range of the operation commands for WAR applications

Operation command	Support	Execution results
<code>cjaddapp</code>	N	The KDJE42400-E message is output.
<code>cjchmodapp</code>	N	The KDJE42400-E message is output.
<code>cjdeleteapp</code>	P	If the command is executed by specifying <code>-type war</code> or <code>-type filter</code> , the KDJE42401-E message is output. In other cases, the result is the same as when the command is executed for a J2EE application.
<code>cjdeletelibjar</code>	N	--
<code>cjexportapp</code>	N	The KDJE42400-E message is output.
<code>cjgencmpsql</code>	N	--
<code>cjgetappprop</code>	Y	The result is the same as when the command is executed for a J2EE application.
<code>cjgetstubsjar</code>	N	--
<code>cjimportapp</code>	N	--
<code>cjimportwar</code>	Y	The WAR directory or WAR file is imported.
<code>cjimportlibjar</code>	N	The KDJE42400-E message is output.
<code>cjlistapp</code>	Y	The context root is displayed for a WAR application.
<code>cjreloadapp</code>	Y	The result is the same as when the command is executed for a J2EE application.
<code>cjrenameapp</code>	N	The KDJE42400-E message is output.
<code>cjreplaceapp</code>	P	<code>cosminexus.xml</code> cannot be read again [#] .

Operation command	Support	Execution results
<code>cjsetappprop</code>	Y	The result is the same as when the command is executed for a J2EE application.
<code>cjstartapp</code>	Y	The result is the same as when the command is executed for a J2EE application.
<code>cjstopapp</code>	Y	The result is the same as when the command is executed for a J2EE application.
<code>cjlistlibjar</code>	N	--
<code>cjlistpool</code>	N	--
<code>cjtestres</code>	N	--
<code>cjclearpool</code>	N	--

Legend:

- Y: Supported
- P: Partially supported
- N: Not supported
- : Not applicable

To change application attributes, use the server management commands (`cjgetappprop` and `cjsetappprop` commands).

■ Operation constraints for WAR applications

The following operations cannot be executed with WAR applications:

- Adding and deleting resources
 - Adding an EJB-JAR/ WAR/ RAR file to a WAR application
 - Deleting a WAR file from a WAR application
 - Adding a library JAR to a WAR application
 - Adding a filter to a WAR application
 - Deleting a filter from a WAR application
- Test mode
- Exporting applications
- Changing the application name

Reference note

To add or delete a resource, delete the WAR application, and then create a J2EE application for an EAR file.

13.9.2 Executable WAR application formats

You can import a WAR application in an archive format or an exploded archive format. For details on the application formats, see *13.2 Executable J2EE application formats*.

Archive-format WAR applications

Users who execute the `cjimportwar` command must have read permission for the WAR file to be imported.

Note that the WAR file name specified when you import the application is used as the directory name in the work directory. Specify the WAR file name such that the work directory path length does not reach the maximum limit for the platform. For details on estimating the work directory path length, see *Appendix C.1 Work directory of a J2EE server* in the *uCosminexus Application Server System Setup and Operation Guide*.

Exploded archive-format WAR applications

The exploded archive-format WAR applications are imported as exploded archive-format applications by creating a root directory called the *WAR directory*. If you import a WAR application, the components such as Web

applications are stored beneath the WAR directory. Note that an exploded archive-format WAR application cannot be executed with a remote environment.

■ Notes on using the exploded archive format

The notes on using the exploded archive format are as follows:

- When you import an exploded archive-format WAR application, you cannot specify a path containing the UNC name as the WAR directory path. If a path containing the UNC name is specified, a command execution error occurs.
- You cannot specify a path (including relative paths) indicating a directory immediately beneath the Windows drive (such as `C:\`) or the UNIX root directory (`/`) in the WAR directory. If a path indicating a directory immediately beneath the drive or the root directory is specified as a WAR directory in the `-a` option of the `cjimportwar` command, a command execution error occurs.
- If a WAR application with the same directory as the directory specified in the `-a` option of the `cjimportwar` command already exists in a J2EE server, a command execution error occurs during the import operation.
- If the J2EE server already contains a J2EE application with the following directories as the application directory or WAR directory, a command execution error occurs during the import operation:
 - Directory above the directory specified in the `-a` option of the `cjimportwar` command
 - Directory beneath the directory specified in the `-a` option of the `cjimportwar` command

13.9.3 Replacing WAR applications

For a WAR application, as in the case of an existing J2EE application (EAR format), you can use the redeploy functionality to replace a WAR file existing in the development environment with a WAR application that is already running. For details on the replacement of J2EE applications, see *13.7 Redeploying J2EE applications*.

! Important note

When you redeploy and replace a WAR application, the `cosminexus.xml` file cannot be read again. To change the WAR application attributes, use the server management commands (`cjgetappprop` and `cjsetappprop` command).

13.9.4 Names of WAR applications

Specify the name of the application with the `-name` option of the `cjimportwar` command. The strings that can be specified in the `-name` option are the same as the strings that can be specified in the `<display-name>` tag beneath the `<application>` tag in `application.xml`. For details on the available characters, see *cjimportwar (Importing WAR applications)* in the *uCosminexus Application Server Command Reference Guide*. If a character that cannot be specified is used, the KDJE37206-E message is displayed.

If the `-name` option is not specified in the `cjimportwar` command, the J2EE server replaces the non-usable characters with underscores (`_`) and specifies the application name based on the WAR file name (in the archive format) or WAR directory name (in the exploded archive format). Note that if the application name specified by the J2EE server is duplicated in the J2EE server, a serial number (1 to 2147483647) is added at the end of that application name so that the application name becomes unique in the J2EE server.

The following tables describe example settings for the application names.

Table 13–24: Example settings for the application name (In the archive-format WAR applications)

-name option of the <code>cjimportwar</code> command	WAR file name	Application name
SampleTP	SampleWeb.war	SampleTP
(Not specified)	SampleWeb.war	SampleWeb

Table 13-25: Example settings for the application name (in the exploded archive-format WAR applications)

-name option of the <code>cjimportwar</code> command	Directory name of the WAR application	Application name
SampleTP	SampleWeb	SampleTP
(Not specified)	SampleWeb	SampleWeb

13.9.5 Determining the context root

Specify the context root in the `-contextroot` option of the `cjimportwar` command. Specify the characters that can be used with URI (RFC3986) in the `-contextroot` option. If you specify a character that cannot be used with URI (RFC3986) or a string beginning with `ejb/`, `web/`, `/ejb/`, and `/web/`, the KDJE37206-E message is output. For details on the `cjimportwar` command, see *cjimportwar (Importing WAR applications)* in the *uCosminexus Application Server Command Reference Guide*.

If the `-contextroot` option is not specified in the `cjimportwar` command, the J2EE server determines the context root based on the WAR file name (in the archive format) or WAR directory name (in the exploded archive format) as follows:

- If the string excluding the extension is `ejb` or `web`, the strings are converted into `ejb1` or `web1` respectively.
- For strings other than `ejb` or `web`, the J2EE server determines the context root by replacing the non-usable characters with underscores (`_`).

The following tables describe examples of determining the context root.

Table 13-26: Example of determining the context root (In the archive-format WAR applications)

-contextroot option of the <code>cjimportwar</code> command	WAR file name	Context root
example/Servlet	SampleWeb.war	example/Servlet
(Not specified)	SampleWeb.war	SampleWeb

Table 13-27: Example of determining the context root (in the exploded archive-format WAR applications)

-contextroot option of the <code>cjimportwar</code> command	Directory name of the WAR application	Context root
example/Servlet	SampleWeb	example/Servlet
(Not specified)	SampleWeb	SampleWeb

If you execute the `cjlistapp` command, the context root is output in the displayed string. For details on the `cjlistapp` command, see *cjlistapp (Displaying a list of applications)* in the *uCosminexus Application Server Command Reference Guide*.

13.9.6 Reading the `cosminexus.xml` file

Specify the `cosminexus.xml` file to be read in the `-c` option of the `cjimportwar` command. If the `-c` option is omitted, the `cosminexus.xml` file is not read. For details on the `cjimportwar` command, see *cjimportwar (Importing WAR applications)* in the *uCosminexus Application Server Command Reference Guide*.

For details on the operations of an application containing `cosminexus.xml`, see *11.3.6 Operations of the applications containing cosminexus.xml*.

14

Container Extension Library

This chapter describes the container extension library functionality.

With Application Server, you can use the common libraries from the Enterprise Beans and servlets as the container extension library.

14.1 Organization of this chapter

The following table describes the container extension library functionality and the reference locations.

Table 14-1: Container extension library functionality and reference locations

Functionality	Reference locations
Using the container extension library	<i>14.2</i>
Container extension library functionality	<i>14.3</i>
Server start and stop hook functionality	<i>14.4</i>
Invoking the CORBA objects through Smart Agent	<i>14.5</i>
Constraints on using the container extension library and server start and stop hook functionality	<i>14.6</i>

14.2 Using the container extension library

This section provides an overview of the container extension library and server start and stop hook functionality.

With Application Server, you can use a user-created library if a processing is to be commonly used between an EJB-JAR and WAR or between different EARs. By using the user-created library, you can extend the functionality of the servlets, JSPs, and Enterprise Beans.

The library that can be used in common by the servlets, JSPs, and Enterprise Beans is called the *container extension library*. By using this library, you can invoke the user-created library commonly from the Enterprise Beans, servlets, and JSPs.

Furthermore, by using the **server start and stop hook functionality**, you can invoke the container extension library when you start and stop the server. You can also initialize the JNI functionality that is used in the container extension library.

To use the container extension library, compile the libraries into a single JAR file and define the settings for using the container extension library in `usrconf.cfg`. When the container extension library uses the JNI, you must also specify the settings for using the server start/ stop hook functionality.

For details on the settings for the container extension library, see *14.3.3 Settings for using the container extension library functionality*.

14.3 Container extension library functionality

This section describes the container extension library functionality.

The following table describes the organization of this section.

Table 14-2: Organization of this section (Container extension library functionality)

Category	Title	Reference location
Explanation	Determining the usage of the container extension library	14.3.1
	Procedure of creating and using the container extension library	14.3.2
Settings	Settings for using the container extension library functionality	14.3.3

#1: The functionality-specific explanation is not available for "Implementation" and "Operations".

#2: For details about the constraints on using the container extension library, see *14.6 Constraints on using the container extension library and server start and stop hook functionality*.

14.3.1 Determining the usage of the container extension library

This section describes how to determine the usage of the container extension library according to the processing type and processing contents.

(1) Determining the usage based on the processing type

Classify the processing into the following three types and then determine whether to use the container extension library. If you do not want to use the container extension library, include the common libraries in the EJB-JAR files, WAR files, or library JARs.

- **Business processing**

Include the common libraries in the EJB-JAR files or WAR files because the processing for each business is different. You need not use the container extension library.

- **Processing shared by the EJB-JAR files and WAR files**

If the Enterprise Beans, servlets, JSPs, and business processing included in multiple EJB-JAR files and WAR files can commonly utilize the processing, use a library JAR. If you cannot use the library JAR, create a common processing class and use the container extension library.

- **Processing shared by the EAR files**

If the EJB-JARs and WARs included in the multiple EAR files can commonly utilize the processing, use a library JAR. If you cannot use the library JAR, create a common processing class and use the container extension library.

(2) Determining the usage based on the processing contents

Classify the operations of the Enterprise Beans, servlets, and JSPs as follows, and determine whether to use the container extension library for each. The following table describes the guidelines for using the container extension library.

Table 14-3: Guidelines for using the container extension library for each operation

Operations	Container extension library	EJB-JAR file or WAR file	Library JAR
Operations to access the files and directories	Y	N	N
Operations using the JNI [#]	Y	N	N

Operations	Container extension library	EJB-JAR file or WAR file	Library JAR
Operations for invoking the CORBA objects through Smart Agent	Y	N	N
Operations of the J2EE container functionality	N	Y	N
Class references within the EJB-JAR files and WAR files	N	--	N

Legend:

Y: Included

N: Not included

--: Not applicable

#: If you attempt to load the same native library in J2EE applications or Web applications, `UnsatisfiedLinkError` is thrown due to the JNI specifications. You can avoid the error by registering the native library that is commonly used by the applications, as the container extension library.

! Important note

The following access permission is assigned to the container extension library. You cannot change the access permission.

```
java.security.AllPermission
```

However, the `setSecurityManager` access permission of `java.lang.RuntimePermission` is not assigned.

14.3.2 Procedure of creating and using the container extension library

The procedure of creating and using the container extension library is as follows:

1. Implement and compile the user-created classes.

If the server start and stop hook functionality is used, specify `Cosminexus-installation-directory\CC\lib\ejbserver.jar` in the class path and execute compile. Note that when the server start and stop hook functionality is used, you implement and compile the classes with the method in which the IDE is not used.

With the server start and stop hook functionality, you can also implement the invoke processing of the CORBA objects through Smart Agent, provided by Cosminexus TPBroker.

2. Archive the created classes in a JAR file.

Archive the user-created classes in the JAR file for the container extension library. Do not include the classes in the EJB-JAR files and WAR files.

3. Specify the archived JAR file in the system class path of the J2EE server.

To use the container extension library, apart from this procedure, you must specify the settings in the J2EE server (such as specifying the definition files).

For details on the settings in the J2EE server, see *14.3.3 Settings for using the container extension library functionality*.

14.3.3 Settings for using the container extension library functionality

This subsection describes the settings for using the container extension library. Note that if the container extension library uses the JNI, use the server start and stop hook functionality.

To use the container extension library, you must specify the following settings:

1. Create the JAR file for the container extension library.

For details on using the container extension library, see *14.2 Using the container extension library*.

2. Specify the following parameter in the `<configuration>` tag of the logical J2EE server (`j2ee-server`) in the Easy Setup definition file:

- `add.class.path`

In the value set up for `add.class.path`, specify the path of the JAR file created in 1 as the system class path of the J2EE server.

Set `add.class.path` in the extension parameter of the J2EE server in the Easy Setup definition file.

3. To use the JNI functionality from the container extension library, specify the following parameter in the `<configuration>` tag of the logical J2EE server (`j2ee-server`) in the Easy Setup definition file:

- `add.library.path`

In the value set up for `add.library.path`, specify the search path of the JNI library.

- `ejbserver.application.InitTermProcessClasses`

In the value set up for `ejbserver.application.InitTermProcessClasses`, specify the class name of the server start and stop hook functionality.

Set `add.library.path` and `ejbserver.application.InitTermProcessClasses` in the extension parameters of the J2EE server in the Easy Setup definition file.

4. Specify the JAR file for the container extension library in the key of the server management command file. The file and key to be specified differ according to the OS.

- **In Windows**

`USRCONF_JVM_CLASSPATH` key of `usrconf.bat`

- **In UNIX**

`USRCONF_JVM_CLPATH` key of `usrconf`

5. To use the JNI functionality from the container extension library, specify the search path of the JNI library in the key of the server management command file.

To specify multiple search paths, demarcate with semicolons (;).

The file to be specified differs according to the OS (the key is common).

- **In Windows**

`USRCONF_JVM_LIBPATH` key of `usrconf.bat`

- **In UNIX**

`USRCONF_JVM_LIBPATH` key of `usrconf`

For details on the files, see 4.6 *Easy Setup definition file* and 5.3 *usrconf.bat (Option definition file for server management commands)* in the *uCosminexus Application Server Definition Reference Guide*.

The following is an example of the settings for the Easy Setup definition file and the user-defined files. In this example, the JAR file of the container extension library is `extended_container.jar` and the container extension library uses the JNI to invoke `extended_container.dll` (in UNIX, `extended_container`).

- In Windows
 - Example of the settings for the Easy Setup definition file

```
<configuration>
  <logical-server-type>j2ee-server</logical-server-type>
  <param>
    <param-name>add.class.path</param-name>
    <param-value>c:\jar\extended_container.jar</param-value>
    <param-name>add.library.path</param-name>
    <param-value>c:\lib</param-value>
  </param>
  :
</configuration>
```

- Example of the settings for `usrconf.bat` used for the server management commands

```
rem system classpath
set USRCONF_JVM_CLASSPATH=c:\jar\extended_container.jar

rem library path
set USRCONF_JVM_LIBPATH=c:\lib
```

- In UNIX

- Example of the settings for the Easy Setup definition file

```
<configuration>
  <logical-server-type>j2ee-server</logical-server-type>
  <param>
    <param-name>add.class.path</param-name>
    <param-value>/work/classes/extended_container.jar</param-value>
    <param-name>add.library.path</param-name>
    <param-value>/work/lib</param-value>
  </param>
  :
</configuration>
```

- Example of the settings for `usrconf` used for the server management commands

```
#!/bin/csh -f
# system classpath
set USRCONF_JVM_CLPATH=/work/classes/extended_container.jar

# library path
set USRCONF_JVM_LIBPATH=/work/lib
```

14.4 Server start and stop hook functionality

This section describes the server start and stop hook functionality.

The following table describes the organization of this section.

Table 14-4: Organization of this section (Server start and stop hook functionality)

Category	Title	Reference location
Explanation	Invocation order of the server start and stop hook processing	14.4.1
Implementation	Implementing the server start and stop hook functionality	14.4.2
Settings	Specifying the class path for using the server start and stop hook functionality	14.4.3

#1: The functionality-specific explanation is not available for "Operations".

#2: For details on the constraints for using the server start and stop hook functionality, see 14.6 Constraints on using the container extension library and server start and stop hook functionality.

14.4.1 Invocation order of the server start and stop hook processing

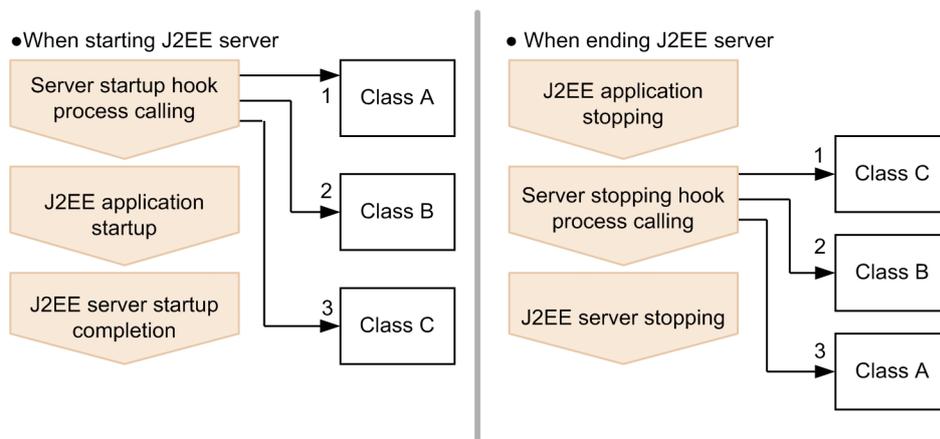
This subsection describes the invocation order of the server start and stop hook processing when multiple server start and stop hook functionality are registered.

(1) When the server is started and stopped normally

When you start the J2EE server, the `serverInitializing` method that performs the server start hook processing is invoked in the order in which the classes are registered. Also, when you stop the J2EE server, the `serverTerminating` method that performs the server stop hook processing is invoked in the reverse of the registered order.

The following figure shows the invocation order of the server start and stop hook functionality when the J2EE server is started normally.

Figure 14-1: Invocation order of the server start and stop hook processing (when the server is started normally)

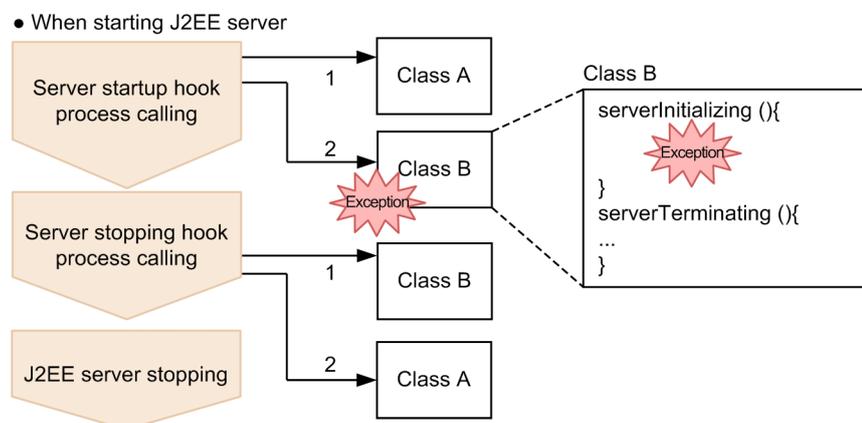


The three classes A, B, and C implement the server start/ stop hook processing and the classes are assumed to be registered in the order of class A, class B, and class C. With the server start hook processing, if the processing is invoked normally, the classes are invoked in the order of class A, class B, and class C. With the server stop hook processing, the classes are invoked in the order of class C, class B, and class A.

(2) When an exception occurs during the server start hook processing

The following figure shows the invocation order of the server start and stop hook functionality when an exception occurs during the server start hook processing.

Figure 14-2: Invocation order of the server start and stop hook processing (when an exception occurs during the start hook processing)

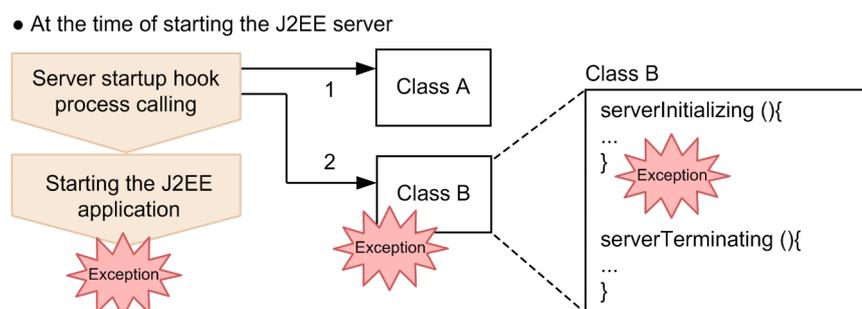


The three classes A, B, and C implement the server start/ stop hook processing and the classes are assumed to be registered in the order of class A, class B, and class C. If an exception occurs during the server start hook processing (`serverInitializing` method) of class B, the server start hook processing of class C is not invoked. Also, the server stop hook processing is invoked in the order of class B and class A, without invoking the server stop hook processing (`serverTerminating` method) of class C.

(3) When an exception occurs after the server start hook processing

The following figure shows the invocation order of the server start and stop hook functionality when an exception occurs after the execution of the server start hook processing.

Figure 14-3: Invocation order of the server start and stop hook processing (when an exception occurs after the start hook processing)



The three classes A, B, and C implement the server start/ stop hook processing and the classes are assumed to be registered in the order of class A, class B, and class C. If an exception occurs after the execution of the server start hook processing (`serverInitializing` method) of class B, the server start hook processing of class C is not invoked. Also, the server stop hook processing is not invoked.

14.4.2 Implementing the server start and stop hook functionality

You can use the server start and stop hook functionality by implementing the `com.hitachi.software.ejb.application.InitTermProcess` interface. Implement the server start hook processing in the `serverInitializing` method and the stop hook processing in the `serverTerminating` method.

The following is an example of implementing the `InitTermProcess` interface:

```

package sample;

import com.hitachi.software.ejb.application.InitTermProcess;
import com.hitachi.software.ejb.application.InitTermException;

public class AppInitTerm implements InitTermProcess {
    public void serverInitializing() throws InitTermException {
        try {
            // Server start hook processing
        } catch (Exception e) {
            throw new InitTermException("Detailed message");
        }
    }
    public void serverTerminating() throws InitTermException {
        try {
            // Server stop hook processing
        } catch (Exception e) {
            throw new InitTermException("Detailed message");
        }
    }
}

```

The J2EE server uses the default constructor during the startup to generate an instance of the server start and stop hook functionality. Therefore, specify `public` in the access specifier of the class and default constructor.

The server stop hook processing is invoked after the stop processing of the application. With the J2EE server, the stop processing is not performed for threads, so the processing threads remain behind, but after the server stop hook processing, a new application is not executed.

Specify the class name for the server start/ stop hook in the property key (`ejbserver.application.InitTermProcessClasses`) for the server start and stop hook functionality in the user-defined file (`usrconf.properties`). You can also specify multiple class names for the server start/ stop hook. For details on the user-defined file (`usrconf.properties`), see the *uCosminexus Application Server Definition Reference Guide*.

14.4.3 Specifying the class path for using the server start and stop hook functionality

When you use the server start and stop hook functionality, you must add the JAR file path in the system class path of the J2EE server. Specify the addition of the system class path in the user-defined file (`usrconf.cfg`).

For details on how to add the system class path, see *14.3.3 Settings for using the container extension library functionality*. For details on the user-defined file (`usrconf.cfg`), see the *uCosminexus Application Server Definition Reference Guide*.

14.5 Invoking the CORBA objects through Smart Agent

By using the container extension library, the J2EE applications or Web applications can use Cosminexus TPBroker to invoke the CORBA objects through Smart Agent. When Smart Agent is used to acquire the object references of the CORBA objects and to invoke the methods, Smart Agent must be running and the object references of the applicable CORBA objects must be registered in Smart Agent. Also, the container extension library includes the processing for invoking the CORBA objects from the J2EE applications or Web applications.

This section describes the invocation of the CORBA objects through Smart Agent.

The following table describes the organization of this section.

Table 14-5: Organization of this section (Invoking the CORBA objects through Smart Agent)

Category	Title	Reference location
Implementation	Notes on the implementation of the CORBA object invocation processing	14.5.1
	Notes on the packaging of the CORBA object invocation processing	14.5.2

Note: The functionality-specific explanation is not available for "Explanation", "Settings", and "Operations".

The notes on the implementation of the CORBA object invocation processing and the notes on the packaging of the CORBA object invocation processing are as follows:

14.5.1 Notes on the implementation of the CORBA object invocation processing

Note the following points when you implement the CORBA object invocation processing:

- To use a property for customizing `org.omg.ORB`, specify the property in the `props` parameter of the `org.ORB.init(String[] args, Properties props)` method.
Note that only the `vbroker.agent.port`, `vbroker.agent.enableLocator`, and `vbroker.agent.addr` keys can be specified in the J2EE server system properties.
- The server functionality of Cosminexus TPBroker cannot be used. Also, the CORBA objects cannot be activated on the J2EE server.
- The Cosminexus TPBroker interceptor cannot be used.
- The DII functionality of Cosminexus TPBroker cannot be used.

14.5.2 Notes on the packaging of the CORBA object invocation processing

Note the following points when you package the CORBA object invocation processing:

- Do not include the classes that use the interfaces and classes existing in the package under `org.omg.CORBA` in the EJB-JAR files or WAR files. Include these classes in the JAR file for the container extension library.
- Do not include the interfaces and classes generated from the IDL definition and the classes that use these interfaces and classes in the EJB-JAR files or WAR files. Include these classes in the JAR file for the container extension library.

14.6 Constraints on using the container extension library and server start and stop hook functionality

This section describes the constraints on using the container extension library and server start and stop hook functionality.

- **Application portability**

The container extension library functionality is not included in the J2EE specifications. Therefore, when the container extension library is used, the portability of the application is reduced.

- **Invoking the container extension library**

The container extension library is presumed to have been invoked from the servlets, JSPs, and Enterprise Beans. Note that the following usage forms cannot be applied:

- Referencing the EJB-JAR files and WAR files from the container extension library and server start and stop hook functionality (including the classes of the EJB-JAR files and WAR files inheriting the container extension library class).
- Using the J2EE container functionality from the container extension library and server start and stop hook functionality (invoking the APIs such as the Enterprise Beans, JNDI, and JDBC).
- Referencing the server start and stop hook functionality directly from the classes of the EJB-JAR files and WAR files (including the classes of the container extension library inheriting the classes of the EJB-JAR files and WAR files).

- **Accessing the files and directories**

Do not operate the following files and directories from the container extension library and server start and stop hook functionality:

- Files and directories beneath the Cosminexus installation directory
- Files and directories beneath the J2EE server work directory

- **Using the JNI functionality**

When the JNI functionality is used from the container extension library and server start and stop hook functionality, the processing with the native method cannot be managed with the J2EE server. For example, if a memory access violation occurs in the native method, the J2EE server terminates abnormally in each JavaVM process.

- **Using Cosminexus TPBroker**

- The server functionality of Cosminexus TPBroker cannot be used with the container extension library and server start and stop hook functionality. This is because the CORBA objects of Cosminexus TPBroker cannot be activated on the J2EE server.
- The DII functionality of Cosminexus TPBroker cannot be used.

- **Using the installation type option package**

Do not use the container extension library and server start and stop hook functionality as the installation type option package. The installation type option package is the file placed in the following directories:

- The JAR file placed in *Cosminexus-installation-directory*\jdk\jre\lib\ext
- The native code binary placed in *Cosminexus-installation-directory*\jdk\jre\bin

- **Registering the shutdown hook**

Do not register the shutdown hook with the container extension library and server start and stop hook functionality.

- **Setting up the handler function (in Windows)**

When you use the container extension library for setting up the handler function of a process in Windows, do not execute operations such as returning `TRUE`, performing the DLL end processing, and terminating a process by invoking functions such as `ExitProcess` with the handler function that processes the `CTRL+BREAK` signal.

The J2EE server and the other programs that use Component Container might not operate.

- **C++ library (in Linux)**

If the container extension library is implemented using C++ in Linux, that library must be built with Red Hat Enterprise Linux 4 or later. A C++ library built with Red Hat Enterprise Linux 3 cannot be executed.

15

Notes on Implementation of Applications

This chapter describes the notes for implementing applications.

15.1 Notes on using the thread-local variables

If you do not delete the class instances of the J2EE application stored in the thread-local variable until the J2EE application stops, a memory leak occurs because the class loader references remain behind even after you stop the J2EE application.

Use the `java.lang.ThreadLocal.remove()` method to delete the class instances of the J2EE application stored in the thread-local variable.

If you cannot delete the instances in which the items such as the framework are stored in a J2EE application, do not start and stop the J2EE application repeatedly. To start and stop a J2EE application repeatedly, restart the J2EE server.

15.2 Notes on Developer's Kit for Java

This section describes common notes on Developer's Kit for Java and notes related to the differences in the specifications of the JDK versions.

15.2.1 Notes common to Application Server versions

This subsection describes notes related to Developer's Kit for Java common to Application Server versions.

(1) Notes common to OSs

This subsection describes the notes common to OSs.

- Compatibility with the earlier Java SE versions
Some Java SE 6 functionality is not compatible with the earlier versions of Java SE (J2SE). For details, see the relevant page (<http://www.oracle.com/technetwork/java/javase/compatibility-137541.html>).
- Other changes in Java SE 6
For details on the other extensions and changes in Java SE 6, see the relevant page (<http://www.oracle.com/technetwork/java/javase/releasenotes-136954.html>).
- JVMPI and JVMDI
JVMPI and JVMDI are removed. Instead of these interfaces, use JVM TI.
- Functionality for obtaining the GC memory information
The `JP.co.Hitachi.soft.jvm.MemoryInfo` class that obtains the GC memory information is available with Developer's Kit for Java. This class is not available with other products.
- Loading of classes in `URLClassLoader`
When a class is loaded from a `jar` file via `URLClassLoader`, the objects created in JavaVM during the loading process might not be deleted until JavaVM is terminated. When appropriate, restart the Java application.
- Using Java2D
To avoid the `DirectDraw` and `Direct3D` problems in Java2D, specify `true` in the `sun.java2d.noddraw` property that disables `DirectDraw` and `Direct3D`.
- Maximum method length
According to the JavaVM specifications, the byte code of one method must be less than 64 kilobytes. If the byte code exceeds 64 kilobytes, an error occurs when a class file is generated, or the `java.lang.LinkageError` exception occurs when a class is loaded. Also, if the byte code is less than 64 kilobytes, but is extremely complicated and has a large number of lines, the following negative effects might occur:
 - The execution of garbage collection takes a very long time.
 - The JIT compilation takes a very long time.
 - The JIT compilation consumes a lot of memory.
 Additionally, if the functionality to output the local variable information is enabled, the following negative effects might also occur:
 - The output of the extension thread dump takes time.
 - The obtaining of the thread stack trace takes time.
 - The generation of an exception object when an exception occurs takes time.
 Therefore, we recommend that you specify less than about 500 lines, excluding comments and blank lines, for each method in the Java source.
- Serialized version UID (`serialVersionUID`)
With JDK 1.4 or later, if the serializable nested class contains an explicit reference to the class object (for example, `Class c = Object.class;`), the value of the default serialized version UID differs from the versions earlier than JDK 1.4 in both the nested class and the class including that class. This is because of the

changes in the `javac` command implemented in JDK 1.4. If all of these conditions are satisfied, add the serialized version UID (`serialVersionUID`) in the serializable class.

- Closing the file descriptor

The child processes started with `java.lang.Runtime.exec()` and `java.lang.ProcessBuilder.start()` communicate with each other through the streams fetched with the `Process.getInputStream`, `getErrorStream`, or `getOutputStream` methods respectively.

With the parent process, note that three file descriptors are consumed even if these methods are not used.

Close the file descriptors in the following cases:

- When the `Process` object is finalized
- When `Process.destroy()` is invoked
- When the `close()` method is explicitly invoked for these streams

- System properties `java.ext.dirs` and `java.library.path`

In the system property `java.ext.dirs`, specify a directory that allocates the `jar` files containing classes that are loaded on higher priority than the normal application classes. The default values are as follows:

In Windows

```
JDK-installation-directory\jdk\jre\lib\ext
```

In UNIX

```
/opt/Cosminexus/jdk/jre/lib/ext
```

Also, in the system property `java.library.path`, specify the search path of the native library for the user. The default values are as follows:

In Windows

```
"JDK-installation-directory\bin, system-directory, Windows-directory, %PATH%, current-directory"
```

In AIX

Value set in the environment variable `LD_LIBRARY_PATH`, directory of the native library beneath `JDK`, `/usr/lib:/lib`

In HP-UX

Directory of the native library beneath `JDK`, value set in the environment variable `LD_LIBRARY_PATH`, `/usr/lib:/usr/lib/hpux64:/usr/ccs/lib/hpux64`

In Linux

Directory of the native library beneath `JDK`, value set in the environment variable `LD_LIBRARY_PATH`, `/usr/lib64:/lib64:/lib:/usr/lib`

- Storage time of the positive DNS cache

JDK has the functionality for caching query results of the DNS host name resolution. The cache functionality includes a positive cache that stores the results of a successful resolution and a negative cache that stores the results of a failed resolution.

Among these, only when the security manager is disabled, the default storage time of the positive cache for the DNS query results changes from "permanent cache" to "implementation dependent". For the same behavior as JDK 5.0, set the value of the security property `networkaddress.cache.ttl` to `-1`. For details, see the relevant page (<http://docs.oracle.com/javase/6/docs/api/java/net/InetAddress.html>).

- Messages output by the instrumentation functionality

If memory is insufficient while the instrumentation functionality is being processed, the following message is displayed and the processing of the instrumentation functionality might fail. For details, see the relevant page (<http://docs.oracle.com/javase/jp/1.5.0/api/java/lang/instrument/package-summary.html>).

Message contents

```
*** java.lang.instrument ASSERTION FAILED ***
```

- Timeout of `java.net.Socket.connect()`

The socket connection for `java.net.Socket.connect()` times out as per the timeout value for the TCP communication specified in the OS. When the TCP communication timeout value is reached, the connection times out even if the timeout value specified in `java.net.Socket.connect()` has not been reached, and even if the timeout value is not specified.

When the connection times out at the TCP communication timeout value, the `java.net.ConnectException` exception containing the following detailed message is thrown:

Message contents

```
Connection timed out: connect [errno=10060, syscall=select]
```

For details on the TCP communication timeout, see the OS documentation.

- **Class loading time**
With Java, a class file is read into the memory (class loading) when that class is required for the first time during the execution of a program. Therefore, if the class loading frequency is high during the initial execution of a particular processing, the processing time might be longer than the second and subsequent times. In this case, improve processing time by loading the classes required for execution in advance.
- **Default encoding**
The default Application Server encoding is as follows:
In Windows
 Windows-31J (MS932)
In UNIX
 - When `LANG` is `C` or `POSIX`: US-ASCII (ASCII)
 - When `LANG` is `jp_JP`, `jp_JP.eucJP`, `ja_JP.ujis`, `japanese`, or `japanese.euc`: `x-euc-jp-linux` (`EUC_JP_LINUX`)
 - When `LANG` is `ja_JP.utf8`: UTF-8 (optional name UTF8)
- **Garbling of characters between different OSs**
The mapping to the character code Unicode varies depending on the OS. Therefore, garbling might occur in the following cases. The typical characters that are garbled include `-`, `~`, `//`, `-`, `↑`, `↓`, and `✦`.
 - When Unicode encoding such as UTF-8 is used to pass the above character data between different OSs
 - When the class files created from a source program with a literal string containing the above characters are executed as they are without recompilation, on different OSs
- **Files used with the Java APIs**
The configuration files, such as the policy file or login configuration file, used with Java APIs, must be encoded using the UTF-8 encoding method.
- **Encoding of the URL strings handled by `java.net.URL`**
The encoding for the URL strings handled by `java.net.URL` is Modified UTF-8 and not the standard UTF-8. If a string that is not Modified UTF-8 is passed, `java.lang.IllegalArgumentException` occurs.
- **UTF-8 encoding for the string operations with the JNI function**
In JNI, the encoding to be used for the functionality performing the following string operations is Modified UTF-8 and not the standard UTF-8:
 - `NewStringUTF`
 - `GetStringUTFLength`
 - `GetStringUTFChars`
 - `ReleaseStringUTFChars`
 - `GetStringUTFRegion`
- **Maximum jumpable length for determining the `if` statement**
According to the JavaVM specifications, the jumpable length for determining the `if` statement is up to 32 kilobytes. If the jump destination exceeds 32 kilobytes, `java.lang.LinkageError` occurs.
- **Maximum file size**
The `java.util.zip` and `java.util.jar` packages do not support JAR files or ZIP files exceeding 4 gigabytes.
- **`map`, `transferFrom`, and `transferTo` methods of the `java.nio.channels.FileChannel` class**
The `map` method does not support files exceeding 2 gigabytes.
Also, the `count` specification in the `transferFrom` and `transferTo` method does not support a value of 2 gigabytes or more.

- Stack size of threads

When you use a function such as the JNI function `AttachCurrentThread()` to connect the native thread to JavaVM, use a thread with a stack size greater than the value specified in the `-Xss` option for connection.

(2) In Windows

This subsection describes the notes for Windows.

- No support for the third-level and fourth-level characters of JIS X0213:2004 in the AWT components
The AWT components do not support the third-level and fourth-level characters of JIS X0213:2004. Such characters are garbled in the information output by the AWT components. To handle these characters in Windows Server 2012, Windows Server 2008, Windows 8, Windows 7, and Windows Vista, use the Swing components.
- Garbage collection elapsed time for the JavaVM log files
The garbage collection elapsed time for the JavaVM log files is output up to the seventh place after the decimal point in second units (100 nanoseconds).
In Windows, with version 08-70 or earlier, a valid number is output up to the seventh place after the decimal point (100 nanoseconds). With 08-70 or later, a valid number is output up to the sixth place after the decimal point (1 microsecond), and 0 is always output in the seventh place after the decimal point (100 nanoseconds).
- Libraries that cannot be used with the JNI programs
The following libraries cannot be used with the JNI programs because the information managed in JavaVM might be updated incorrectly:
 - `setjmp()`
 - `longjmp()`
- `java.io.tmpdir` property
In the `java.io.tmpdir` property, specify an existing directory with write permissions. The initial value of the `java.io.tmpdir` property is the directory obtained with the `GetTempPath()` function of the Windows API. Also, with the dynamic class loading functionality of Java RMI and `java.io.File.createTempFile()` of the Java API, a temporary file is created in the directory specified in the `java.io.tmpdir` property. To operate the functionality normally, do not delete the creation destination of the temporary file while the JavaVM process is running.
- Starting independent processes that require administrator privileges (In Windows Server 2012, Windows Server 2008, Windows 8, Windows 7, and Windows Vista)
To start independent processes with administrator privileges using `java.lang.Runtime.exec()` or `java.lang.ProcessBuilder.start()` from a java application started by a user who does not have administrator privileges:
 1. Add a manifest to the command to be started as an administrator privilege. For details on how to add a manifest, see the OS documentation.
 2. Start the process through `Windows cmd.exe`.
For example, to start `sample.exe`, the operations are as follows:


```
- Runtime.getRuntime().exec("cmd.exe", "/c", "sample.exe");
- new ProcessBuilder("cmd.exe", "/c", "sample.exe").start();
```

 When you start this java application, the User Account Control dialog box appears. Click **Continue** to continue the application.
- Data sharing by `java.util.prefs.Preferences.systemNodeForPackage` (In Windows Server 2012, Windows Server 2008, Windows 8, Windows 7, and Windows Vista)
If a user who does not have administrator privileges uses `java.util.prefs.Preferences.systemNodeForPackage`, the functionality for sharing data between different users cannot be used.
For example, when the same key is specified in the argument, different data is returned among different users.
- Garbling of characters during encoding
To handle the following characters with a java program, specify MS932, windows-31j, or cswindows31j in the encoding. If any other encoding is specified, the characters might be garbled.

- NEC extended character ①②...⑳, I II ... X, (株), 明治大正昭和平成, 三ツキ etc
- NEC appointment IBM extended character i ii ... x etc
- IBM extended character I II ... X, i ii ... x, No., TEL etc

(Example)

```
import java.io.*;
class encode_eucjis {
    public static void main( String arg[] ) {
        try {
            String string_data = " I II III";
            byte[] data = string_data.getBytes();
            InputStreamReader isr =
                new InputStreamReader(
                    new ByteArrayInputStream( data ), "eucjis");
            char[] read_data = new char[4];
            isr.read( read_data, 0, 4 );
            System.out.println(new String(read_data));
        }
        catch ( Exception e ) {
            e.printStackTrace();
        }
    }
}
```

In this program, the IBM extension characters are encoded with eucjis, so after execution, the characters are garbled as follows:

```
java encode_eucjis
???
```

To avoid the garbling, encode by specifying MS932, windows-31j, or cswindows31j.

- UTF-8 encoding supported by Java

The UTF-8 encoding supported by Java is UTF-8 without BOM. The data stored by specifying "UTF-8" as "character code" in a Windows text editor (notepad) becomes UTF-8 with BOM, so the data cannot be handled with Java.

(3) Notes common to UNIX

This subsection describes notes common to UNIX.

- `fork` system call

If you generate or execute a child process of a copy of the current process only by issuing the `fork()` system call in the native method or native code invoked by JNI and JVMTI, the operations of that parent-child process are not guaranteed. After generating the child process by issuing the `fork()` system call, make sure that you load the new program with the `exec()` system call and then start the program. Also, to generate a child process in the Java environment, we recommend use of the `java.lang.Runtime.exec()` method of the Java class library.

- Signals

If a signal handler is registered for the following signals with the native method or native code invoked by JNI and JVMTI, the operations might not function properly:

SIGHUP, SIGINT, SIGQUIT, SIGILL, SIGFPE, SIGBUS, SIGSEGV, SIGPIPE, SIGTERM, SIGUSR2, SIGCHLD, SIGXCPU, SIGXFSZ, signal of number (`_SIGRTMAX-2`)

- Receiving signals while the system library functions and system call are being invoked

The following signals might be received during the invocation of the system library functions and system call:

SIGHUP, SIGINT, SIGQUIT, SIGILL, SIGFPE, SIGBUS, SIGSEGV, SIGPIPE, SIGTERM, SIGUSR2, SIGCHLD, SIGXCPU, SIGXFSZ, signal of number (`_SIGRTMAX-2`)

When you invoke a system library function and system call, the processing of the applicable function is interrupted due to the receipt of these signals and an error is returned (EINTR is set in the `errno` value). In this case, take the appropriate action (for example, re-execution).

- Libraries that cannot be used with the JNI programs

The following libraries cannot be used with the JNI programs because the information managed in JavaVM might be updated and become invalid:

- `setjmp()`

- `longjmp()`
- `_setjmp()`
- `_longjmp()`
- `sigsetjmp()`
- `siglongjmp()`
- **Passing of character data to Windows**
 To pass the character data to Windows, specify MS932, windows-31j, or cswindows31j for the encoding.
 If you specify Shift_JIS or SJIS, the following characters are converted into invalid character codes:
 - NEC extended character ①②...㉔, I II ... X, (株), 明治大正昭和平成, ㄣ ㄩ ㄐ ㄑ etc
 - NEC appointment IBM extended character i ii ... x etc
 - IBM extended character I II ... X, i ii ... x, No., TEL etc
- **Maximum number of threads that can be generated**
 The maximum number of threads that can be generated in the entire system is `/proc/sys/kernel/threads-max`. Also, the maximum number of threads that can be generated per user is the value of `nproc` (same as `ulimit -u`) in `/etc/security/limits.conf`. Adjust these values in accordance with the system.
- **Number of notification pending monitors**
 The information on the notification pending monitors obtained by using the JVMTI interface might be invalid.

(4) In AIX

This subsection describes notes for AIX.

- **Exception message for `Runtime.exec`**
 If an exception occurs with `Runtime.exec`, the error message might be garbled.
- **`java.security.SecureRandom` class**
 The default random number generator of the OS used by the `SecureRandom` class is `/dev/urandom` file. If you specify `file:/dev/random` in the `java.security.egd` property, you can use the `/dev/random` file. However, if the `/dev/random` file is used, the speed at which the random numbers are generated for the OS is limited. Therefore, if you implement the following processes in which the `SecureRandom` class obtains random numbers from the `/dev/random` file at short intervals, note that the processing does not finish until the OS finishes random number generation:
 - When the `generateSeed()` method of the `SecureRandom` class is invoked
 - When the `getSeed()` method of the `SecureRandom` class is invoked
 For details on the `java.security.egd` property, see the relevant page (<http://docs.oracle.com/javase/6/docs/technotes/guides/security/jaas/JAASRefGuide.html>).
- **Print support using `java.awt.print`**
 The AIX print sub-system supported by the `java.awt.print` package print API of Developer's Kit for Java is the System V print sub-system. To use the Print API, set up the System V print sub-system as follows:
 1. Execute the `smit printer` command.
 2. Choose **Print spooling, Change/ Display the current print sub-system**, and then **Current print sub-system**. Use the **Tab** key to change to **SystemV**.
- **Environment variable LANG**
 If a value other than the following values is specified in the environment variable `LANG`, the exception message output by the `java.nio.channels` package might be garbled:
 - C
 - POSIX
- **`java.io.tmpdir` property**
 Specify an existing directory with write permissions in the `java.io.tmpdir` property. The initial value of the `java.io.tmpdir` property is `/tmp`.

Also, with the dynamic class loading functionality of Java RMI and `java.io.File.createTempFile()` of the Java API, a temporary file is created in the directory specified in the `java.io.tmpdir` property. To operate the functionality normally, do not delete the creation destination of the temporary file while the JavaVM process is running.

(5) In HP-UX

This subsection describes notes for HP-UX.

- No support for the third-level and fourth-level characters of JIS X0213:2004 in the AWT components
The AWT components do not support the third level and fourth level characters of JIS X0213:2004. Such characters are garbled in the information output by the AWT components. To handle these characters, use the Swing components.
- Exception message for `Runtime.exec`
If an exception occurs with `Runtime.exec`, the error message might be garbled.
- `java.security.SecureRandom` class
With the platform on which the `/dev/random` file exists, you obtain the random numbers generated by the OS from the `/dev/random` file using some of the API of the `SecureRandom` class. The random numbers are obtained from the `/dev/random` file when all the following conditions are satisfied:
 - `file:/dev/random/` is specified in `securerandom.source` in the `java.security.egd` property or in the `java.security` file
 - The `generateSeed()` method or `getSeed()` method of the `SecureRandom` class is invoked
 The speed at which the random numbers are generated for the OS is limited. Therefore, if you execute the process of obtaining the random numbers at short intervals, note that the processing does not finish until the OS finishes random number generation.
- Library path loaded in a java program
The paths specified for the libraries loaded with the `java.lang.System.load` method and `java.lang.Runtime.load` method are ignored, and the library is loaded from the path specified in the environment variables `SHLIB_PATH` and `LD_LIBRARY_PATH`.
- `java.nio.MappedByteBuffer.isLoaded` method
If you use the `isLoaded()` method that queries whether the direct buffer exists in physical memory, you cannot query whether a specific memory area exists in physical memory. The return value of the `isLoaded()` method is always `false`.
- `java.lang.Thread.setPriority` method
The priority order of each thread depends on the OS-based control. Therefore, even if you specify the priority order of the threads with the `setPriority()` method, the order is not enabled.
- Environment variable `LANG`
If a value other than the following values is specified in the environment variable `LANG`, the exception message output by the `java.nio.channels` package might be garbled:
 - C
 - POSIX
- `java.io.tmpdir` property
Specify an existing directory with write permissions in the `java.io.tmpdir` property. The initial value of the `java.io.tmpdir` property is `/var/tmp`.
Also, with the dynamic class loading functionality of Java RMI and `java.io.File.createTempFile()` of the Java API, a temporary file is created in the directory specified in the `java.io.tmpdir` property. To operate the functionality normally, do not delete the creation destination of the temporary file while the JavaVM process is running.

(6) In Linux

This subsection describes notes for Linux.

- **Single-byte Katakana**
Among the characters established by JIS X 201, single-byte Katakana cannot be used with the GUI implementation using the JavaSE API.
- `java.security.SecureRandom` class
With the platform on which the `/dev/random` file exists, you obtain the random numbers generated by the OS from the `/dev/random` file using some of the API of the `SecureRandom` class. The random numbers are obtained from the `/dev/random` file when all the following conditions are satisfied:
 - `file:/dev/random/` is specified in `securerandom.source` in the `java.security.egd` property or in the `java.security` file
 - The `generateSeed()` method or `getSeed()` method of the `SecureRandom` class is invoked
 The speed at which the random numbers are generated for the OS is limited. Therefore, if you execute the process of obtaining the random numbers at short intervals, note that the processing does not finish until the OS finishes random number generation.
- `java.io.tmpdir` property
Specify an existing directory with write permissions in the `java.io.tmpdir` property. The initial value of the `java.io.tmpdir` property is `/tmp`.
Also, with the dynamic class loading functionality of Java RMI and `java.io.File.createTempFile()` of the Java API, a temporary file is created in the directory specified in the `java.io.tmpdir` property. To operate the functionality normally, do not delete the creation destination of the temporary file while the JVM process is running.
- Using the product with Red Hat Enterprise Linux 5.1
To use the product in an environment in which IPv6 of Red Hat Enterprise Linux 5.1 is enabled, specify `true` in the `java.net.preferIPv4Stack` property.

15.2.2 Notes related to the differences in specifications with JDK 1.4.2 provided in Application Server Version 6

This subsection describes the notes related to the differences in specification with JDK 1.4.2 provided in Application Server Version 6.

(1) Notes common to OSs

This subsection describes the notes common to OSs.

- **Differences in the display of values of the `java.math.BigDecimal` class**
With JDK 5.0 and JDK 6, the value output format varies because exponent notations are now used for string conversion by the `toString()` method:
(Example) To output the value of `1E-15` with the `toString()` method
 - In version JDK 1.4.2 or earlier: `0.000000000000001`
 - In JDK 5.0 and JDK 6: `1E-15`
 Note that to output the value in the JDK 1.4.2 or earlier format, use the `toPlainString()` method.
- **Unicode version**
With JDK 6, the character processing is based on the default Unicode version 4.0. Therefore, take precautions when you reference and parse the character code individually.
For details, see *Supporting Supplementary Characters in Your Application* on the relevant page (<http://www.oracle.com/technetwork/articles/javase/supplementary-142654.html>).
- **Handling annotations in the java source program**
With JDK 6, you check the character code for the characters in the source program annotations. For example, if a source program, containing the Shift-JIS code in an annotation, is compiled in an environment where the default encoding is EUC, a warning is displayed. During compilation, specify the appropriate encoding in the `-encoding` option.
- **Compiling the Java programs**

With the extension of the Java language specifications in J2SE 5.0, a caution or warning message is displayed and an error occurs if the syntax of the Java programs, which could be compiled by default with the `javac` command previously, violates the J2SE 5.0 Java language specifications. In the case of a caution or warning message, the programs that were being operated in the previous versions can be operated as they are. However, revising the Java code is preferable. Note that if you specify the `-source` option in the `javac` command, you can compile the programs by specifying the source code version 1.4 that is received by the `javac` command. With this, you can control the caution or warning message.

- Caution or warning messages output during the compilation of the Java programs

With the extension of the Java language specifications in J2SE 5.0, source coding methods based on the new language specifications are recommended; therefore, the following caution messages might be output when the compilation ends:

- Note: The `AccountEJB.java` operation is not checked or is not safe.
- Note: For details, specify the `-Xlint:unchecked` option and re-compile the program.

Also, when the `-Xlint` option is specified, the following warning message is output at the relevant location:

- Warning: Unchecked invocation to `add(E)` as the member of `[unchecked] raw type java.util.List`.

These warning and caution messages indicate that a check could not be performed during compilation. In the case of an actual error, a runtime exception occurs in JDK 1.4 as well. Therefore, if the program has operation results before migration, no action need be taken.

- Main compatibility items related to the language specifications in the change from JDK 1.4 to JDK 5.0

With the extension of the Java language specifications in J2SE 5.0, the type-guaranteed enumeration type has been added to the Java language. Therefore, `enum` becomes a keyword and cannot be used as an identifier.

- Effect of the addition of the `java.net.Proxy` class

With JDK 5.0, the `java.net.Proxy` class has been added in addition to the existing `java.lang.reflect.Proxy` class. Therefore, if you code `Proxy` in a class where an import declaration is specified, as shown in example of coding 1, a compilation error occurs because the package to which the `Proxy` class belongs is unclear.

Example of coding 1

```
import java.lang.reflect.*;
import java.net.*;
```

In this case, if you change the import declaration as shown in the example of coding 2, the compilation error can be avoided because the `Proxy` class of the `java.lang.reflect` package is clearly specified.

Example of coding 2

```
import java.lang.reflect.*;
import java.net.*;
import java.lang.reflect.Proxy;
```

- Change in the class initialization timing

With JDK 5.0 or earlier versions, a class was initialized when a class literal (such as `Foo.class`) was referenced. However, the class is not initialized with JDK 5.0 or later. To initialize the class when a class literal is referenced, you must add the initialization code. An example of coding is as follows:

Example of coding before the initialization code is added

```
Class c1 = Foo.class;
```

Example of coding after the initialization code is added

```
Class c1 = forceInit(Foo.class);
public static <T> Class<T> forceInit(Class<T> klass) {
    try {
        Class.forName(klass.getName(), true, klass.getClassLoader());
    } catch (ClassNotFoundException e) {
        throw new AssertionError(e); // Can't happen
    }
    return klass;
}
```

```
}

```

- Changes in the `java.lang.ClassLoader` class

According to the API specifications, you must specify a binary name delimited with a period (.) in the class name of the `ClassLoader` method with the `String` class name as the argument.

With JDK 5.0 or earlier, operations are performed without a problem even with a name delimited with forward slashes (/) (for example, `java/lang/String`). However, with JDK 5.0 or later, `java.lang.ClassNotFoundException` occurs if you specify a class name that is not a binary name. Specify the binary name of the class according to the API specifications in the method of the `ClassLoader` class.

- Changes in `java.util.logging.Level`

With JDK 5.0 or earlier, an exception does not occur even if you specify a null name argument in the constructor `java.util.logging.Level(String name, int value, String resourceName)`. However, with JDK 5.0 or later, `NullPointerException` occurs. Make sure you do not specify a null argument.

- Changes in `java.sql.Timestamp.compareTo`

With JDK 5.0, `java.sql.Timestamp.compareTo(java.util.Date)` is added to the API. The accuracy of this API is different compared to `java.util.Date.compareTo(java.util.Date)` in JDK 1.4.2, and is accurate up to nanoseconds. JDK 1.4.2 does not have this API, so `java.util.Date.compareTo(java.util.Date)` was invoked from an inheritance relationship, but with JDK 5.0 or later, `java.sql.Timestamp.compareTo(java.util.Date)` is invoked.

When all the following conditions are satisfied, the comparison results might differ with JDK 1.4.2 and JDK 5.0 or later:

- `java.sql.Timestamp.compareTo(java.util.Date o)` is invoked
- There is a difference of nanoseconds in the time in the `java.sql.Timestamp` object and the time in the argument `java.util.Date` object

To compare `java.sql.Timestamp` and `java.util.Date` with the same accuracy as JDK 1.4.2, you must use the `compareTo(java.util.Date)` method of `java.util.Date`, and pass `java.sql.Timestamp` to the argument.

- Serialized version UID

If a method of the included class[#] in a nested class (for example, `class-name-including-nested-class.this.method-name(argument)`;) is invoked, and if that included class is serializable, with JDK 5.0 or later, the value of the default serialized version UID of the included class varies from JDK 5.0 or earlier. This is because of the changes in the `javac` command implemented in JDK 5.0. If all of these conditions are satisfied, add the serialized version UID in the serializable class.

An included class is a class that includes a nested class.

(Example of coding)

```
import java.io.*;
class A implements Serializable { // serializable included class
void method1() {}
class B { // nested class
void method2() { A.this.method1(); }
}
}
```

- Effect on the method cancellation functionality (In Windows, AIX, and Linux)

In Windows, AIX, and Linux, compared to Application Server 06-70, the method cancellation functionality fails easily with Application Server 07-00 or later.

This is because of the addition of the protected area in Component Container 06-70-/G, and the upgrade from JDK 1.4.2 to JDK 5.0 in Application Server 07-00.

- Addition of protected areas in Component Container 06-70-/G

With Component Container 06-70-/G, multiple Java class library classes are added in a protected area. If method cancellation is executed with the Java class library classes added in the protected area, an inconsistency occurs in the internal status.

Component Container 07-00 or later inherits the protected areas of Component Container 06-70-/G or later, so there are several protected areas compared to Component Container 06-70-/G or earlier. For details on the contents defined as the protected area, see *Appendix C Contents of the Protected Area List* in the *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*.

- Upgrading to JDK 5.0 in Application Server 07-00

Method cancellation cannot be executed due to the restrictions in the Java specifications and JDK implementation. Such an area is assumed to be a special protected area without classes. The following table describes the differences in the specifications in Component Container 06-70 and 07-00 or later for handling the areas for which method cancellation cannot be executed due to the restrictions in the Java specifications and JDK implementation.

Table 15-1: Handling the areas for which method cancellation cannot be executed

Java program execution	Handling in Component Container 06-70	Handling in Component Container 07-00 or later
Synchronized and lock allocation pending	Method cancellation can be executed	Method cancellation cannot be executed
End of for and while loop	Method cancellation can be executed	Method cancellation cannot be executed
Generating objects (new keyword)	Method cancellation can be executed	
A Java program other than above is being executed	Method cancellation cannot be executed (however, method cancellation can be executed during the execution by an interpreter)	

Note that if method cancellation cannot be executed due to the restrictions in the Java specifications and JDK implementation, the details for the KDJE52718-W message display the class name (`JP.co.Hitachi.soft.jvm.CriticalClass.dummy(Native Method)`).

(2) In HP-UX

This subsection describes notes for HP-UX.

- Initial value of the stack size (`-Xss` option)

With JDK 1.4.2, the initial value of the Java thread stack size is 1 megabyte, but with JDK 6 or later, the initial value of the Java thread stack size is 4 megabytes.

(3) In Linux

This subsection describes notes for Linux.

- `waitFor()` and `exitValue()` methods of the `java.lang.Process` class

The exit codes of the child processes that can be obtained with the `waitFor()` and `exitValue()` methods was changed to 0 or more and 255 or less. If a child process terminates with a negative exit code, that value is considered as an unsigned integer.

15.2.3 Notes related to the differences in specifications with JDK 5.0 provided in Application Server Version 7 and Version 8

This subsection describes notes related to the differences in specifications with JDK 5.0 provided in Application Server Version 7 and Version 8.

(1) Notes common to OSs

This subsection describes notes common to OSs.

- Changes in the class files

With the extension of the class file format, the version of the class file generated by the default compilation of the `javac` command is changed from 49.0 to 50.0. Also, if you execute this class file in the JDK 5.0 or earlier execution environment, `java.lang.UnsupportedClassVersionError` is thrown.

- Changes in `java.io.File`

The implementation of `java.io.File.deleteOnExit()` is changed. The heap area that stores the file information to be deleted when Java VM terminates is changed from the C heap area to the Java heap area.

- Non-standard options of the java application startup tool

The following options are not supported from JDK 6:

- `-Xrunhprof`
- `-Xdebug`

The operations might not function properly if these options are specified. Do not specify these options. Note that if these options are specified, the operations are executed without returning an error.

- Changes in the `javac` command

Among the return codes for the execution of the `javac` command, the return code indicating that the compile target source was not found has changed (1 with JDK 5.0, 2 with JDK 6).

(2) Notes common to UNIX

This subsection describes notes common to UNIX.

- Changes in AWT

The AWT implementation is changed from Motif-based MAWT to the X Window System-based XAWT.

(3) In AIX

This subsection describes notes for AIX.

- Initial value of the stack size (`-Xss` option)

With JDK 5.0, the initial value of the Java thread stack size is 512 kilobytes, but with JDK 6 or later, the initial value of the Java thread stack size is 1 megabyte.

(4) In HP-UX

This subsection describes notes for HP-UX.

- Initial value of the stack size (`-Xss` option)

With JDK 5.0, the initial value of the Java thread stack size is 1 megabyte, but with JDK 6 or later, the initial value of the Java thread stack size is 4 megabytes.

15.3 Notes on using loggers beginning with sun.rmi

The J2EE server uses an RMI registry.

Also, the J2EE server uses loggers with the following names to collect the RMI implementation log:

- `sun.rmi.transport.tcp`
- `sun.rmi.server.call`
- `sun.rmi.client.call`

Note the following points when you use these loggers with a J2EE application:

- The RMI implementation log of the J2EE server is output.
- With the J2EE server, settings are specified so that the output from these loggers is not sent to the parent logger by using the `setUseParentHandlers(boolean)` method of the `java.util.logging.Logger` class. Therefore, the RMI implementation log cannot be output by the parent logger.
- With the J2EE server, the RMI implementation log is output for the log level `FINER`. When you change the log level of these loggers, do not set the following log levels:
 - `SEVERE`
 - `WARNING`
 - `INFO`
 - `CONFIG`
 - `FINE`
 - `OFF`

Appendix

A. Character Codes

HTTP is a protocol used mainly for downloading the data from the Web server to the browser, but is also used for uploading the data in the cases such as the HTML form.

The standards for passing on the character codes of the downloaded contents from the Web server to the browser are defined explicitly in the Servlet APIs as well. However, the handling of the character codes of the HTTP query strings and the HTTP request body was not clear in the earlier Servlet APIs (Servlet API 2.2 and earlier); therefore, each vendor handled the character codes differently.

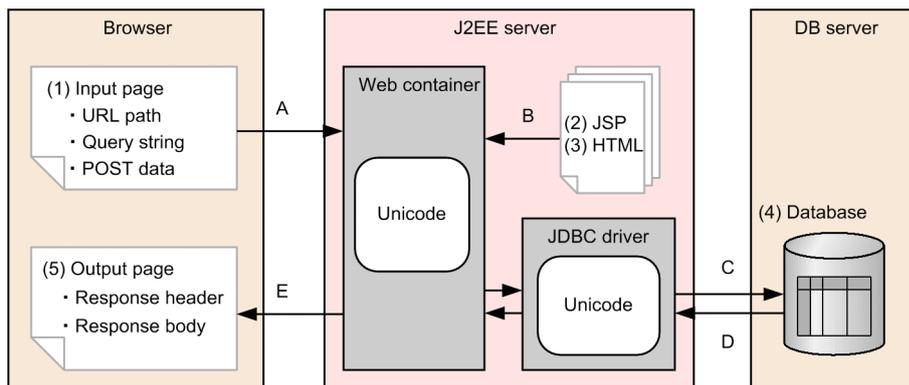
From Servlet API 2.3, you can specify the character codes when you reference the HTTP query strings and HTTP request body through the Servlet APIs (however, excluding the HTTP request body with the multipart/form-data encoding type). Such data is considered as the data of the specified character codes, is transformed to Unicode, which is the internal expression of Java, and is then passed to the application. If the resource character codes and the character code conversion to Unicode are incorrect, 'garbled characters' might occur, so the execution processes and the resource character codes must be considered during application development.

This section describes the character codes used in the applications and the notes. This section also describes the notes on character code conversion when the data is exchanged with the browser.

A.1 Character codes handled in an application

The following figure shows the flow of character code conversion on the basis of the application configuration.

Figure A-1: Flow of character code conversion in an application



The numbers in the figure ((1) to (5)) indicate the programs and resources for which the character codes must be taken into account. Also, the letters in the figure (A to E) indicate the flow of data during character code conversion. The following tables describe the character codes handled in the programs and resources and the notes, and the operations and notes on character code conversion respectively.

Table A-1: Character codes handled in the programs and resources and notes

Program and resource names	Items	Handled character codes and notes
(1) Input page	URL path	The character code of the URL path must be ISO-8859-1. Non-ASCII characters such as shift JIS cannot be coded.
	Query string	The characters sent using the HTML <i>FORM</i> tag are sent by URL encoding using the same character code as the HTML page displaying the form.
	POST data	
(2) JSP	JSP	<p>A JSP can be created using any character code.</p> <p>To specify the character encoding for the JSP documents in the Web applications of version 2.4 or later (JSP 2.0 specifications or later), specify the character encoding using XML declaration.</p> <p>With the JSP 1.2 specifications, the JSP character code must be coded in the <code>pageEncoding</code> attribute of the <code>page</code> directive.</p>

Program and resource names	Items	Handled character codes and notes
(3) HTML	HTML	An HTML page can be created using any character code.
(4) Database	Database	Determine the character code of the data stored in the database taking into account the character code displayed in the browser.
(5) Output page	Response header	The character code must be ISO-8859-1. When you use non-ASCII characters, the URL must be encoded.
	Response body	Any character code can be used for a user program.

Note

With a Web container, you cannot use character codes that express the alphanumeric characters in double bytes. The character codes that use double bytes for the alphanumeric characters are as follows:

- UCS-2 (ISO/IEC 10646)
- UCS-4 (ISO/IEC 10646-1)
- UTF-16

Table A-2: Operations and notes on character code conversion

Conversion location	Target	Operations and notes on character code conversion
A Browser to J2EE server	URL path	With a Web container, the character code for the URL path is processed as ISO-8859-1.
	Query string	The character code for the query string or POST data is determined in the application randomly. With the servlets and JSPs, the character code is handled with Unicode; therefore, convert the character code so that the character codes are consistent in the application.
	POST data	
B In the J2EE server	JSP files	The file is read with the encoding coded in the <code>pageEncoding</code> attribute of the <code>page</code> directive. If the <code>pageEncoding</code> attribute is omitted, the <code>contentType</code> attribute is used.
	HTML files	Sent to the browser using the character code of the HTML file as is.
C J2EE server to database	Database	Unicode is converted to the database storage character code using the JDBC driver.
D Database to J2EE server	Database	The database storage character code is converted to Unicode using the JDBC driver.
E J2EE server to browser	Response header	The Web container converts the character code of the response header to ISO-8859-1.
	Response body	For servlets Specify the character code using the <code>setContentType</code> method of the <code>ServletResponse</code> class. For JSPs Specify the character code in the <code>contentType</code> attribute of the <code>page</code> directive.

Reference note

Note the following about possible garbled characters:

The character code is handled as Unicode in the execution environment of an application. Therefore, a string sent from the browser is converted into Unicode once. Also, when the database is accessed, the character codes must be converted between Unicode and the database storage character codes, and during response, the character codes must be converted from Unicode to the response character code. If character code conversion is not performed appropriately, garbled characters are caused.

This occurs due to inclusion of the machine dependent characters in the character code called shift JIS or due to the presence of characters in which the results of Unicode conversion for the same characters differ from the other character codes[#]. For example, if the browser sends character data containing machine dependent characters and this data is converted to Unicode, if this string is converted to shift JIS during response, the result has garbled characters.

If the client OS is definable on Windows, garbled characters are avoided by specifying the character code as MS932 or Windows-31J, instead of shift JIS.

```
#
Includes characters such as -, ~, //, 「, and 」.
```

A.2 Character code conversion between the browser and application

This section describes the methods to be used for character code conversion between the browser and application, and the notes on the implementation of the JSPs and JavaScript.

(1) Methods used for character code conversion between the browser and application

Use the method of the `HttpServletRequest` class to obtain the query string and POST data sent using the browser. From among the methods of the `HttpServletRequest` class, the methods related to character code conversion and the notes on usage are as follows:

(a) `setCharacterEncoding` method

With this method, you set the character code used in the message body of the request, and in the parameters sent with the GET request query. This method must be executed before the input stream is read by using the `getParameter` method and the `getReader` method.

(a) `setCharacterEncoding` method

With this method, you set the character code used in the message body of the request, and in the parameters sent with the GET request query. This method must be executed before the input stream is read by using the `getParameter` method and the `getReader` method.

The following table describes the range in which the character code set up with the `setCharacterEncoding` method is used.

Table A-3: Range in which the character code set up with the `setCharacterEncoding` method is used

Method	Data sent from the HTML form (including queries)	Message body other than that described at the left
<code>getParameter</code>	A	--
<code>getParameterNames</code>	A	--
<code>getParameterValues</code>	A	--
<code>getParameterMap</code>	A	--
<code>getInputStream</code>	C	C
<code>getReader</code>	C	B
<code>getQueryString</code>	C	--

Legend:

A: The obtained value is converted to Unicode as the character code set up with the `setCharacterEncoding` method.

B: The character data is converted into the character code set up with the `setCharacterEncoding` method.

C: The encoded string is obtained.

--: Cannot be obtained.

(b) `getParameter` method

With this method, you specify the parameter name and obtain the value of the parameter included in the request. The obtained value is URL-decoded and converted to Unicode. Before you obtain the parameter name and parameter value, you must specify the character code by using the `setCharacterEncoding` method. If the character code is not specified, Unicode conversion is performed assuming ISO-8859-1. The same applies to the

`getParameterNames` method, `getParameterValues` method, and `getParameterMap` method of the `HttpServletRequest` class.

(c) `getInputStream` method

With this method, you obtain the stream for reading the binary data included in the message body of the request. The encoded string is obtained as is from the data sent from the HTML form; therefore, the obtained string must be decoded using an appropriate character code.

(d) `getReader` method

With this method, you use the `BufferedReader` class in the message body of the request and extract the data as the character data. The character data is converted to the same character code as the message body. The encoded string is obtained as is from the data sent from the HTML form; therefore, the obtained string must be decoded using an appropriate character code.

(e) `getQueryString` method

This method returns the query string included behind the requested URL path. The encoded string is obtained as is from the data sent from the HTML form; therefore, the obtained string must be decoded using an appropriate character code.

(2) Notes on including a file in a JSP

To include a file using the `include` directive of the JSP file, use the `contentType` attribute to specify the encoding in the JSP file that forms the include source. Also, specify the character code for the JSP file in the `pageEncoding` attribute. If the character code is not specified, the include destination characters might not be displayed normally.

(3) Notes on creating a query string using JavaScript

- If you use the `encodeURIComponent` function and `encodeURIComponent` function[#] of JavaScript, you can perform URL encoding using UTF-8. You use the string converted using these functions as the query string. In this case, if Unicode conversion is performed in the J2EE server as UTF-8, you can obtain the Japanese string data.

#

The dissimilarity between the `encodeURIComponent` function and `encodeURIComponent` function is whether to convert the special characters such as `'`, `/`, `?`, `:`, `@`, `&`, `=`, `+`, `$`, `,`, `#`. The `encodeURIComponent` function does not convert the special characters such as `'`, `/`, `?`, `:`, `@`, `&`, `=`, `+`, `$`, `,`, `#`.

- The operations of the JavaScript `escape` function differ depending on the type and version of the browser. A string converted using the `escape` function is not handled normally in the server; therefore, instead of using the `escape` function, use the `encodeURIComponent` function or the `encodeURIComponent` function.

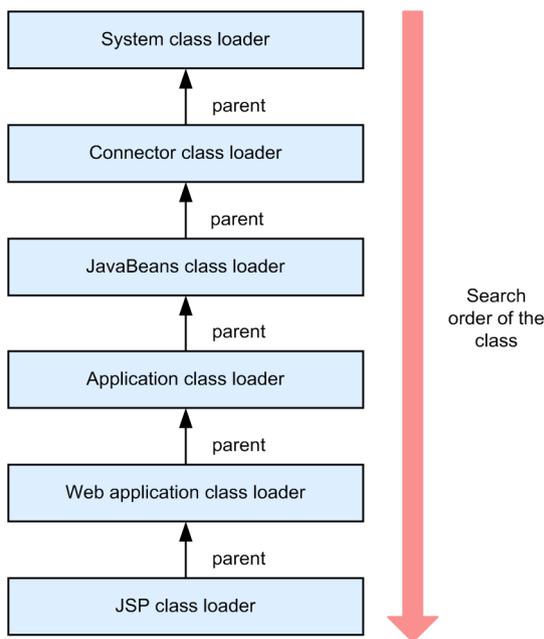
B. Configuration of the Class Loader

The class loader configuration of Application Server includes the default class loader configuration, the class loader configuration for using the local call optimization functionality, and the class loader configuration for downward compatibility. This appendix describes each of these class loader configurations and the class path set in the class loader.

B.1 Default class loader configuration

The following figure shows the default class loader configuration after a new installation.

Figure B-1: Default class loader configuration



The contents of the class loaders are as follows:

- **System class loader**
The system class loader loads the classes provided by the Cosminexus components.
 - Generation time: When a J2EE server starts
 - Destruction time: When a J2EE server stops
- **Connector class loader**
The connector class loader loads the classes included in an individually deployed resource adapter. Only one connector class loader exists in the J2EE server.
 - Generation time: When a J2EE server starts
 - Destruction time: When a J2EE server stops
- **JavaBeans class loader**
The JavaBeans class loader loads the classes of the JavaBeans resources.
 - Generation time: When a J2EE server starts
 - Destruction time: When a J2EE server stops
- **Application class loader**
The application class loader loads the classes included in the EJB-JAR, library JAR, and resource adapters existing in the application. There is an application class loader for each application. The class name is `com.hitachi.software.ejb.server.ApplicationClassLoader`.

- **Generation time:** When a J2EE application starts
KDJE42143-I is output to `cjmessage?.log` (J2EE server operation log) (? indicates the number of log files).
- **Destruction time:** When a J2EE application stops
KDJE42144-I is output to `cjmessage?.log` (J2EE server operation log) (? indicates the number of log files).

- **Web Application class loader**

The Web application class loader loads the classes included in a WAR file within a J2EE application. There is a Web application class loader for each WAR file. The class name is `org.apache.catalina.loader.WebappClassLoader`.

- **Generation time:** When a J2EE application starts (When a Web application starts)
KDJE39219-I is output to `cjmessage?.log` (J2EE server operation log) (? indicates the number of log files).
- **Destruction time:** When a J2EE application stops (When a Web application stops)
KDJE39220-I is output to `cjmessage?.log` (J2EE server operation log) (? indicates the number of log files).

- **JSP class loader**

The JSP class loader loads the classes generated when the JSP files and tag files are compiled. There is a JSP class loader for each JSP.

- **Generation time:** When a JSP class is loaded
- **Destruction time:** When a J2EE application stops (When a Web application stops)

Tip

Loading a resource adapter

A resource adapter deployed as a J2EE resource adapter is loaded using the connector class loader. A resource adapter that is included and used in a J2EE application is loaded using the application class loader.

! Important note

Destruction of the class loaders

After stopping a J2EE application, when you execute the finalize processing[#] (processing of the finalize method) of the Web application class loader or application class loader, the KDJE39220-I or KDJE42144-I message is output.

When a full garbage collection occurs after this message is output, the class loader and application are released from the heap.

[#]: The finalize processing is executed with a thread dedicated to finalize processing (finalizer thread). One finalizer thread always exists for JavaVM execution. The thread operates in parallel with the other Java threads and processes the `finalize()` methods of the objects one by one.

If the KDJE39220-I or KDJE42144-I message is not output, the possible causes are as follows:

Cause

(C-1): There is a software reference to an application class loader or application.

(C-2): An object allocated to the Explicit heap is referencing an application class loader or application.

For causes (C-1) and (C-2), implement the following systematic elimination of the causes (I-1) and (I-2) respectively, and check if the message is output:

Systematic elimination of the causes

(I-1): Specify `-XX:SoftRefLRUPolicyMSPerMB=0` and execute the operation.

For details on the `SoftRefLRUPolicyMSPerMB` option, see *7.8 Estimating the memory size of the Permanent area in the Java heap* in the *uCosminexus Application Server System Design Guide*.

(I-2): If the Explicit Memory Management functionality is enabled, disable the Explicit Memory Management functionality (`-XX:-HitachiUseExplicitMemory`).

For details on the Explicit Memory Management functionality, see *16.2 Details of the extended JavaVM options* in the *uCosminexus Application Server Definition Reference Guide*.

If causes (C-1) and (C-2) are not applicable and if the KDJE39220-I or KDJE42144-I message is not output, even if the application is started or stopped repeatedly, a memory leak might have occurred.

B.2 Class loader configuration for local call optimization

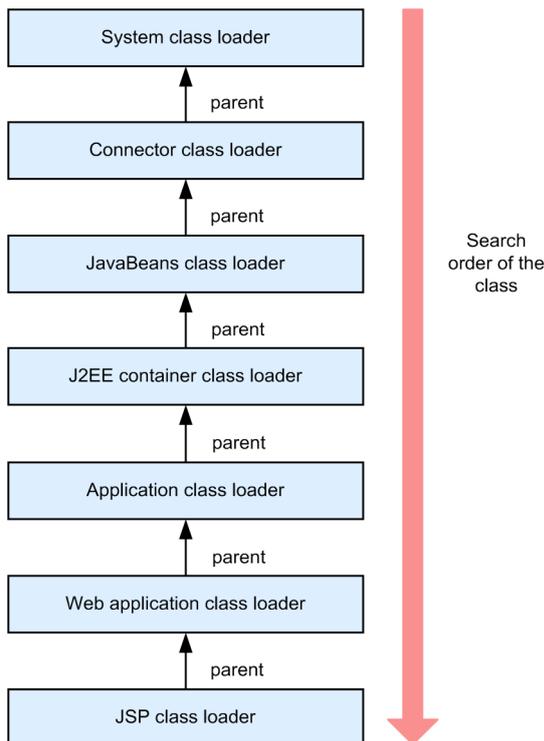
When you use the local call optimization functionality, the class loader configuration differs from the default class loader. The local call optimization functionality enables the invocation of the Enterprise Beans of other applications at a high speed.

To use the local call optimization functionality, specify the following settings in the user-defined file for the J2EE servers (*Cosminexus-installation-directory*\CC\server\usrconf\ejb*server-name* \usrconf.properties):

```
ejbserver.rmi.localinvocation.scope=all
```

The class loader configuration for local call optimization is as follows:

Figure B-2: Class loader configuration for local call optimization



The contents of the class loaders are as follows:

- **System class loader**
 The system class loader loads the classes provided by the Cosminexus components.
 Generation time: When a J2EE server starts
 Destruction time: When a J2EE server stops
- **Connector class loader**
 The connector class loader loads the classes included in an individually deployed resource adapter. Only one connector class loader exists in the J2EE server.
 Generation time: When a J2EE server starts
 Destruction time: When a J2EE server stops
- **JavaBeans class loader**
 The JavaBeans class loader loads the classes of the JavaBeans resources.
 Generation time: When a J2EE server starts
 Destruction time: When a J2EE server stops
- **J2EE container class loader**

The J2EE container class loader becomes the parent class loader of all the application class loaders and sets the class path of all the application class loaders. Only one J2EE container class loader exists in a J2EE server.

Generation time: When a J2EE server starts

Destruction time: When a J2EE server stops

- **Application class loader**

The application class loader has in the class path the classes included in the EJB-JAR, library JAR, and resource adapters existing in the application. There is an application class loader for each application. During local call optimization, the class path of the application class loaders is also set in the J2EE container class loader that forms the parent class loader; therefore, no classes are loaded with this class loader. The class name is `com.hitachi.software.ejb.server.ApplicationClassLoader`.

- Generation time: When a J2EE application starts
KDJE42143-I is output to `cjmessage?.log` (J2EE server operation log) (? indicates the number of log files).
- Destruction time: When a J2EE application stops
KDJE42144-I is output to `cjmessage?.log` (J2EE server operation log) (? indicates the number of log files).

- **Web Application class loader**

The Web application class loader loads the classes included in a WAR file within a J2EE application. There is a Web application class loader for each WAR file. The class name is `org.apache.catalina.loader.WebappClassLoader`.

- Generation time: When a J2EE application starts (When a Web application starts)
KDJE39219-I is output to `cjmessage?.log` (J2EE server operation log) (? indicates the number of log files).
- Destruction time: When a J2EE application stops (When a Web application stops)
KDJE39220-I is output to `cjmessage?.log` (J2EE server operation log) (? indicates the number of log files).

- **JSP class loader**

The JSP class loader loads the classes generated when the JSP files and tag files are compiled. There is a JSP class loader for each JSP.

- Generation time: When a JSP class is loaded
- Destruction time: When a J2EE application stops (When a Web application stops)

For the notes on the destruction of a class loader, see *Appendix B.1 Default class loader configuration*.

Tip

Loading a resource adapter

A resource adapter deployed as a J2EE resource adapter is loaded using the connector class loader. A resource adapter that is included and used in a J2EE application is loaded using the application class loader.

! Important note

Note the following points when you set `ejbserver.rmi.localinvocation.scope=all` in the property:

- The EJB-JARs and library JARs of the J2EE application are loaded using the same class loader, so the classes with the same name are shared between the J2EE applications. If the classes with the same name, but different contents exist, only the class loaded first is enabled as per the Java specifications. Note that the classes in a WAR file are loaded using the class loaders existing for the WAR files, so these classes are not shared.
 - To replace the EJB-JARs and library JARs of a J2EE application, stop the J2EE application and then restart the J2EE server. The settings will be enabled after restart.
-

B.3 Class path set in the class loader

The class path set in the class loader and the order in which the classes are searched when the class loader loads the classes is as follows:

B. Configuration of the Class Loader

- Connector class loader

The connector class loader loads the stand-alone RAR classes. This class loader sets the JAR files included in the individually deployed RAR file in the class path.

- J2EE container class loader

The J2EE container class loader is enabled when `ejbserver.rmi.localinvocation.scope=all` (using the local call optimization functionality) is specified. This class loader becomes the parent class loader of all the application class loaders and has the class paths of all the running application class loaders in its own class path.

- Application class loader

The application class loader loads the EJB-JAR, library JAR, and container-generated classes existing in the J2EE application. The sequence of the specified class path is as follows:

1. JAR files included in the RAR that is included in the J2EE application
2. EJB-JARs included in the J2EE application
3. Library of the referencing method
4. Library JAR included in the J2EE application
5. EJB-JAR files of another application being referenced by the Naming Service switching functionality (when the reference destination application exists on the same J2EE server)

C. Contract Between the JPA Provider and EJB Container

This section describes the contract between the JPA provider and EJB container.

C.1 Runtime-related contract

The runtime-related contract includes the responsibilities of the container and the responsibilities of the JPA provider.

(1) Responsibilities of the container

- **Contract related to the transaction scope persistence context**

If the transaction scope persistence context is used, the container executes the following processing if the entity manager is not associated with a JTA transaction:

- The container invokes `EntityManagerFactory.createEntityManager` in the following cases and creates a new entity manager:
 - If the entity manager method that uses the transaction scope persistence context is invoked for the first time in a business method within the scope of the JTA transaction
- After the JTA transaction is concluded (committed or rolled back), the container invokes `EntityManager.close` to close the entity manager.

Also, if the container satisfies all the following conditions, `TransactionRequiredException` is thrown:

- When the transaction scope persistence context is used
- When the transaction is not active
- When the application invokes the `persist`, `remove`, `merge`, and `refresh` methods of `EntityManager`

- **Contract related to the extended persistence context**

If the extended persistence context is used, the container executes the following processing:

- The container invokes `EntityManagerFactory.createEntityManager` in the following cases and creates a new entity manager:
 - If the references to the entity manager using the extended persistence context are defined when the Stateful Session Bean instance is created
- The container invokes `EntityManager.close` at the following timing and closes the entity manager:
 - When the Stateful Session Bean that created the entity manager and the Stateful Session Bean that inherited the same persistence context are deleted
- When a business method of the Stateful Session Bean that uses a container-managed transaction is invoked, if the entity manager is not associated with a JTA transaction, the container associates the entity manager with the JTA transaction and invokes `EntityManager.joinTransaction`. If another entity manager is already associated with the JTA transaction, the container throws `EJBException`.
- When `UserTransaction.begin` is invoked in a business method of the Stateful Session Bean that uses a bean-managed transaction, the container associates the entity manager with the JTA transaction and invokes `EntityManager.joinTransaction`. For details on the bean-managed transactions, see 2.7.2 *BMT* in the *uCosminexus Application Server EJB Container Functionality Guide*.

- **Contract related to the container-managed entity manager**

If the application invokes `EntityManager.close` when the container-managed entity manager is being used, the container throws `IllegalStateException`.

If the property is specified in `@PersistenceContext` or in the `<persistence-context-ref>` tag of the DD, the container uses the `EntityManagerFactory.createEntityManager(Map map)` method, creates the entity manager, includes the specified property in the `map` argument, and passes the property to the JPA provider.

- **Automatic connection closing functionality**

In Application Server, the connection obtained by the JPA provider with the extension of the Session Bean and Web components sometimes closes automatically by the automatic closing functionality of the container. The automatic connection closing functionality is used to prevent a connection leak.

The connection is subject to being closed automatically if the following conditions are satisfied:

- A connection obtained by a Stateless Session Bean is closed automatically when returned from a business method.
- A connection obtained by a Stateful Session Bean is closed automatically when the Stateful Session Bean is destroyed.
- A connection obtained by a Web component is closed automatically when returned from a service method.

However, even if the above conditions are applicable, if the connection is participating in a JTA transaction, automatic close is reserved until the JTA transaction is committed.

(2) Responsibilities of the JPA provider

Whether the entity manager to be used with an application is defined to use the transaction scope persistence context or the extended persistence context, is not transmitted to the JPA provider. The responsibilities of the JPA provider include creating the entity manager when requested by the container and registering Synchronization in the transaction in order to receive the notification about transaction conclusion from the transaction.

- When the container invokes `EntityManagerFactory.createEntityManager`, the JPA provider must create a new entity manager and return the created entity manager to the container. If a JTA transaction is active, the JPA provider must register Synchronization in the JTA transaction.
- When the container invokes `EntityManager.joinTransaction`, the JPA provider must register Synchronization in the JTA transaction. However, if `joinTransaction` was invoked previously and Synchronization is already registered in the JTA transaction, nothing need be done.
- When a JTA transaction is committed, the JPA provider must flush all the changed entity statuses in the database.
- When a JTA transaction is rolled back, the JPA provider must detach all the managed entities.
- When the JPA provider throws an exception that causes transaction rollback, the JPA provider must mark the transaction for rollback.
- When the container invokes `EntityManager.close`, the JPA provider must release all the allocated resources after all the unresolved transactions related to that entity manager are concluded. If the entity manager is already closed, the JPA provider must throw `IllegalStateException`.
- When the container invokes `EntityManager.clear`, the JPA provider must detach all the managed entities.

(3) `javax.transaction.TransactionSynchronizationRegistry` interface

The JPA provider can use the `TransactionSynchronizationRegistry` interface to register Synchronization in a transaction and to mark a transaction for rollback. The `TransactionSynchronizationRegistry` instance can be looked up with the name `java:comp/TransactionSynchronizationRegistry` using the JNDI.

The interface definition is as follows:

```
package javax.transaction;

/**
 * This interface is used from system level components
 * of Application Server such as the
 * JPA provider and resource adapters.
 * Using this interface, you can
 * register synchronization invoked in a particular order,
 * register the resource object in the current transaction,
 * obtain the current transaction context,
 * obtain the current transaction status,
 * and mark the current transaction for rollback.
 *
 * This interface is implemented by Application Server
 * as a stateless service object.
 * The same object can be used from multiple components
 * in a multi-thread safe manner.

```

```

*
* With default Application Server, the instance implementing this
* interface can be looked up with the default name using the JNDI.
* The default name is
* java:comp/TransactionSynchronizationRegistry.
*/
public interface TransactionSynchronizationRegistry {

    /**
    * When this method is invoked, a unique object expressing the
    * transaction associated with the current thread is returned.
    * The hashCode and equals method of this object is overridden
    * and can be used as the hashmap key.
    * If the transaction does not exist, null is returned.
    *
    * All the objects returned by invoking this method in the same
    * transaction context of same Application Server have the same
    * hashCode and the comparison results in the equal method are
    * true.
    *
    * The toString method returns the transaction context information
    * as a string in an easy-to-read format.
    * However, the string format returned by toString is not defined.
    * Also, the compatibility of the toString results between versions
    * is not guaranteed.
    *
    * There is no guarantee that the obtained object can be serialized
    * and the operations when the object is sent outside JavaVM are
    * not defined.
    *
    * @return Object that uniquely expresses the transaction
    * associated with the thread when this method is invoked.
    */
    Object getTransactionKey();

    /**
    * The object is added or replaced in the resource map
    * of the transaction associated with the thread used when this
    * method is invoked.
    * The map key must be a class defined on the method invocation
    * side so that conflict does not occur.
    * The class used as the key must have the appropriate hashCode
    * and equals method as the map key.
    * The map key and value is not evaluated and used by this class.
    * The general contract of this method is the same as the put
    * method of Map and the key must be other than null, but the
    * value can be set as null.
    * If a value associated with the key already exists, the value
    * is replaced.
    *
    * @param key Entry key of map
    * @param value Entry value of map
    * @exception IllegalStateException When an active transaction
    * does not exist
    * @exception NullPointerException When the argument key is null
    */
    void putResource(Object key, Object value);

    /**
    * The object is extracted from the resource map of the transaction
    * associated with the thread used when this method is invoked.
    * The key must be the same as the object specified in the
    * putResource method beforehand, in the same transaction.
    * If the specified key does not exist in the current resource
    * map, null is returned.
    * The general contract of this method is the same as the put
    * method of Map and the key must be other than null, but the
    * value can be set as null.
    * If the key is not stored in the map or if a null value is stored
    * for the key, the return value is null.
    *
    * @param key Entry key of map
    * @return Value associated with the key
    * @exception IllegalStateException When an active transaction
    * does not exist
    */

```

C. Contract Between the JPA Provider and EJB Container

```
* @exception NullPointerException When the argument key is null
*/
Object getResource(Object key);

/**
 * Registers the synchronization instances invoked in a particular
 * order.
 * beforeCompletion of Synchronization registered with this
 * method is invoked after
 * SessionSynchronization.beforeCompletion, and
 * Synchronization.beforeCompletion, which is directly
 * registered in a transaction, are invoked, and before the 2-phase
 * commit processing starts.
 * Similarly, afterCompletion of Synchronization registered with
 * this method is invoked after the completion of 2-phase commit
 * processing and before SessionSynchronization.afterCompletion
 * and Synchronization.afterCompletion, which is directly
 * registered in the transaction, are invoked.
 *
 * beforeCompletion is invoked by the transaction context
 * associated with the thread when this method is invoked.
 * With beforeCompletion, access to resources such as connector
 * is permitted, but access is not permitted to user components
 * such as timer service and bean methods.
 * This is because the data managed on the invocation side and
 * the data that is already flushed by other Synchronization
 * registered with registerInterposedSynchronization might be
 * changed.
 * The general context becomes the context of the component
 * that invokes registerInterposedSynchronization.
 *
 * The context when afterCompletion is invoked is not defined.
 * Note that access to user components is not allowed.
 * Also, the resource can be closed, but
 * transactional operations cannot be performed for the resource.
 *
 * If this method is invoked when the transaction is not active,
 * IllegalStateException is thrown.
 *
 * If this method is invoked after the 2-phase commit processing
 * starts, IllegalStateException is thrown.
 *
 * @param sync Instance of Synchronization to be registered
 * @exception IllegalStateException When an active transaction
 * does not exist
 */
void registerInterposedSynchronization(Synchronization sync);

/**
 * When this method is invoked, the status of the transaction
 * associated with the thread is returned.
 * The return value of this method is the same as the result of *
 * TransactionManager.getStatus().
 *
 * @return Status of the transaction associated with
 * the thread when this method is invoked.
 */
int getTransactionStatus();

/**
 * When this method is invoked, the transaction associated with
 * the thread is marked for rollback.
 *
 * @exception IllegalStateException When an active transaction
 * does not exist
 */
void setRollbackOnly();

/**
 * When this method is invoked, returns information about whether
 * the transaction associated with the thread is marked for
 * rollback.
 *
 * @return true if the transaction is marked for rollback
 */
```

```

    * @exception IllegalStateException When an active transaction
    * does not exist
    */
    boolean getRollbackOnly();
}

```

C.2 Deployment-related contract

The deployment-related contract includes the responsibilities of the container and the responsibilities of the JPA provider.

(1) Responsibilities of the container

During deployment, the container searches `persistence.xml` packaged at a decided location within the application. If `persistence.xml` exists in the application, the container processes the definition of the persistence unit defined in `persistence.xml`. For details on the locations searched by the container, see *5.8 Definitions in persistence.xml*.

The container verifies `persistence.xml` file using `persistence_1_0.xsd`. If the verification results in an error, the container reports to the user. If the provider and data source information is not specified in `persistence.xml`, the default value is used. For details on the default values used, see *5.8 Definitions in persistence.xml*. When creating the entity manager factory of the persistence unit, the container passes the properties to the JPA provider.

The container creates the implementation class instance of `javax.persistence.spi.PersistenceProvider` defined for each persistence unit in `persistence.xml`, invokes the `createContainerEntityManagerFactory` method, and obtains `EntityManagerFactory` for creating the container-managed entity manager. The Meta data of the persistence unit is passed as the `PersistenceUnitInfo` object to the JPA provider using the argument of the `createContainerEntityManagerFactory` method. The container creates only one `EntityManagerFactory` for one persistence unit definition and creates multiple `EntityManagers` from that `EntityManagerFactory`.

When the persistence unit is re-deployed, the container invokes the `close` method of `EntityManagerFactory` that is already obtained and then invokes `createContainerEntityManagerFactory` along with the new `PersistenceUnitInfo`.

(2) Responsibilities of the JPA provider

The JPA provider must implement `PersistenceProvider` SPI, and when the `createContainerEntityManagerFactory` method of `PersistenceProvider` is invoked, the JPA provider must use the Meta data (`PersistenceUnitInfo`) of the persistence unit passed in the argument to create `EntityManagerFactory`, and return the created `EntityManagerFactory` to the container.

The JPA provider processes the Meta data annotation of the managed class (such as an entity class) included in the persistence unit. Also, when the O/R mapping file is used in the persistence unit, the JPA provider must interpret the file. At this time, the JPA provider verifies the O/R mapping file by using `orm_1_0.xsd` and must notify the user if an error occurs.

(3) `javax.persistence.spi.PersistenceProvider` interface

The JPA provider must implement the `javax.persistence.spi.PersistenceProvider` interface. This interface is invoked by the container and is not invoked from the application. The `PersistenceProvider` implementation class must be public and must have a constructor without argument.

The `javax.persistence.spi.PersistenceProvider` interface is as follows:

```

package javax.persistence.spi;

/**
 * Interface implemented by the JPA provider.
 * This interface is used for creating EntityManagerFactory.
 * In the Java EE environment, the interface is invoked by the container
 * and in the JavaSE environment, the interface is invoked by the

```

```

* persistence class.
*/
public interface PersistenceProvider {

    /**
     * Invoked when the persistence class creates EntityManagerFactory.
     *
     * @param emName Name of the persistence unit
     * @param map property Map used by the JPA provider.
     * The property specified here is used to overwrite the property
     * corresponding to the persistence.xml file or to specify
     * the property that is not specified in the persistence.xml file.
     * (If the property need not be specified, null is passed)
     * @return EntityManagerFactory of persistence unit
     * If the JPA provider is incorrect, null is returned.
     */
    public EntityManagerFactory createEntityManagerFactory(String
emName, Map map);

    /**
     * Invoked when the container creates EntityManagerFactory.
     *
     * @param info Meta data to be used by the JPA provider
     * @param map Integration level property to be used by the JPA provider
     * (if not specified, null is passed)
     * @return EntityManagerFactory of the persistence unit specified
     * in Meta data
     */
    public EntityManagerFactory createContainerEntityManagerFactory(
PersistenceUnitInfo info, Map map);
}

```

(4) javax.persistence.spi.PersistenceUnitInfo interface

The javax.persistence.spi.PersistenceUnitInfo interface definition is as follows:

```

import javax.sql.DataSource;
/**
 * This interface is implemented by the container and is passed to
 * the JPA provider when EntityManagerFactory is created.
 */
public interface PersistenceUnitInfo {

    /**
     * @return Persistence unit name defined in persistence.xml
     */
    public String getPersistenceUnitName();

    /**
     * @return Fully qualified class name of the JPA provider
     * implementation class defined in the <provider> element of
     * persistence.xml
     */
    public String getPersistenceProviderClassName();

    /**
     * @return Returns the transaction type of EntityManager
     * created by EntityManagerFactory
     * Type specified in the transaction-type attribute of
     * persistence.xml.
     */
    public PersistenceUnitTransactionType getTransactionType();

    /**
     * @return Returns a data source with JTA enabled
     * for use by the JPA provider.
     * Data source specified in the <jta-data-source> element of
     * persistence.xml or the data source determined by the container
     * during deployment.
     */
    public DataSource getJtaDataSource();

    /**

```

```

* @return Returns a data source with JTA disabled for the JPA provider
* to access the data outside the JTA transaction.
* Data source specified in <non-jta-data-source> element of
* persistence.xml or the data source determined by the container
* during deployment.
*/
public DataSource getNonJtaDataSource();

/**
* @return List of mapping file names that must be loaded for the
* JPA provider to determine the entity class mapping.
* The mapping file must have the standard XML mapping format.
* The mapping file must have a unique name and must be
* loadable as a resource from the application class path.
* The mapping file names are specified in the
* <mapping-file> tag of persistence.xml.
*/
public List<String> getMappingFileNames();

/**
* Returns the URL list of JAR files or directory deploying the
* JAR files, required for searching the managed class of the
* persistence unit by the JPA provider.
* Each URL is specified in the <jar-file> tag of the persistence.xml
* file.
* The URL is in a file URL format indicating JAR files or directory
* deploying the JAR files or in another URL format that can obtain
* InputStream in the JAR format.
*
* @return List of URL objects indicating the JAR files or directories
*/
public List<URL> getJarFileUrls();

/**
* Returns the URL of the JAR file or directory forming the persistence
* unit root
* (If the persistence unit root is WEB-INF/classes directory,
* returns the URL of WEB-INF/classes directory)
* The URL is in a file URL format indicating JAR files or directory
* deploying the JAR files or in another URL format that can obtain
* InputStream in the JAR format.
*
* @return URL object indicating JAR files or directories
*/
public URL getPersistenceUnitRootUrl();

/**
* @return List of class names that the JPA provider must handle
* as managed classes.
* Each class name is specified in the <class> tag of the
* persistence.xml file
*/
public List<String> getManagedClassNames();

/**
* @return Returns whether to handle the class that is deployed
* in the persistence unit root and is not explicitly specified
* as managed class, as a managed class.
* This value is specified in the <exclude-unlisted-classes> tag
* of the persistence.xml file.
*/
public boolean excludeUnlistedClasses();

/**
* @return Properties object.
* Each property is specified in the <property> tag of the
* persistence.xml file.
*/
public Properties getProperties();

/**
* @return ClassLoader that can be used by the JPA provider
* to load classes and resources and to open URLs.
*/

```

```

public ClassLoader getClassLoader();

/**
 * Class loader returned by the PersistenceUnitInfo.getClassLoader
 * method and registers the JPA provider transformer that is invoked
 * every time a new class is defined or a class is re-defined.
 * This transformer is returned by the
 * PersistenceUnitInfo.getNewTempClassLoader method
 * and does not affect the classes loaded by the class loader.
 * Even if some persistence units are defined in the class loading
 * scope, the class is only converted once in the same class loading
 * scope.
 *
 * @param transformer JPA provider transformer invoked by the
 * container to define (re-define) a class.
 */
public void addTransformer(ClassTransformer transformer);

/**
 * Returns a new instance of the class loader
 * that can be used temporarily by the JPA provider to load classes
 * and resources and to open the URLs.
 * The scope and class path of this class loader
 * is exactly similar to the class loader returned
 * by PersistenceUnitInfo.getClassLoader.
 * The classes loaded with this class loader cannot
 * be referenced from the application components.
 * The JPA provider can use this class loader only in
 * the extended invocation of createContainerEntityManagerFactory.
 *
 * @return Temporary class loader having the same scope
 * or class path as the current class loader.
 */
public ClassLoader getNewTempClassLoader();
}

```

Reference note

With Application Server, if the JTA data source and non-JTA data source are not defined in the persistence unit, `getJtaDataSource()` or `getNonJtaDataSource()` return null. The preceding text "if the JTA data source and non-JTA data source are not defined in the persistence unit" indicates the following state:

- When `<jta-data-source>` and `<non-jta-data-source>` are omitted in `persistence.xml` and the default value is not defined in the system properties `ejbserver.jpa.defaultJtaDsName` and `ejbserver.jpa.defaultNonJtaDsName`
 - When the system properties `ejbserver.jpa.overrideJtaDsName` and `ejbserver.jpa.overrideNonJtaDsName` are also not defined
-

D. BNF for JPQL

This appendix describes BNF for JPQL provided in the JPA 1.0 specifications.

Table D-1: Rules for the BNF expressions

Expression	Contents
{ }	Indicates a group.
[]	Indicates the option syntax.
*	Indicates 0 or more.
A B	Indicates A or B.

Note that the bold parts indicate keywords.

BNF is as follows:

```

QL_statement ::= select_statement | update_statement | delete_statement
select_statement ::= select_clause from_clause [where_clause] [groupby_clause]
[having_clause] [orderby_clause]
update_statement ::= update_clause [where_clause]
delete_statement ::= delete_clause [where_clause]
from_clause ::=
FROM identification_variable_declaration
{, {identification_variable_declaration | collection_member_declaration}}*
identification_variable_declaration ::= range_variable_declaration { join |
fetch_join }*
range_variable_declaration ::= abstract_schema_name [AS]
identification_variable
join ::= join_spec join_association_path_expression [AS]
identification_variable
fetch_join ::= join_spec FETCH join_association_path_expression
association_path_expression ::=
collection_valued_path_expression | single_valued_association_path_expression
join_spec ::= [ LEFT [OUTER] | INNER ] JOIN
join_association_path_expression ::= join_collection_valued_path_expression |
join_single_valued_association_path_expression
join_collection_valued_path_expression ::=
identification_variable.collection_valued_association_field
join_single_valued_association_path_expression ::=
identification_variable.single_valued_association_field
collection_member_declaration ::=
IN (collection_valued_path_expression) [AS] identification_variable
single_valued_path_expression ::=
state_field_path_expression | single_valued_association_path_expression
state_field_path_expression ::=
{identification_variable |
single_valued_association_path_expression}.state_field
single_valued_association_path_expression ::=
identification_variable.{single_valued_association_field.}*
single_valued_association_field
collection_valued_path_expression ::=
identification_variable.
{single_valued_association_field.}*collection_valued_association_field
state_field ::= {embedded_class_state_field.}*simple_state_field
update_clause ::= UPDATE abstract_schema_name [[AS] identification_variable]
SET update_item {, update_item}*
update_item ::= [identification_variable.]{state_field |
single_valued_association_field} =
new_value
new_value ::=
simple_arithmetic_expression |
string_primary |
datetime_primary |
boolean_primary |
enum_primary
simple_entity_expression |
NULL
delete_clause ::= DELETE FROM abstract_schema_name [[AS]

```

```

identification_variable]
select_clause ::= SELECT [DISTINCT] select_expression {, select_expression}*
select_expression ::=
single_valued_path_expression |
aggregate_expression |
identification_variable |
OBJECT(identification_variable) |
constructor_expression
constructor_expression ::=
NEW constructor_name ( constructor_item {, constructor_item}* )
constructor_item ::= single_valued_path_expression | aggregate_expression
aggregate_expression ::=
{ AVG | MAX | MIN | SUM } ([DISTINCT] state_field_path_expression) |
COUNT ([DISTINCT] identification_variable | state_field_path_expression |
single_valued_association_path_expression)
where_clause ::= WHERE conditional_expression
groupby_clause ::= GROUP BY groupby_item {, groupby_item}*
groupby_item ::= single_valued_path_expression | identification_variable
having_clause ::= HAVING conditional_expression
orderby_clause ::= ORDER BY orderby_item {, orderby_item}*
orderby_item ::= state_field_path_expression [ ASC | DESC ]
subquery ::= simple_select_clause subquery_from_clause [where_clause]
[groupby_clause] [having_clause]
subquery_from_clause ::=
FROM subselect_identification_variable_declaration
{, subselect_identification_variable_declaration}*
subselect_identification_variable_declaration ::=
identification_variable_declaration |
association_path_expression [AS] identification_variable |
collection_member_declaration
simple_select_clause ::= SELECT [DISTINCT] simple_select_expression
simple_select_expression ::=
single_valued_path_expression |
aggregate_expression |
identification_variable
conditional_expression ::= conditional_term | conditional_expression OR
conditional_term
conditional_term ::= conditional_factor | conditional_term AND
conditional_factor
conditional_factor ::= [ NOT ] conditional_primary
conditional_primary ::= simple_cond_expression | (conditional_expression)
simple_cond_expression ::=
comparison_expression |
between_expression |
like_expression |
in_expression |
null_comparison_expression |
empty_collection_comparison_expression |
collection_member_expression |
exists_expression
between_expression ::=
arithmetic_expression [NOT] BETWEEN
arithmetic_expression AND arithmetic_expression |
string_expression [NOT] BETWEEN string_expression AND string_expression |
datetime_expression [NOT] BETWEEN
datetime_expression AND datetime_expression
in_expression ::=
state_field_path_expression [NOT] IN ( in_item {, in_item}* | subquery)
in_item ::= literal | input_parameter
like_expression ::=
string_expression [NOT] LIKE pattern_value [ESCAPE escape_character]
null_comparison_expression ::=
{single_valued_path_expression | input_parameter} IS [NOT] NULL
empty_collection_comparison_expression ::=
collection_valued_path_expression IS [NOT] EMPTY
collection_member_expression ::= entity_expression
[NOT] MEMBER [OF] collection_valued_path_expression
exists_expression ::= [NOT] EXISTS (subquery)
all_or_any_expression ::= { ALL | ANY | SOME } (subquery)
comparison_expression ::=
string_expression comparison_operator {string_expression |
all_or_any_expression} |
boolean_expression { = | <> } {boolean_expression | all_or_any_expression} |
enum_expression { = | <> } {enum_expression | all_or_any_expression} |

```

```

datetime_expression comparison_operator
{datetime_expression | all_or_any_expression} |
entity_expression { = | <> } {entity_expression | all_or_any_expression} |
arithmetic_expression comparison_operator
{arithmetic_expression | all_or_any_expression}
comparison_operator ::= = | > | >= | < | <= | <>
arithmetic_expression ::= simple_arithmetic_expression | (subquery)
simple_arithmetic_expression ::=
arithmetic_term | simple_arithmetic_expression { + | - } arithmetic_term
arithmetic_term ::= arithmetic_factor | arithmetic_term { * | / }
arithmetic_factor
arithmetic_factor ::= [{ + | - }] arithmetic_primary
arithmetic_primary ::=
state_field_path_expression |
numeric_literal |
(simple_arithmetic_expression) |
input_parameter |
functions_returning_numerics |
aggregate_expression
string_expression ::= string_primary | (subquery)
string_primary ::=
state_field_path_expression |
string_literal |
input_parameter |
functions_returning_strings |
aggregate_expression
datetime_expression ::= datetime_primary | (subquery)
datetime_primary ::=
state_field_path_expression |
input_parameter |
functions_returning_datetime |
aggregate_expression
boolean_expression ::= boolean_primary | (subquery)
boolean_primary ::=
state_field_path_expression |
boolean_literal |
input_parameter |
enum_expression ::= enum_primary | (subquery)
enum_primary ::=
state_field_path_expression |
enum_literal |
input_parameter |
entity_expression ::=
single_valued_association_path_expression | simple_entity_expression
simple_entity_expression ::=
identification_variable |
input_parameter
functions_returning_numerics ::=
LENGTH(string_primary) |
LOCATE(string_primary, string_primary[, simple_arithmetic_expression]) |
ABS(simple_arithmetic_expression) |
SQRT(simple_arithmetic_expression) |
MOD(simple_arithmetic_expression, simple_arithmetic_expression) |
SIZE(collection_valued_path_expression)
functions_returning_datetime ::=
CURRENT_DATE |
CURRENT_TIME |
CURRENT_TIMESTAMP
functions_returning_strings ::=
CONCAT(string_primary, string_primary) |
SUBSTRING(string_primary,
simple_arithmetic_expression, simple_arithmetic_expression) |
TRIM([[trim_specification] [trim_character] FROM] string_primary) |
LOWER(string_primary) |
UPPER(string_primary)
trim_specification ::= LEADING | TRAILING | BOTH

```

E. Use Cases for Cosminexus JMS Provider

This appendix describes the use cases for Cosminexus JMS Provider. Note that operations other than the use cases described in this section might not function properly.

To use Cosminexus JMS Provider, execute the environment setup and operations as per the procedure described in this section. Cosminexus JMS Provider might not operate correctly if the setup and operations are not executed with the procedure described here.

The following table lists the use case types that will be described in this appendix.

Table E-1: Types of use cases for Cosminexus JMS Provider

Category	Task	Task contents	Task details	Reference location	
Setup	New environment setup	Sets up a new environment.	Environment setup for using Cosminexus JMS Provider	<i>Appendix E. 3</i>	
	Environment re-setup	Resets up the environment.	Adding applications that use Cosminexus JMS Provider	<i>Appendix E. 4</i>	
			Deleting applications that use Cosminexus JMS Provider	<i>Appendix E. 5</i>	
Operation	Periodic operations	Executes operations that are required periodically (daily or other operation timing) for using Cosminexus JMS Provider	Starting the Cosminexus JMS Provider services (for the initial startup)	<i>Appendix E. 6</i>	
			Starting the Cosminexus JMS Provider services (for restarting a running system)	<i>Appendix E. 7</i>	
			Checking the state of the CJMSP resource adapters and CJMSP Broker	<i>Appendix E. 8</i>	
			Checking the message delivery status and the action to be taken for accumulated messages (Procedure for pausing CJMSP Broker)	<i>Appendix E. 9</i>	
			Checking the message delivery status and the action to be taken for accumulated messages (Procedure for stopping an application)	<i>Appendix E. 10</i>	
			Terminating the Cosminexus JMS Provider services	<i>Appendix E. 11</i>	
	Non-periodic operations	Executes operations that are required non-periodically for using Cosminexus JMS Provider	Compressing the destinations	<i>Appendix E. 12</i>	
			Changing the destination size	<i>Appendix E. 13</i>	
			Deleting the persistence subscribers	<i>Appendix E. 14</i>	
	Monitoring task	Monitors the resources used with Cosminexus JMS Provider.	Monitoring the CJMSP Broker status	<i>Appendix E. 15</i>	
			Checking the CJMSP Broker details	<i>Appendix E. 16</i>	
			Checking the destination status	<i>Appendix E. 17</i>	
			Checking the persistence subscriber status	<i>Appendix E. 18</i>	
	Errors	Error analysis procedure	Analyzes the cause of error when an error occurs while Cosminexus JMS Provider is being used.	Analysis of errors in the CJMSP resource adapters	<i>Appendix E. 19</i>

Category	Task	Task contents	Task details	Reference location
Errors	Error analysis procedure	Analyzes the cause of error when an error occurs while Cosminexus JMS Provider is being used.	Analysis when CJMSP Broker stops due to an error	<i>Appendix E. 20</i>
			Analysis when there is no response from a Cosminexus JMS Provider service	<i>Appendix E. 21</i>
	Recovery procedure	Recovers the system when an error occurs while Cosminexus JMS Provider is being used.	Recovery when an error occurs in a CJMSP resource adapter	<i>Appendix E. 22</i>
			Recovery when CJMSP Broker stops due to an error	<i>Appendix E. 23</i>
			Recovery when there is no response from a Cosminexus JMS Provider service	<i>Appendix E. 24</i>
Deletion	Deleting the environment	Deletes the Cosminexus JMS Provider environment.	Deleting the Cosminexus JMS Provider service instances	<i>Appendix E. 25</i>

E.1 Preconditions common to all the use cases

The preconditions common to all the use cases are as follows:

- To perform a forced execution without checking the user with the Cosminexus JMS Provider-related command operations, specify `-force` option for the `cjmsbroker` command, and the `-f` option for the `cjmsicmd` command.
- Execute the Cosminexus JMS Provider-related command operations on a computer on which CJMSP Broker is running.
- To use Cosminexus JMS Provider, the user must have Administrator permissions.
- Do not execute multiple Cosminexus JMS Provider-related command operations concurrently.
- To display the help for the command-line tool (`cjmsbroker` and `cjmsicmd`) of Cosminexus JMS Provider, specify the `-h` option or `-help` option.
- Set the J2EE server start option so that the security manager is not used. If use of security manager is specified, an invalid permission error occurs when a CJMSP resource adapter accesses a resource.
- To specify a provider URL in the command operations used in a J2EE server, follow the below examples:

Format of execution

```
cjlistrar server-name -nameserver provider-URL
```

Example of execution

```
cjlistrar MyServer -nameserver corbaname::localhost:900
```

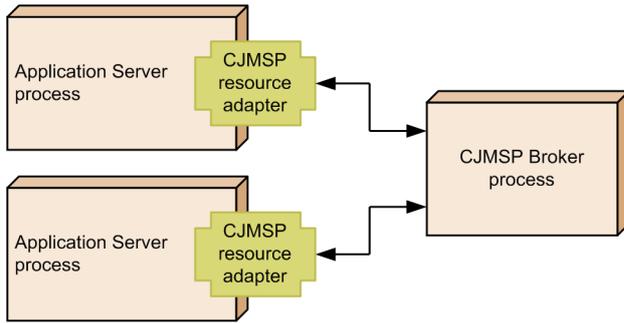
In *provider-URL*, specify *protocol-name::host-name:port-number*.

- For the command details coded in the description, see the *uCosminexus Application Server Command Reference Guide*.

E.2 Prerequisite process model

The following figure shows the process model assumed in the use cases.

Figure E-1: Process model assumed in the use cases

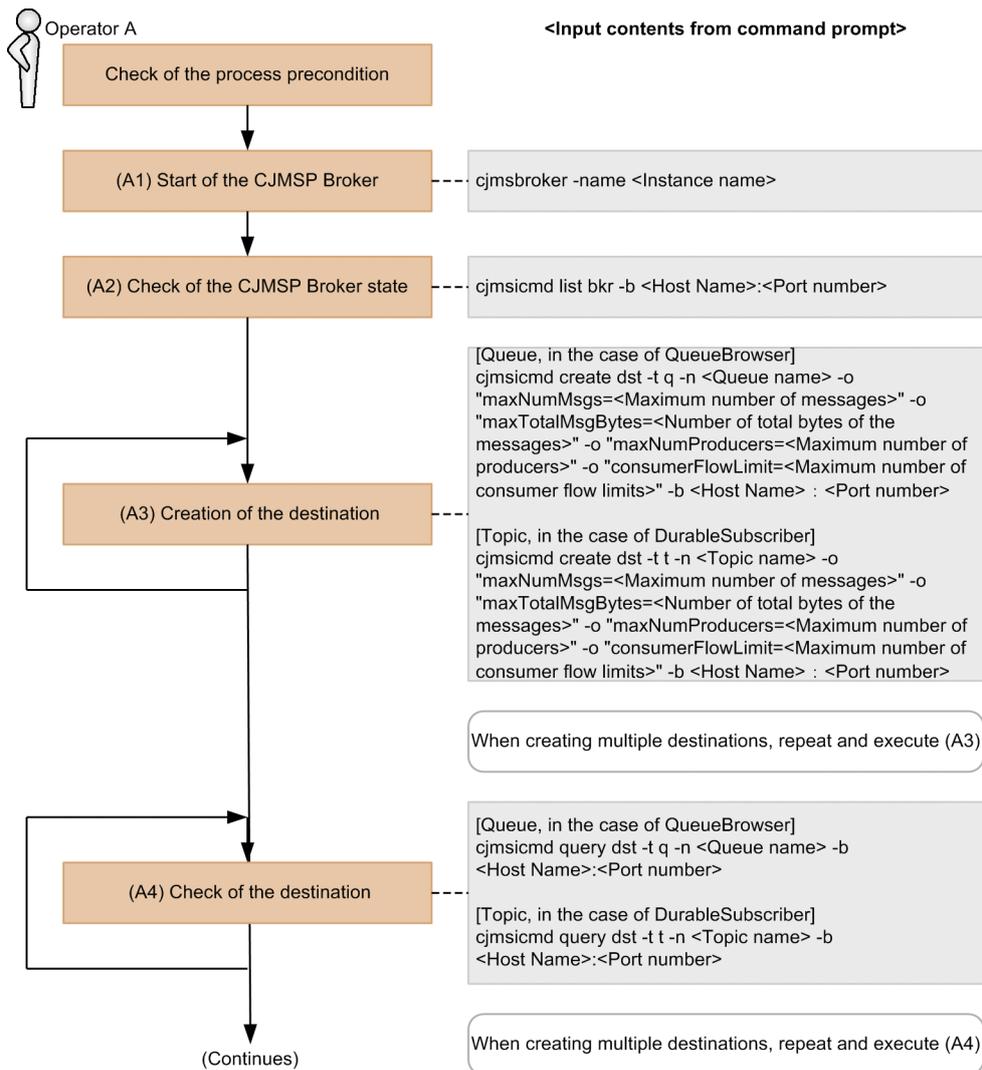


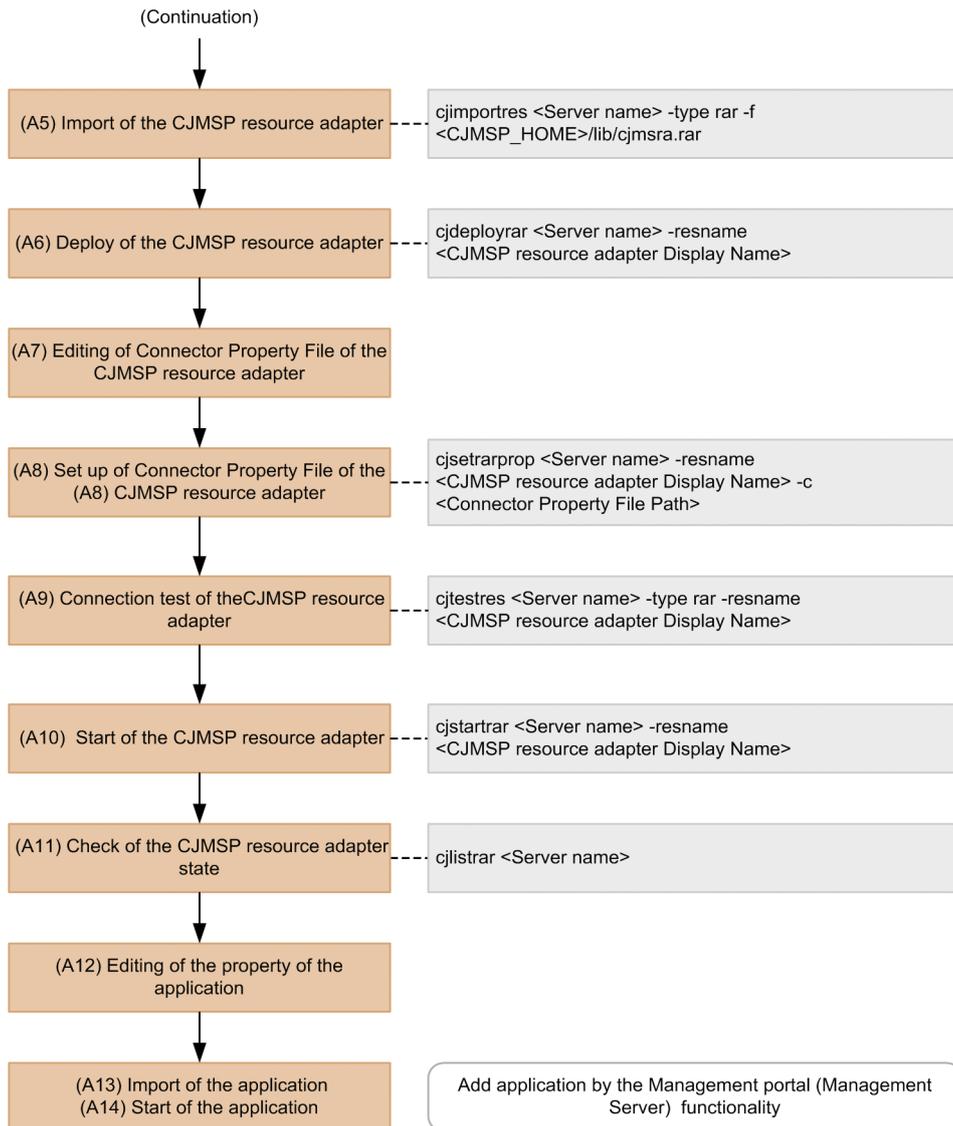
- One CJMSP Broker can be accessed from multiple J2EE servers.
- One CJMSP resource adapter can be defined in one J2EE server.

E.3 Environment setup for using Cosminexus JMS Provider

The following figure shows the environment setup procedure for using Cosminexus JMS Provider.

Figure E-2: Environment setup procedure for using Cosminexus JMS Provider





(1) Preconditions

- Cosminexus installation and J2EE server setup has been completed using management portal (Management Server).
- After the completion of Application Server installation, Cosminexus JMS Provider is installed in the following directory:
`<COSMINEXUS_HOME>/CC/cjmsp`
 Before you execute this procedure, make sure the directory exists.

(2) Process preconditions

- The J2EE server process is already running.
- The CJMSP resource adapter is not running (not imported).
- The CJMSP Broker process is not running.
- The application is not running (not imported).

(3) Information and supplementary explanation required for execution

The coding, such as (A1), corresponds to the coding in the figure.

(A1)

instance-name

If omitted, `cjmsbroker` is used by default.

If you want to use multiple instances, make sure that the names do not overlap. To overlap the names, specify the `-name` option, and assign any non-existent name.

(A2)

host-name:port-number

The host name and port number on which CJMSP Broker is running.

If you omit the port number when CJMSP Broker is started, the default port number 7676 is used. In this case, you can omit the `-b` option.

Make sure that the status of CJMSP Broker is `OPERATING`.

! Important note

If you start the CJMSP resource adapter when CJMSP Broker is not running normally, an exception occurs and Cosminexus JMS Provider cannot be used.

(A3)

Queue-name, Topic-name

Queue name or Topic name

maximum-number-of-messages

Maximum number of messages that can be stored in a Queue or Topic

total-number-of-message-bytes

Maximum total message bytes of a Queue or Topic

maximum-number-of-producers

Maximum number of producers for a destination

maximum-consumer-flow-limit

Maximum number of messages that one processing unit can deliver to a consumer

host-name:port-number

The host name and port number on which CJMSP Broker is running.

If you omit the port number when CJMSP Broker is started, the default port number 7676 is used. In this case, you can omit the `-b` option.

(A4)

Queue-name, Topic-name

Queue name or Topic name

host-name:port-number

The host name and port number on which CJMSP Broker is running.

If you omit the port number when CJMSP Broker is started, the default port number 7676 is used. In this case, you can omit the `-b` option.

Make sure that the destination created in (A3) has been created correctly using the specified property. The mapping between the arguments specified in (A3) and the displayed contents is as follows:

- `maxNumMsgs` = Max Number of Messages
- `maxNumProducers` = Max Number of Producers
- `maxTotalMsgBytes` = Max Total Message Bytes
- `consumerFlowLimit` = Consumer Flow Limit

An example display of a property is as follows. This is an example of execution when Queue is created.

```

Destination Name TestQueue
Destination Type Queue
Destination State RUNNING
Created Administratively true

Current Number of Messages
  Actual 0
  Held in Transaction 0
Current Message Bytes
  Actual 0
  Held in Transaction 0
Current Number of Producers 0
Current Number of Active Consumers 0

Max Number of Messages unlimited (-1)
Max Total Message Bytes unlimited (-1)
Max Number of Producers unlimited (-1)

Consumer Flow Limit 100

KDAN34151-I Successfully queried the destination.

```

(A5)

```

server-name
  Server name set up with Management Server
<CJMSP_HOME>
  Cosminexus-installation-directory/CC/cjmsp

```

(A6)

```

server-name
  Server name set up with Management Server
CJMSP-resource-adapter-display-name
  Display name of the CJMSP resource adapter
  In Cosminexus JMS Provider, Cosminexus_JMS_Provider_RA is set up by default.

```

(A7)

Copy the template file and then edit the HITACHI Connector Property file. Use <CJMSP_HOME>/lib/templates/Cosminexus_JMS_Provider_RA_cfg.xml as the template file.

To use a destination, define the Administered object provided by the CJMSP resource adapter.

An example of setup is as follows. Specify the tags to which numbers are allocated (the numbers are not included in the actual definition).

```

<config-property>
<description xml:lang="en"></description>
<config-property-name>ConnectionURL</config-property-name>
<config-property-type>java.lang.String</config-property-type>
(1) <config-property-value>mq://localhost:7676</config-property-value>
</config-property>

<config-property>
<description xml:lang="en"></description>
<config-property-name>clientId</config-property-name>
<config-property-type>java.lang.String</config-property-type>
(2) <config-property-value>Test</config-property-value>
</config-property>

<adminobject>
(3) <adminobject-name>myQueue</adminobject-name>
(4) <adminobject-interface>javax.jms.Queue</adminobject-interface>
(5) <adminobject-class>com.cosminexus.jmsprovider.messaging.Queue</
adminobject-class>
<config-property>
<description xml:lang="en"></description>
<config-property-name>Name</config-property-name>
<config-property-type>java.lang.String</config-property-type>
(6) <config-property-value>TestQueue</config-property-value>
</config-property>
<config-property>
<description xml:lang="en"></description>
<config-property-name>Description</config-property-name>

```

```

<config-property-type>java.lang.String</config-property-type>
<config-property-value></config-property-value>
</config-property>
</adminobject>

```

The content specified for each number is as follows:

1. Specify the host name and port number that starts CJMSP Broker.
2. Specify the client identifier. The client identifier is required for using the persistence subscriber.
3. Specify the Administered object name in <adminobject-name>. The Administered object name must be unique in the CJMSP resource adapter.
4. Specify the interface implemented by the Administered object class in <adminobject-interface>.

When you use Queue, specify `javax.jms.Queue`.

When you use Topic, specify `javax.jms.Topic`.
5. Specify the Administered object class in <adminobject-class>.

When you use Queue, specify `com.cosminexus.jmsprovider.messaging.Queue`.

When you use Topic, specify `com.cosminexus.jmsprovider.messaging.Topic`.
6. Set the name of the destination specified in (A3).
Create the <adminobject> tag for each destination to be used.

! Important note

When you use the Message-driven Beans, set up the following two points:

- The value of (a) shown below must be equal to or more than the value of (b).
- The value of total number of (a) of all the deployed Message-driven Beans must be equal to or less than the value of (c) shown below:

Maximum number of pooled Message-driven Bean instances in the HITACHI Application Property file

```

<pooled-instance>
<minimum>1</minimum>
(a) <maximum>2</maximum>
</pooled-instance>

```

Maximum number of pooled Endpoint instances in the HITACHI Application Property file

```

<activation-config-property>
<activation-config-property-name>endpointPoolMaxSize</activation-config-
property-name>
(b) <activation-config-property-value>1</activation-config-property-value>
</activation-config-property>

```

Maximum number of pooled WorkManager threads in the HITACHI Application Property file for the CJMSP resource adapter

```

<property>
<property-name>MaxTPoolSize</property-name>
<property-type>int</property-type>
(c) <property-value>10</property-value>
<property-default-value>10</property-default-value>
</property>

```

(A8)

server-name

Server name set up with Management Server

CJMSP-resource-adapter-display-name

Display name of the CJMSP resource adapter

In Cosminexus JMS Provider, `Cosminexus_JMS_Provider_RA` is set up by default.

HITACHI-Connector-Property-file-path

Input source path of the property file

(A9)

server-name

Server name set up with Management Server

CJMSP-resource-adapter-display-name

Display name of the CJMSP resource adapter

In Cosminexus JMS Provider, *Cosminexus_JMS_Provider_RA* is set up by default.

Check the following two points with the connection test for the CJMSP resource adapters:

- The operation mode is the 1.4 mode.
- A connection can be established with CJMSP Broker.

(A10)

server-name

Server name set up with Management Server

CJMSP-resource-adapter-display-name

Display name of the CJMSP resource adapter

In Cosminexus JMS Provider, *Cosminexus_JMS_Provider_RA* is set up by default.

(A11)

server-name

Server name set up with Management Server

Make sure that the name is displayed as "*running CJMSP-resource-adapter-display-name*".

(A12)

Specify settings to associate the CJMSP resource adapter connection factory and Administered object in the application property.

An example of setup is as follows. Edit the tags to which numbers are allocated (the numbers are not included in the actual definition).

```

<resource-ref>
(1) <res-ref-name>jms/qcf</res-ref-name>
(2) <res-type>javax.jms.QueueConnectionFactory</res-type>
(3) <res-auth>Container</res-auth>
(4) <res-sharing-scope>Unshareable</res-sharing-scope>
(5) <linked-to>Cosminexus_JMS_Provider_RA!javax.jms.QueueConnectionFactory</
linked-to>
</resource-ref>
<resource-env-ref>
(6) <resource-env-ref-name>jms/TestQueue</resource-env-ref-name>
(7) <resource-env-ref-type>javax.jms.Queue</resource-env-ref-type>
<linked-adminobject>
(8) <resourceadapter-name>Cosminexus_JMS_Provider_RA</resourceadapter-name>
(9) <adminobject-name>myQueue</adminobject-name>
</linked-adminobject>
</resource-env-ref>

```

The content specified for each number is as follows:

1. In `<res-ref-name>`, specify the resource reference name.
2. In `<res-type>`, specify the resource reference type. Specify the connection factory type.
3. In `<res-auth>`, specify whether the authentication for using a resource will be performed on an application or will be entrusted to a container. The specifiable strings are as follows:
Application
Container
4. In `<res-sharing-scope>`, specify whether the resource connections will be shared. The specifiable strings are as follows:
Shareable
Unshareable
5. In `<linked-to>`, specify the corresponding CJMSP resource adapter display name.
CJMSP-resource-adapter-display-name!Connection-definition-identifier

6. In <resource-env-ref-name>, specify the name of the resource environment variable reference.
7. In <resource-env-ref-type>, specify the interface type of the destination as the Administered object type.
8. In <resourceadapter-name>, specify the display name of the CJMSP resource adapter.
9. In <adminobject-name>, specify the Administered object name set at (3) in (A7).

(A13)

No specific value.

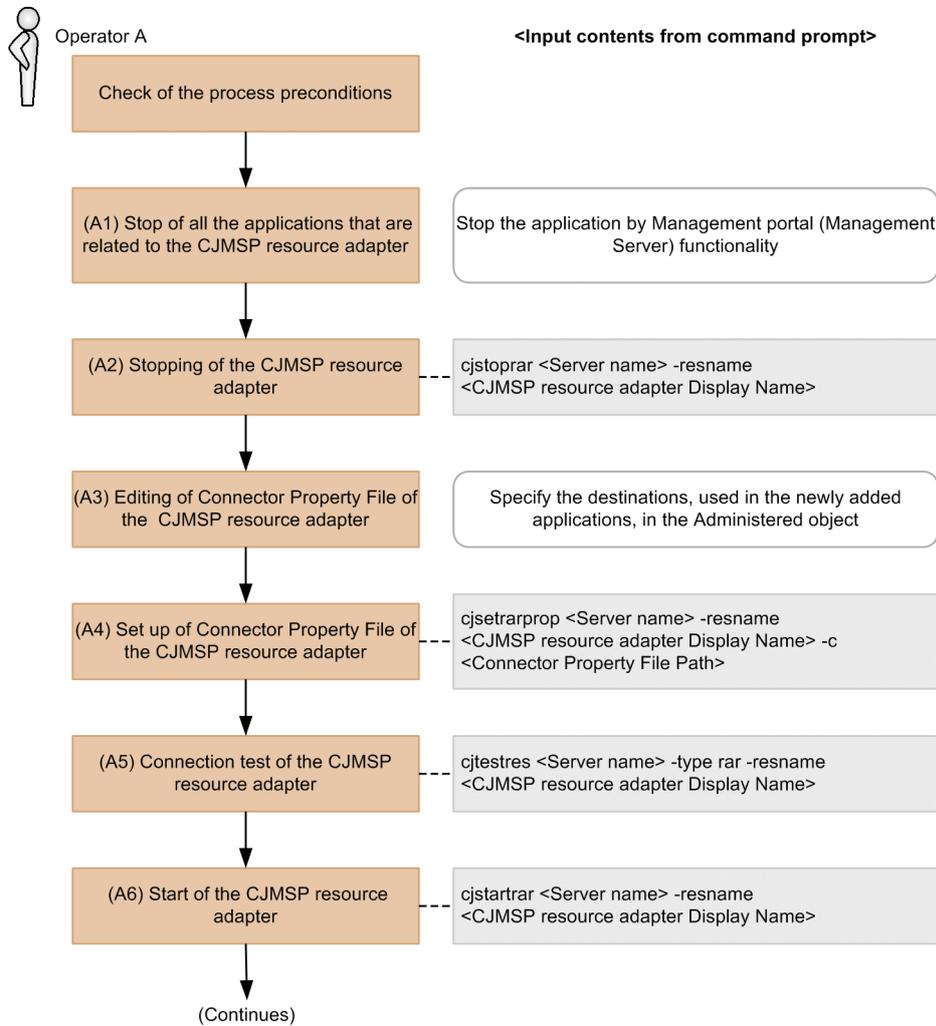
(A14)

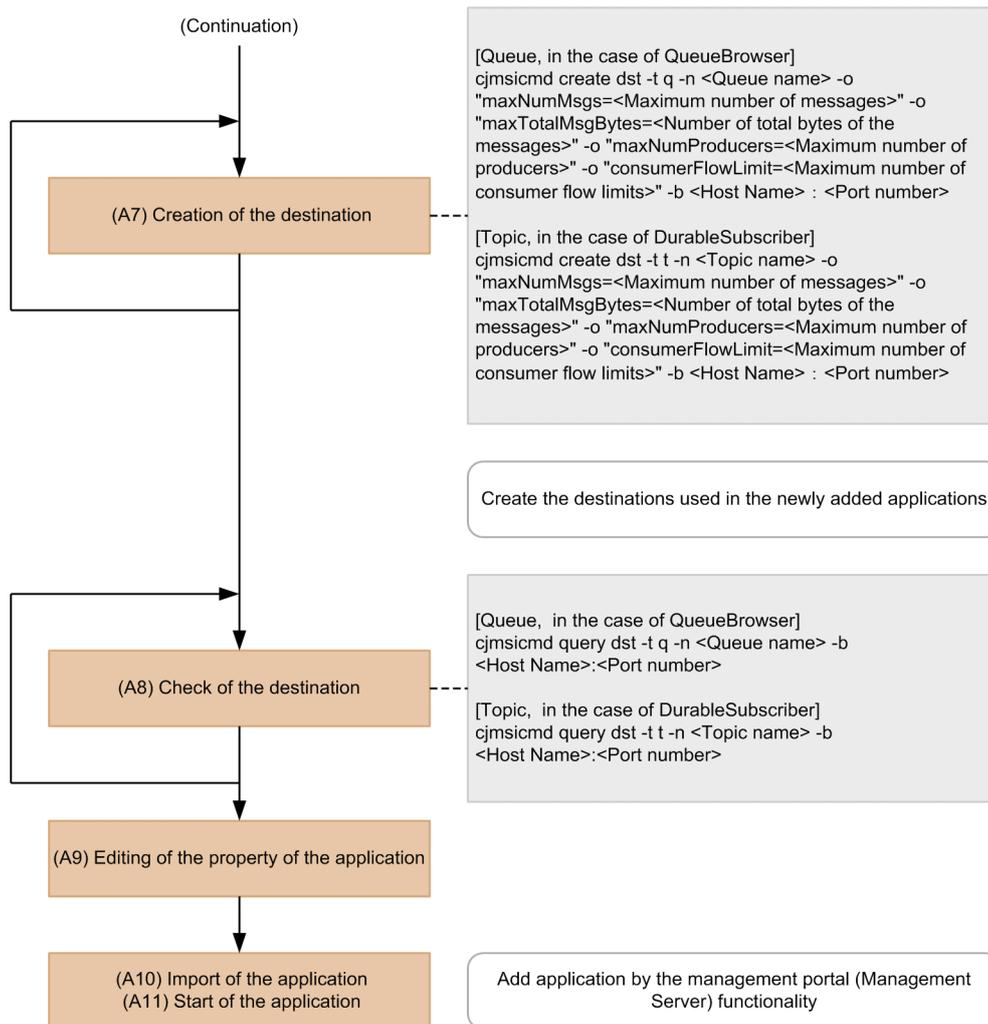
No specific value.

E.4 Adding applications that use Cosminexus JMS Provider

The following figure shows the procedure of adding new applications for a running system.

Figure E-3: Procedure of adding new applications for a running system





(1) Process preconditions

- The J2EE server process is running.
- The CJMSP resource adapter is running.
- The CJMSP Broker process is running.
- The application is not running (not imported).

(2) Information and supplementary explanation required for execution

The coding, such as (A1), corresponds to the coding in the figure.

(A1)

Stop the application with the management portal (Management Server) functionality.

(A2)

server-name

Server name set up with Management Server

CJMSP-resource-adapter-display-name

Display name of the CJMSP resource adapter

In Cosminexus JMS Provider, `Cosminexus_JMS_Provider_RA` is set up by default.

(A3)

Copy the template file and then edit the HITACHI Connector Property file.

Use `<CJMSP_HOME>/lib/templates/Cosminexus_JMS_Provider_RA_cfg.xml` as the template file.

To use a destination, define the Administered object provided by the CJMSP resource adapter.

An example of setup is as follows. Specify the tags to which numbers are allocated (the numbers are not included in the actual definition).

```
<config-property>
<description xml:lang="en"></description>
<config-property-name>ConnectionURL</config-property-name>
<config-property-type>java.lang.String</config-property-type>
(1) <config-property-value>mq://localhost:7676</config-property-value>
</config-property>
<config-property>
<description xml:lang="en"></description>
<config-property-name>clientId</config-property-name>
<config-property-type>java.lang.String</config-property-type>
(2) <config-property-value>Test</config-property-value>
</config-property>

<adminobject>
(3) <adminobject-name>myQueue</adminobject-name>
(4) <adminobject-interface>javax.jms.Queue</adminobject-interface>
(5) <adminobject-class>com.cosminexus.jmsprovider.messaging.Queue</
adminobject-class>
<config-property>
<description xml:lang="en"></description>
<config-property-name>Name</config-property-name>
<config-property-type>java.lang.String</config-property-type>
(6) <config-property-value>TestQueue</config-property-value>
</config-property>
<config-property>
<description xml:lang="en"></description>
<config-property-name>Description</config-property-name>
<config-property-type>java.lang.String</config-property-type>
<config-property-value></config-property-value>
</config-property>
</adminobject>
```

The content specified for each number is as follows:

- Specify the host name and port number that starts CJMSP Broker.
- Specify the client identifier. The client identifier is required for using the persistence subscriber.
- Specify the Administered object name in `<adminobject-name>`. The Administered object name must be unique in the CJMSP resource adapter.
- Specify the interface implemented by the Administered object class in `<adminobject-interface>`.
When you use Queue, specify `javax.jms.Queue`.
When you use Topic, specify `javax.jms.Topic`.
- Specify the Administered object class in `<adminobject-class>`.
When you use Queue, specify `com.cosminexus.jmsprovider.messaging.Queue`.
When you use Topic, specify `com.cosminexus.jmsprovider.messaging.Topic`.
- Set the name of the destination.
Create the `<adminobject>` tag for each destination to be used.

Important note

When you use the Message-driven Beans, set up the following two points:

- In the following points, the value of (a) must be equal to or more than the value of (b).
- The value of total number of (a) of all the deployed Message-driven Beans must be equal to or less than the value of (c) shown below:

Maximum number of pooled Message-driven Bean instances in the HITACHI Application Property file

```
<pooled-instance>
<minimum>1</minimum>
(a) <maximum>2</maximum>
```

```
</pooled-instance>
```

Maximum number of pooled Endpoint instances in the HITACHI Application Property file

```
<activation-config-property>
```

```
<activation-config-property-name>endpointPoolMaxSize</activation-config-property-name>
```

```
(b) <activation-config-property-value>1</activation-config-property-value>
```

```
</activation-config-property>
```

Maximum number of pooled WorkManager threads in the HITACHI Application Property file for the CJMSP resource adapter

```
<property>
```

```
<property-name>MaxTPoolSize</property-name>
```

```
<property-type>int</property-type>
```

```
(c) <property-value>10</property-value>
```

```
<property-default-value>10</property-default-value>
```

```
</property>
```

(A4)

server-name

Server name set up with Management Server

CJMSP-resource-adapter-display-name

Display name of the CJMSP resource adapter

In Cosminexus JMS Provider, `Cosminexus_JMS_Provider_RA` is set up by default.

HITACHI-Connector-Property-file-path

Input source path of the property file

(A5)

server-name

Server name set up with Management Server

CJMSP-resource-adapter-display-name

Display name of the CJMSP resource adapter

In Cosminexus JMS Provider, `Cosminexus_JMS_Provider_RA` is set up by default.

Check the following two points with the connection test for the CJMSP resource adapters:

- The operation mode is the 1.4 mode.
- A connection can be established with CJMSP Broker.

(A6)

server-name

Server name set up with Management Server

CJMSP-resource-adapter-display-name

Display name of the CJMSP resource adapter

In Cosminexus JMS Provider, `Cosminexus_JMS_Provider_RA` is set up by default.

(A7)

Queue-name, Topic-name

Queue name or Topic name

maximum-number-of-messages

Maximum number of messages that can be stored in a Queue or Topic

total-number-of-message-bytes

Maximum total message bytes of a Queue or Topic

maximum-number-of-producers

Maximum number of producers for a destination

maximum-consumer-flow-limit

Maximum number of messages that one processing unit can deliver to a consumer

host-name:port-number

The host name and port number on which CJMSP Broker is running.

If you omit the port number when CJMSP Broker is started, the default port number 7676 is used. In this case, you can omit the `-b` option.

In the *Queue-name* or *Topic-name*, enter the destination name set at (6) in (A3).

(A8)

Queue-name, Topic-name

Queue name or Topic name

host-name:port-number

The host name and port number on which CJMSP Broker is running.

If you omit the port number when CJMSP Broker is started, the default port number 7676 is used. In this case, you can omit the `-b` option.

Make sure that the destination created in (A7) has been created correctly using the specified property. The mapping between the arguments specified in (A7) and the displayed contents is as follows:

- `maxNumMsgs` ? Max Number of Messages
- `maxNumProducers` ? Max Number of Producers
- `maxTotalMsgBytes` ? Max Total Message Bytes
- `consumerFlowLimit` ? Consumer Flow Limit

The specified property is output with the following display. This is an example of execution when Queue is created.

```

Destination Name TestQueue
Destination Type Queue
Destination State RUNNING
Created Administratively true

Current Number of Messages
  Actual 0
  Held in Transaction 0
Current Message Bytes
  Actual 0
  Held in Transaction 0
Current Number of Producers 0
Current Number of Active Consumers 0

Max Number of Messages unlimited (-1)
Max Total Message Bytes unlimited (-1)
Max Number of Producers unlimited (-1)

Consumer Flow Limit 100

KDAN34151-I Successfully queried the destination.

```

(A9)

Specify settings to associate the CJMSP resource adapter connection factory and Administered object in the application property.

An example of setup is as follows. Specify the tags to which numbers are allocated (the numbers are not included in the actual definition).

```

<resource-ref>
(1) <res-ref-name>jms/qcf</res-ref-name>
(2) <res-type>javax.jms.QueueConnectionFactory</res-type>
(3) <res-auth>Container</res-auth>
(4) <res-sharing-scope>Unshareable</res-sharing-scope>
(5) <linked-to>Cosminexus_JMS_Provider_RA!javax.jms.QueueConnectionFactory</
linked-to>
</resource-ref>
<resource-env-ref>
(6) <resource-env-ref-name>jms/TestQueue</resource-env-ref-name>
(7) <resource-env-ref-type>javax.jms.Queue</resource-env-ref-type>
<linked-adminobject>
(8) <resourceadapter-name>Cosminexus_JMS_Provider_RA</resourceadapter-name>
(9) <adminobject-name>myQueue</adminobject-name>
</linked-adminobject>
</resource-env-ref>

```

The content specified for each number is as follows:

1. In `<res-ref-name>`, specify the resource reference name.
2. In `<res-type>`, specify the resource reference type. Specify the connection factory type.
3. In `<res-auth>`, specify whether the authentication for using a resource will be performed on an application or will be entrusted to a container.
The specifiable strings are as follows:
Application
Container
4. In `<res-sharing-scope>`, specify whether the resource connections will be shared.
The specifiable strings are as follows:
Shareable
Unshareable
5. In `<linked-to>`, specify the corresponding CJMSP resource adapter display name.
Specify the following string:
CJMSP-resource-adapter-display-name!Connection-definition-identifier
6. In `<resource-env-ref-name>`, specify the name of the resource environment variable reference.
7. In `<resource-env-ref-type>`, specify the Administered object type. Specify the types of interfaces of the destination.
8. In `<resourceadapter-name>`, specify the display name of the CJMSP resource adapter.
9. In `<adminobject-name>`, specify the Administered object name set at (3) in (A3).

(A10)

No specific value.

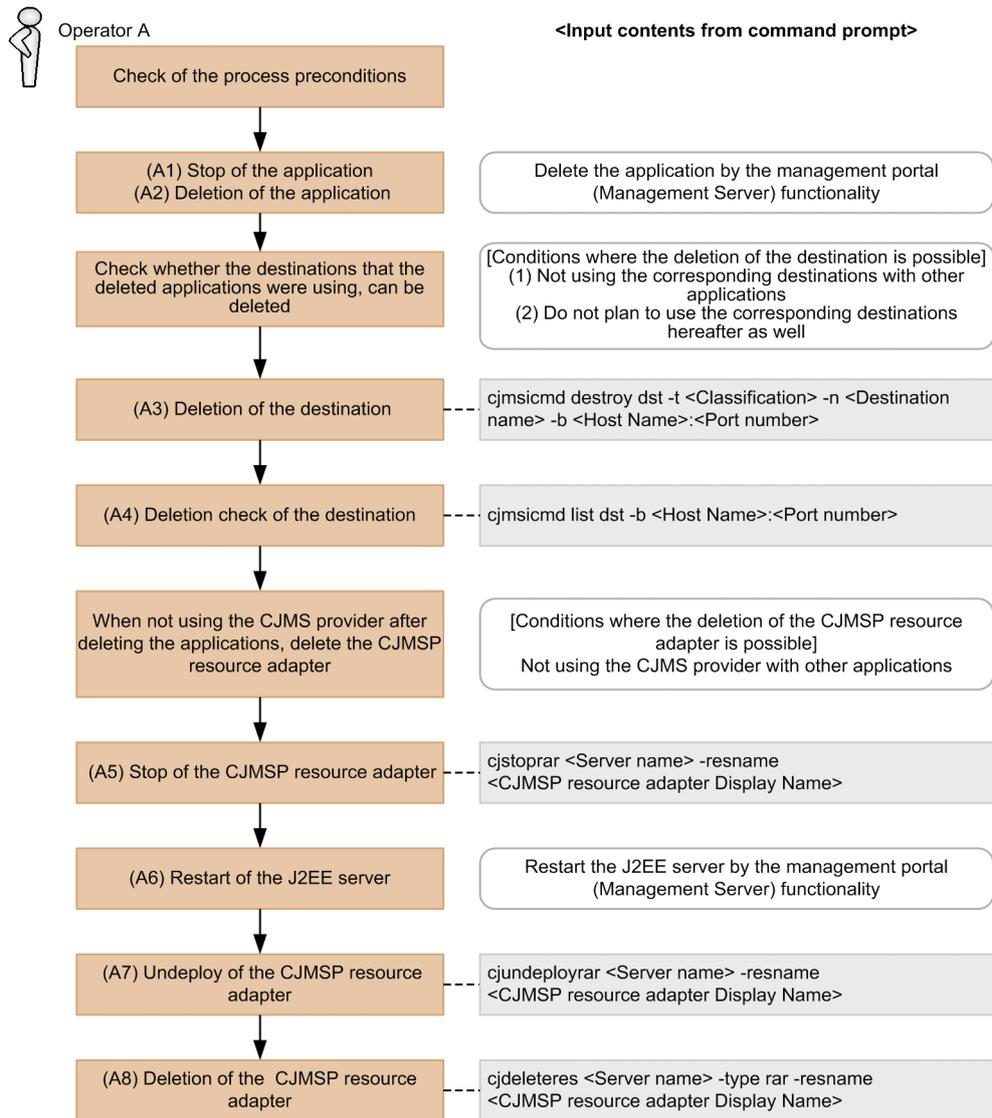
(A11)

No specific value.

E.5 Deleting applications that use Cosminexus JMS Provider

The following figure shows the procedure for deleting applications from a running system.

Figure E-4: Procedure for deleting applications from a running system



(1) Process preconditions

- The J2EE server process is running.
- The CJMSP resource adapter is running.
- The CJMSP Broker process is running.
- The application is running.

(2) Information and supplementary explanation required for execution

The coding, such as (A1), corresponds to the coding in the figure.

(A1)

No specific value.

(A2)

Delete the application using the management portal (Management Server) functionality.

(A3)

Type

q (Queue) or t (Topic)

Destination-name

Queue name or Topic name

host-name:port-number

The host name and port number on which CJMSP Broker is running.

If you omit the port number when CJMSP Broker is started, the default port number 7676 is used. In this case, you can omit the `-b` option.

(A4)

host-name:port-number

The host name and port number on which CJMSP Broker is running.

If you omit the port number when CJMSP Broker is started, the default port number 7676 is used. In this case, you can omit the `-b` option.

Make sure that the destination deleted in (A3) has been deleted correctly.

(A5)

server-name

Server name set up with Management Server

CJMSP-resource-adapter-display-name

Display name of the CJMSP resource adapter

In Cosminexus JMS Provider, `Cosminexus_JMS_Provider_RA` is set up by default.

(A6)

No specific value.

(A7)

server-name

Server name set up with Management Server

CJMSP-resource-adapter-display-name

CJMSP resource adapter display name

In Cosminexus JMS Provider, `Cosminexus_JMS_Provider_RA` is set up by default.

(A8)

server-name

Server name set up with Management Server

CJMSP-resource-adapter-display-name

Display name of the CJMSP resource adapter

In Cosminexus JMS Provider, `Cosminexus_JMS_Provider_RA` is set up by default.

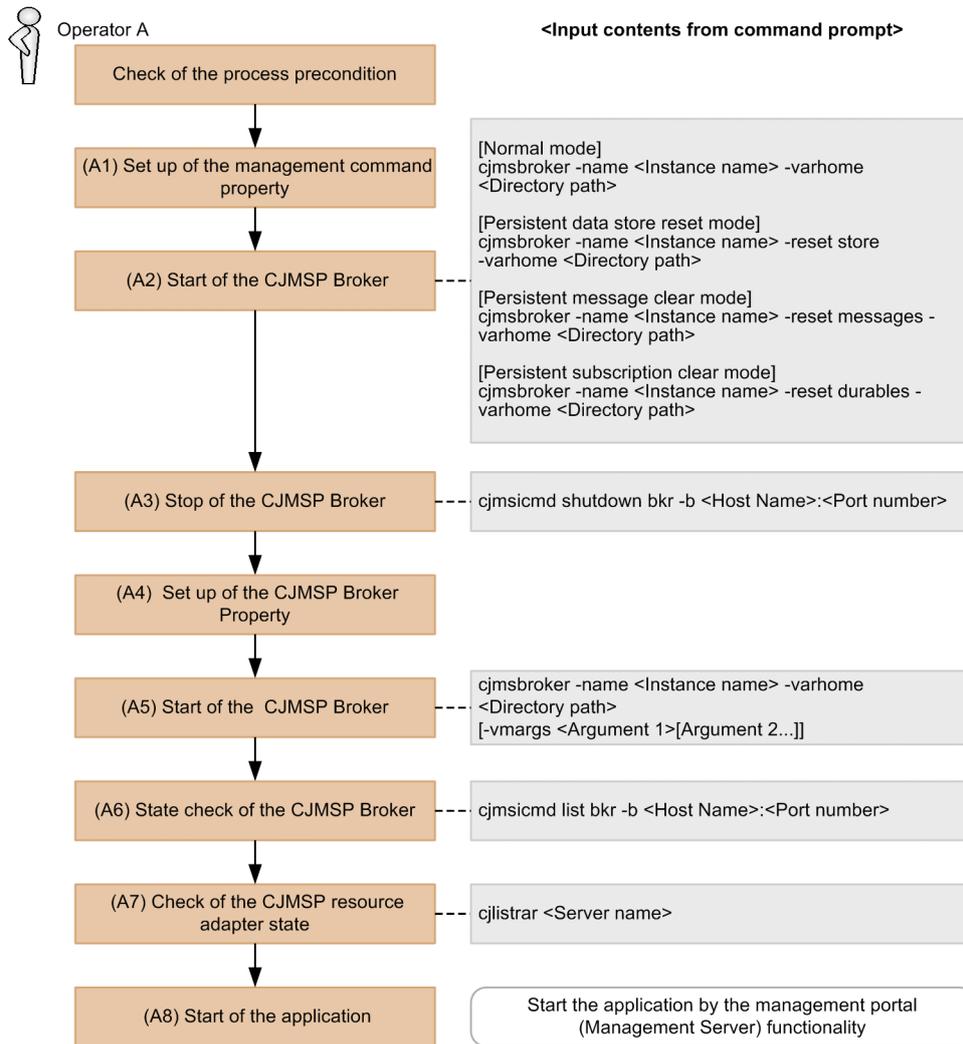
E.6 Starting the Cosminexus JMS Provider services (for the initial startup)

This subsection describes the procedure for starting a Cosminexus JMS Provider service during initial startup in the cases such as immediately after system setup. A service that is started using this procedure can be terminated with *Appendix E.11 Terminating the Cosminexus JMS Provider services*.

If you start the CJMSP resource adapter and application according to this procedure, both CJMSP resource adapter and application start when you restart the J2EE server (however, as a precondition, CJMSP Broker must be started first).

The following figure shows the procedure for starting the Cosminexus JMS Provider services (for the initial startup).

Figure E-5: Procedure for starting the Cosminexus JMS Provider services (for the initial startup)



(1) Process preconditions

- The J2EE server process is running.
- The CJMSP resource adapter is not running.
- The CJMSP Broker process is not running.
- The application is not running.

(2) Information and supplementary explanation required for execution

The coding, such as (A1), corresponds to the coding in the figure.

(A1)

The management command property file is stored in <CJMSP_HOME>/var/admin/config/admin.properties.

The specifiable property names and description are as follows. For details on the file, see 7.2 admin.properties (Management command property file) in the uCosminexus Application Server Definition Reference Guide.

```
admin.logger.MessageLogFile.trace.level
```

Specifies the message log level of a management command.

`admin.logger.MessageLogFile.filepath`

Specifies the file path to output the management command message log.

`admin.logger.MessageLogFile.filenum`

Specifies the maximum number of message log files created with a management command.

`admin.logger.MessageLogFile.filesize`

Specifies the maximum size of the message log file of a management command.

`admin.logger.ExceptionLogFile.filepath`

Specifies the file path to output the management command exception log.

`admin.logger.ExceptionLogFile.filenum`

Specifies the maximum number of exception log files created with a management command.

`admin.logger.ExceptionLogFile.filesize`

Specifies the maximum size of the exception log file of a management command.

(A2)

instance-name

If omitted, `cjmsbroker` is used by default.

If you want to use multiple instances, make sure that the names do not overlap.

To overlap the names, specify the `-name` option, and assign any non-existent name.

Directory-path

Specify the path where you want to output `VAR_HOME`.

Differences in options based on the modes

- Normal mode
No option is specified (starts the Cosminexus JMS Provider services continuously without resetting the status).
- Persistent data store reset mode
The persistence messages and persistence subscribers are deleted.
- Persistent message clear mode
All the persistence messages are deleted.
- Persistent subscription clear mode
All the persistence subscribers are deleted.

(A3)

host-name:port-number

Specify the host name and port number on which CJMSP Broker is running.

If you omit the port number when CJMSP Broker is started, the default port number 7676 is used. In this case, you can omit the `-b` option.

(A4)

The CJMSP Broker Property File is stored in `<VAR_HOME>/instances/instance-name/props/config.properties`.

This file is created when CJMSP Broker is started for the first time. Therefore, the (A2) and (A3) steps must be performed.

The specifiable property names and description are as follows. For details on the file, see *7.4 config.properties (CJMSP Broker Property File)* in the *uCosminexus Application Server Definition Reference Guide*.

If you create a temporary destination, the value of `imq.autocreate.queue.consumerFlowLimit` or `imq.autocreate.topic.consumerFlowLimit` is set as the destination property.

`imq.hostname`

Specifies the default host name or IP address of all the connection services.

`imq.portmapper.port`

Specifies the port number for the port mapper of CJMSP Broker.

`imq.jms.tcp.port`

Specifies the port number of the `jms` service.

`imq.admin.tcp.port`

Specifies the port number of the admin service.

`imq.persist.file.sync.enabled`

Specifies the flag for synchronous or asynchronous disk write operations.

`imq.autocreate.queue.consumerFlowLimit`

Specifies the maximum number of messages that one processing unit can deliver to a consumer of a Queue.

`imq.autocreate.topic.consumerFlowLimit`

Specifies the maximum number of messages that one processing unit can deliver to a consumer of a Topic.

`imq.metrics.interval`

Specifies the time interval in seconds for the output of the metrics information to the log and console.

`broker.logger.MessageLogFile.trace.level`

Specifies the message log level of CJMSP Broker.

`broker.logger.MessageLogFile.filenum`

Specifies the maximum number of message log files created with CJMSP Broker.

`broker.logger.MessageLogFile.filesize`

Specifies the maximum size of the message log file of CJMSP Broker.

`broker.logger.ExceptionLogFile.filenum`

Specifies the maximum number of exception log files created with CJMSP Broker.

`broker.logger.ExceptionLogFile.filesize`

Specifies the maximum size of the exception log file of CJMSP Broker.

`imq.instanceconfig.version`

Specifies the config property version.

(A5)

instance-name

If omitted, `cjmsbroker` is used by default.

If you want to use multiple instances, make sure that the names do not overlap.

To overlap the names, specify the `-name` option, and assign any non-existent name.

Directory-path

Specify the path where you want to output `VAR_HOME`.

Argument

Specify the JVM options such as `-Xms` (maximum heap size), `-Xmx` (minimum heap size), and `-XX:+HitachiVerboseGC` (option to output the extended verbosegc information for garbage collection).

(A6)

Make sure that the State is `OPERATING`.

host-name:port-number

The host name and port number on which CJMSP Broker is running.

If you omit the port number when CJMSP Broker is started, the default port number 7676 is used. In this case, you can omit the `-b` option.

(A7)

server-name

Server name set up with Management Server

(A8)

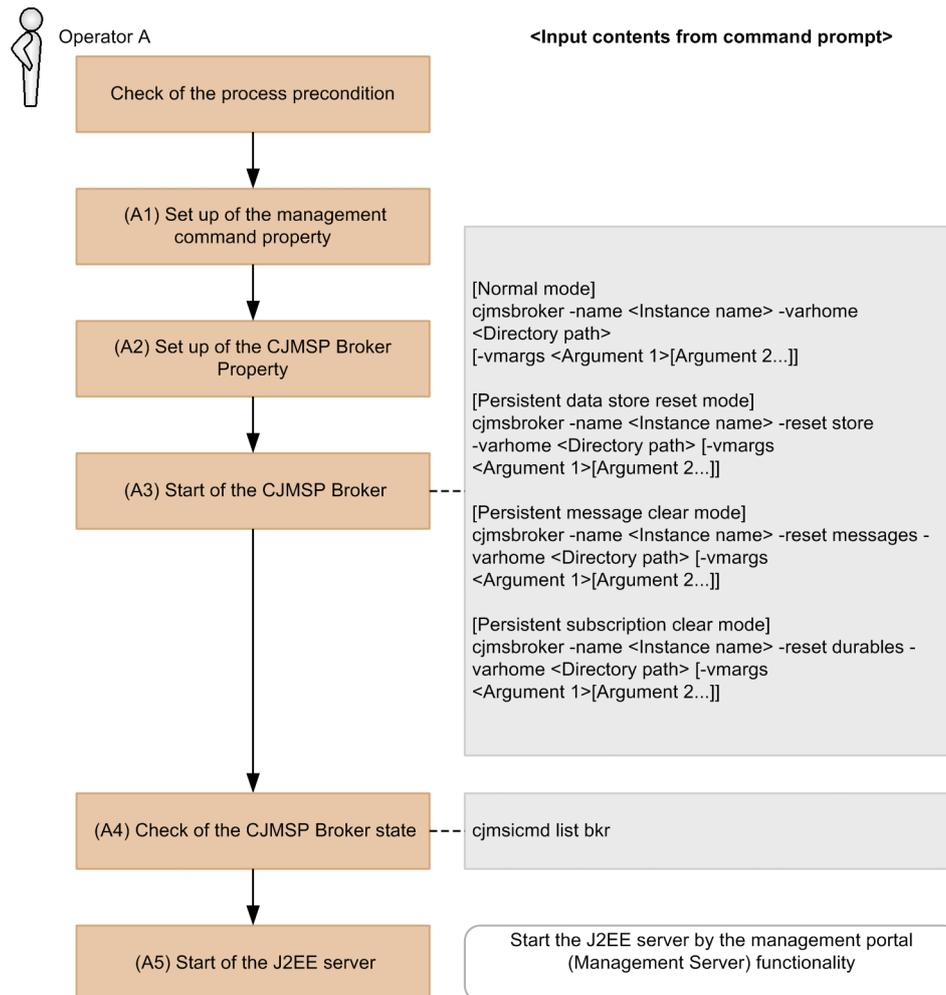
No specific value.

E.7 Starting the Cosminexus JMS Provider services (for restarting a running system)

This subsection describes the procedure for starting a Cosminexus JMS Provider service when you restart a running system. A service that is started using this procedure can be terminated with *Appendix E.11 Terminating the Cosminexus JMS Provider services*.

The following figure shows the procedure for starting the Cosminexus JMS Provider services (for restarting a running system).

Figure E-6: Procedure for starting the Cosminexus JMS Provider services (for restarting a running system)



(1) Process preconditions

- The J2EE server process is not running.
- The CJMSP resource adapter is not running.
- The CJMSP Broker process is not running.
- The application is not running.

(2) Information and supplementary explanation required for execution

The coding, such as (A1), corresponds to the coding in the figure.

(A1)

The management command property file is stored in `<CJMSP_HOME>/var/admin/config/admin.properties`.

The specifiable property names and description are as follows. For details on the file, see *7.2 admin.properties (Management command property file)* in the *uCosminexus Application Server Definition Reference Guide*.

`admin.logger.MessageLogFile.trace.level`

Specifies the message log level of a management command.

`admin.logger.MessageLogFile.filepath`

Specifies the file path to output the management command message log.

`admin.logger.MessageLogFile.filenum`

Specifies the maximum number of message log files created with a management command.

`admin.logger.MessageLogFile.filesize`

Specifies the maximum size of the message log file of a management command.

`admin.logger.ExceptionLogFile.filepath`

Specifies the file path to output the management command exception log.

`admin.logger.ExceptionLogFile.filenum`

Specifies the maximum number of exception log files created with a management command.

`admin.logger.ExceptionLogFile.filesize`

Specifies the maximum size of the exception log file of a management command.

(A2)

The CJMSP Broker Property File is stored in `<VAR_HOME>/instances/instance-name/props/config.properties`.

The specifiable property names and description are as follows. For details on the file, see *7.4 config.properties (CJMSP Broker Property File)* in the *uCosminexus Application Server Definition Reference Guide*.

If you create a temporary destination, the value of `imq.autocreate.queue.consumerFlowLimit` or `imq.autocreate.topic.consumerFlowLimit` is set as the destination property.

`imq.hostname`

Specifies the default host name or IP address of all the connection services.

`imq.portmapper.port`

Specifies the port number for the port mapper of CJMSP Broker.

`imq.jms.tcp.port`

Specifies the port number of the jms service.

`imq.admin.tcp.port`

Specifies the port number of the admin service.

`imq.persist.file.sync.enabled`

Specifies the flag for synchronous or asynchronous disk write operations.

`imq.autocreate.queue.consumerFlowLimit`

Specifies the maximum number of messages that one processing unit can deliver to a consumer of a Queue.

`imq.autocreate.topic.consumerFlowLimit`

Specifies the maximum number of messages that one processing unit can deliver to a consumer of a Topic.

`imq.metrics.interval`

Specifies the time interval in seconds for the output of the metrics information to the log and console.

`broker.logger.MessageLogFile.trace.level`

Specifies the message log level of CJMSP Broker.

`broker.logger.MessageLogFile.filenum`

Specifies the maximum number of message log files created with CJMSP Broker.

`broker.logger.MessageLogFile.filesize`

Specifies the maximum size of the message log file of CJMSP Broker.

`broker.logger.ExceptionLogFile.filenum`

Specifies the maximum number of exception log files created with CJMSP Broker.

`broker.logger.ExceptionLogFile.filesize`

Specifies the maximum size of the exception log file of CJMSP Broker.

`img.instanceconfig.version`

Specifies the config property version.

(A3)

instance-name

If omitted, `cjmsbroker` is used by default.

If you want to use multiple instances, make sure that the names do not overlap.

To overlap the names, specify the `-name` option, and assign any non-existent name.

Directory-path

Specify the path where you want to output `VAR_HOME`.

Argument

Specify the JavaVM options such as `-Xms` (maximum heap size), `-Xmx` (minimum heap size), and `-XX:+HitachiVerboseGC` (option to output the extended verbosegc information for garbage collection).

Differences in options based on the modes

- Normal mode
No option is specified (starts the Cosminexus JMS Provider services continuously without resetting the status).
- Persistent data store reset mode
The persistence messages and persistence subscribers are deleted.
- Persistent message clear mode
All the persistence messages are deleted.
- Persistent subscription clear mode
All the persistence subscribers are deleted.

(A4)

Make sure that the State is `OPERATING`.

(A5)

Start the J2EE server using the management portal (Management Server) functionality.

The CJMSP resource adapter and application start when you restart the J2EE server.

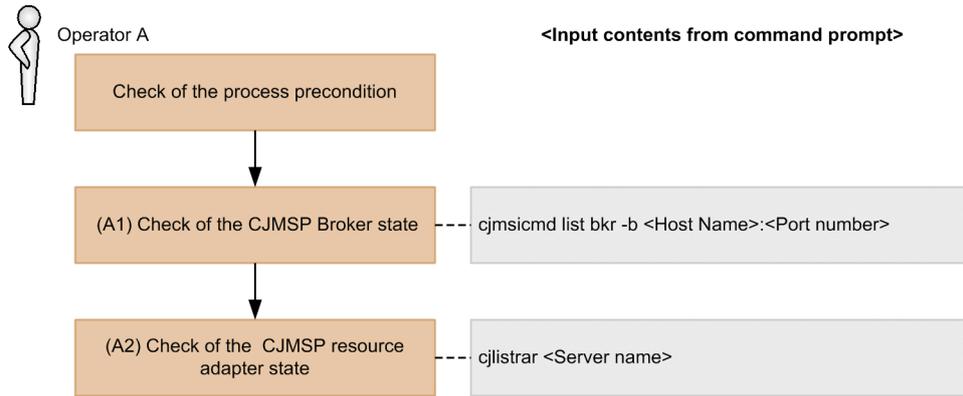
E.8 Checking the state of the CJMSP resource adapters and CJMSP Broker

This subsection describes the procedure for checking whether a Cosminexus JMS Provider service is available.

You check the states of the J2EE servers and applications with the management portal (Management Server) functionality.

The following figure shows the procedure for checking the state of the CJMSP resource adapters and CJMSP Broker.

Figure E-7: Procedure for checking the state of the CJMSP resource adapters and CJMSP Broker



(1) Process preconditions

- The J2EE server process is running.
- The CJMSP resource adapter is running.
- The CJMSP Broker process is running.
- The application is running.

(2) Information and supplementary explanation required for execution

The coding, such as (A1), corresponds to the coding in the figure.

(A1)

host-name:port-number

The host name and port number on which CJMSP Broker is running.

If you omit the port number when CJMSP Broker is started, the default port number 7676 is used. In this case, you can omit the `-b` option.

(A2)

server-name

Server name set up with Management Server

CJMSP service start status

When CJMSP Broker is `OPERATING` and the CJMSP resource adapter is `running`, the state of the Cosminexus JMS Provider service is `running`.

An example result of (A1) is as follows:

```

-----
Address State
-----
localhost:7676 OPERATING
    
```

An example result of (A2) is as follows:

```

-----
running Cosminexus_JMS_Provider_RA
-----
    
```

E.9 Checking the message delivery status and the action to be taken for accumulated messages (Procedure for pausing CJMSP Broker)

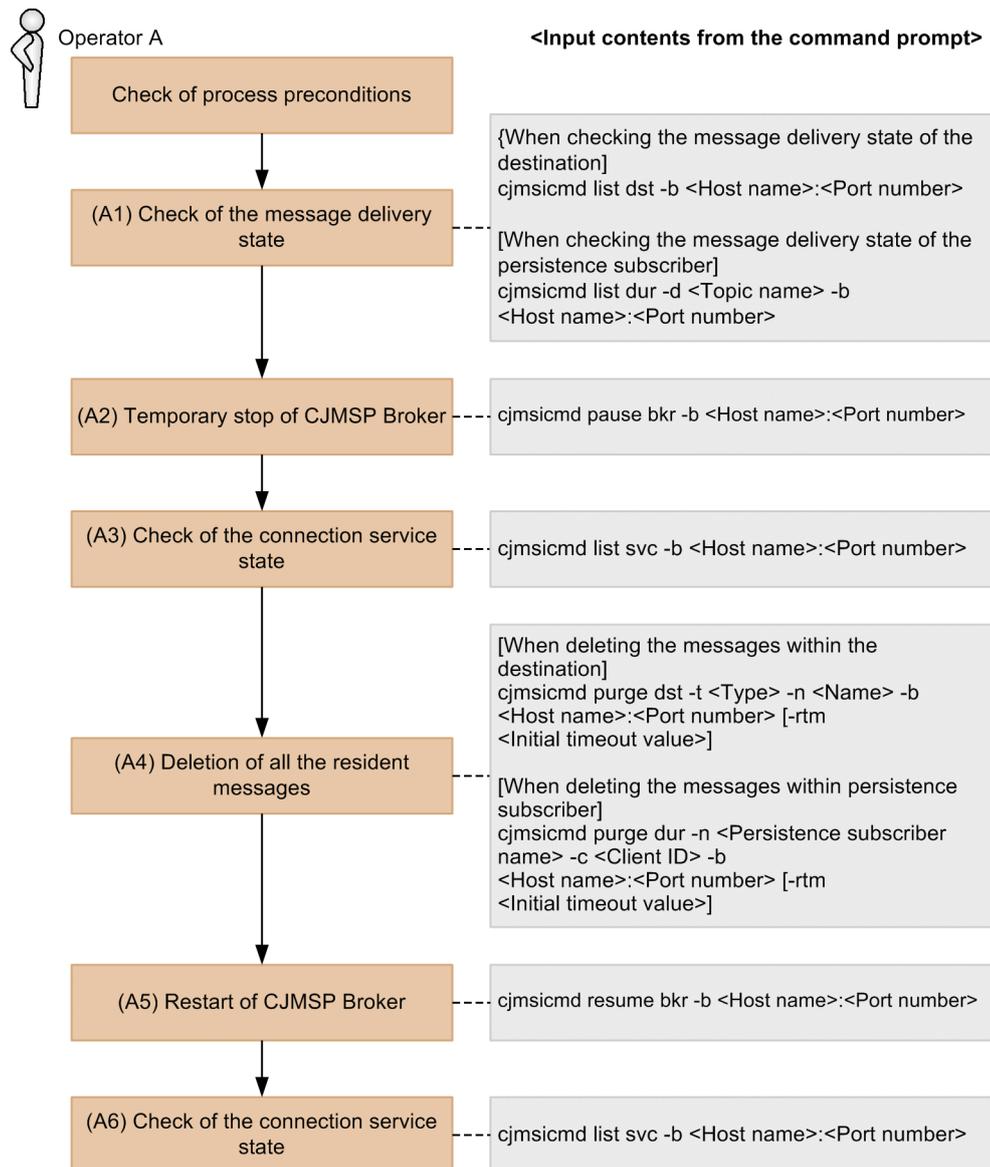
This subsection describes the procedure for checking the message delivery status of Cosminexus JMS Provider and the procedure of action when the messages accumulate.

With this method, an exception might occur due to the occurrence of application timeout after CJMSP Broker is paused.

To delete the messages safely, see *Appendix E.10 Checking the message delivery status and the action to be taken for accumulated messages (Procedure for stopping an application)*.

The following figure shows the procedure for checking the message delivery status and the action to be taken for accumulated messages (procedure for pausing CJMSP Broker).

Figure E-8: Procedure for checking the message delivery status and the action to be taken for accumulated messages (procedure for pausing CJMSP Broker)



(1) Process preconditions

- The J2EE server process is running.
- The CJMSP resource adapter is running.
- The CJMSP Broker process is running.
- The application is running.

(2) Information and supplementary explanation required for execution

The coding, such as (A1), corresponds to the coding in the figure.

(A1)

Check the destination (or persistence subscriber) name and the number of messages.

host-name:port-number

The host name and port number on which CJMSP Broker is running.

If you omit the port number when CJMSP Broker is started, the default port number 7676 is used. In this case, you can omit the `-b` option.

Topic-name

Topic name containing the persistence subscriber you want to check

(A2)

host-name:port-number

The host name and port number on which CJMSP Broker is running.

If you omit the port number when CJMSP Broker is started, the default port number 7676 is used. In this case, you can omit the `-b` option.

(A3)

Check whether the status of the jms service is PAUSED.

host-name:port-number

The host name and port number on which CJMSP Broker is running.

If you omit the port number when CJMSP Broker is started, the default port number 7676 is used. In this case, you can omit the `-b` option.

An example of output is as follows:

```
-----
Service Name Port Number Service State
-----
admin 1248 (dynamic) RUNNING
jms 0 (dynamic) PAUSED
KDAN34113-I Successfully listed services.
```

(A4)

Type

q (Queue) or t (Topic)

Name

Queue name or Topic name

host-name:port-number

The host name and port number on which CJMSP Broker is running.

If you omit the port number when CJMSP Broker is started, the default port number 7676 is used. In this case, you can omit the `-b` option.

Initial-timeout-value

The default initial timeout value is set at 10 seconds. Adjust the value if the processing times out with the default value.

Persistence-subscriber-name

Persistence subscriber name checked in (A1)

Client-ID

Client ID checked in (A1)

(A5)

host-name:port-number

The host name and port number on which CJMSP Broker is running.

If you omit the port number when CJMSP Broker is started, the default port number 7676 is used. In this case, you can omit the `-b` option.

(A6)

Check whether the status of the jms service is RUNNING.

host-name:port-number

The host name and port number on which CJMSP Broker is running.

If you omit the port number when CJMSP Broker is started, the default port number 7676 is used. In this case, you can omit the -b option.

An example of output is as follows:

```
-----
Service Name Port Number Service State
-----
admin 1248 (dynamic) RUNNING
jms 1337 (dynamic) RUNNING

KDAN34113-I Successfully listed services.
-----
```

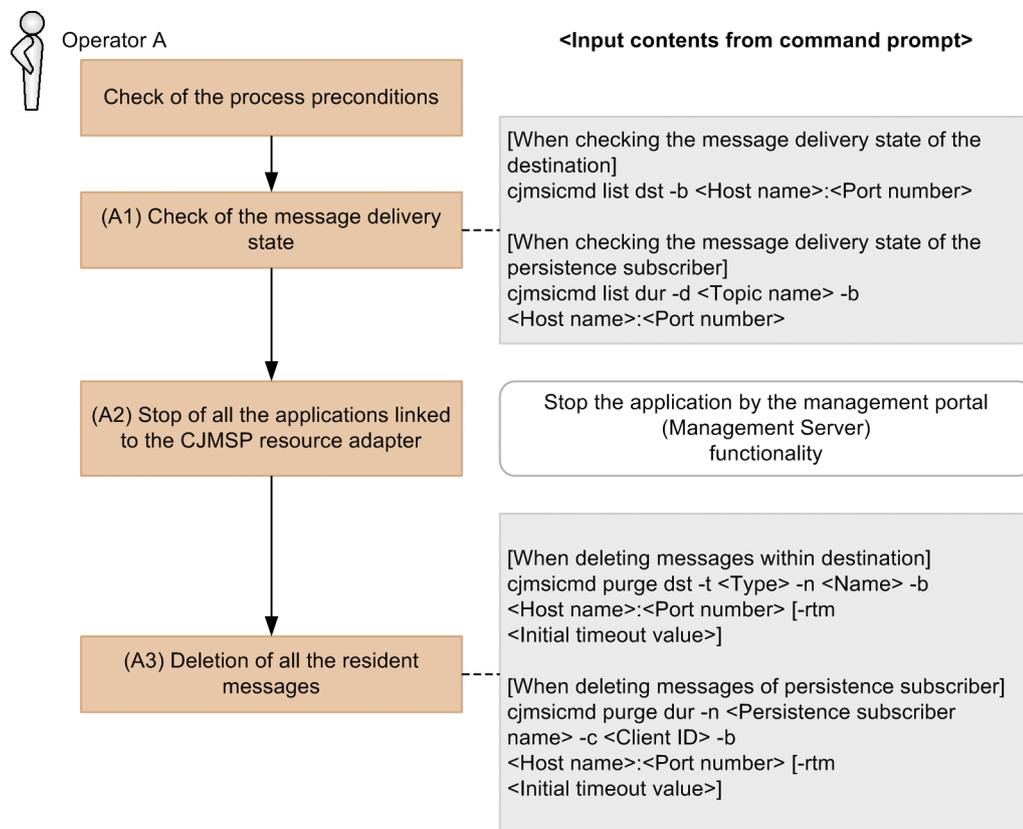
E.10 Checking the message delivery status and the action to be taken for accumulated messages (Procedure for stopping an application)

This subsection describes the procedure for checking the message delivery status of Cosminexus JMS Provider and the procedure of action when the messages accumulate.

If you do not want to stop the application, see *Appendix E.9 Checking the message delivery status and the action to be taken for accumulated messages (Procedure for pausing CJMSP Broker)*.

The following figure shows the procedure for checking the message delivery status and the action to be taken for accumulated messages (procedure for stopping an application).

Figure E-9: Procedure for checking the message delivery status and the action to be taken for accumulated messages (procedure for stopping an application)



(1) Process preconditions

- The J2EE server process is running.
- The CJMSP resource adapter is running.
- The CJMSP Broker process is running.
- The application is running.

(2) Information and supplementary explanation required for execution

The coding, such as (A1), corresponds to the coding in the figure.

(A1)

Check the destination (or persistence subscriber) name and the number of messages.

host-name:port-number

The host name and port number on which CJMSP Broker is running.

If you omit the port number when CJMSP Broker is started, the default port number 7676 is used. In this case, you can omit the `-b` option.

Topic-name

Topic name containing the persistence subscriber you want to check

(A2)

No specific value.

(A3)

Type

q (Queue) or t (Topic)

Name

Queue name or Topic name

host-name:port-number

The host name and port number on which CJMSP Broker is running.

If you omit the port number when CJMSP Broker is started, the default port number 7676 is used. In this case, you can omit the `-b` option.

Initial-timeout-value

The default initial timeout value is set at 10 seconds. Adjust the value if the processing times out with the default value.

Persistence-subscriber-name

Persistence subscriber name checked in (A1)

Client-ID

Client ID checked in (A1)

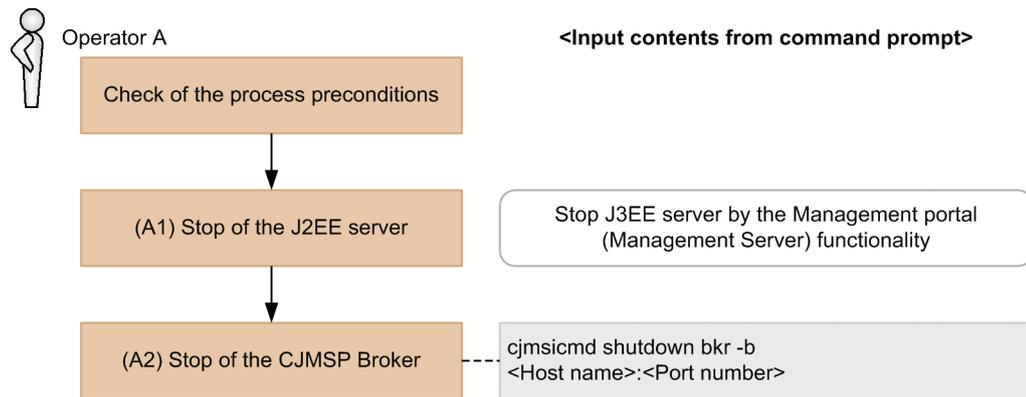
E.11 Terminating the Cosminexus JMS Provider services

This subsection describes the procedure for terminating the Cosminexus JMS Provider service.

A service that is stopped using this procedure can be restarted with the procedure described in *Appendix E.7 Starting the Cosminexus JMS Provider services (for restarting a running system)*.

The following figure shows the procedure for terminating the Cosminexus JMS Provider services.

Figure E-10: Procedure for terminating the Cosminexus JMS Provider services



(1) Process preconditions

- The J2EE server process is running.
- The CJMSP resource adapter is running.
- The CJMSP Broker process is running.
- The application is running.

(2) Information and supplementary explanation required for execution

The coding, such as (A1), corresponds to the coding in the figure.

(A1)

No specific value.

(A2)

host-name:port-number

The host name and port number on which CJMSP Broker is running.

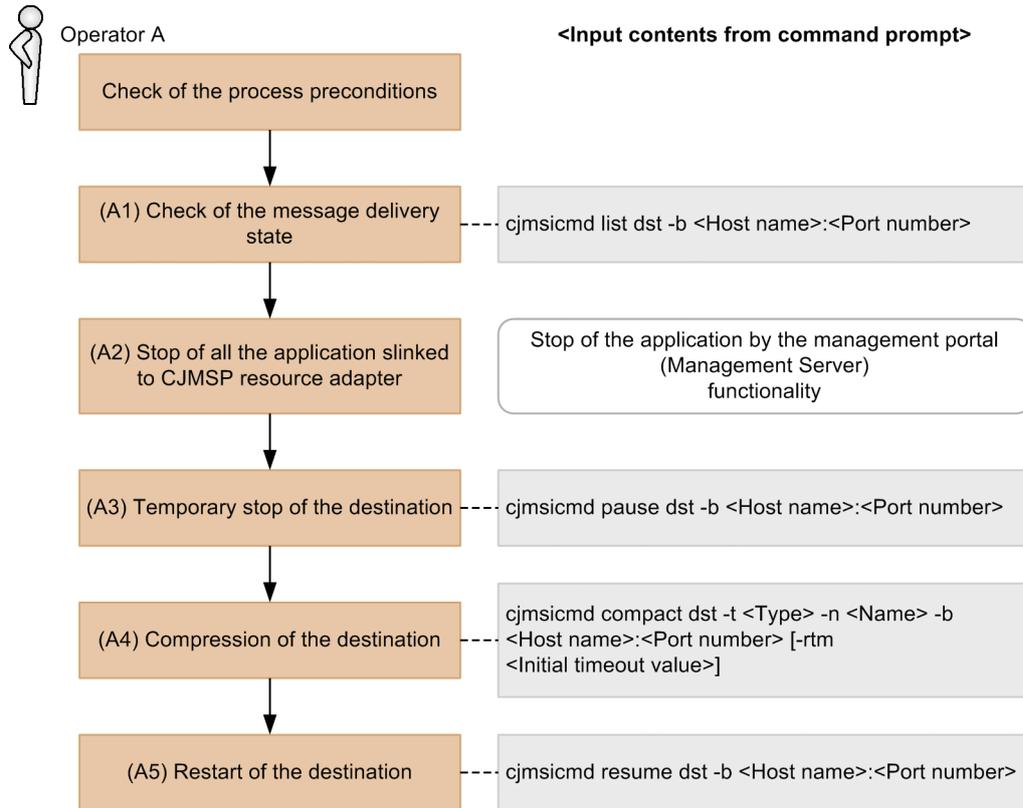
If you omit the port number when CJMSP Broker is started, the default port number 7676 is used. In this case, you can omit the `-b` option.

E.12 Compressing the destinations

This subsection describes the procedure for compressing the destinations to be used with Cosminexus JMS Provider.

The following figure shows the procedure for compressing the destinations.

Figure E - 11: Procedure for compressing the destinations



(1) Process preconditions

- The J2EE server process is running.
- The CJMSP resource adapter is running.
- The CJMSP Broker process is running.
- The application is running.

(2) Information and supplementary explanation required for execution

The coding, such as (A1), corresponds to the coding in the figure.

(A1)

Check the destination name and the number of messages.

host-name:port-number

The host name and port number on which CJMSP Broker is running.

If you omit the port number when CJMSP Broker is started, the default port number 7676 is used. In this case, you can omit the -b option.

(A2)

No specific value.

(A3)

host-name:port-number

The host name and port number on which CJMSP Broker is running.

If you omit the port number when CJMSP Broker is started, the default port number 7676 is used. In this case, you can omit the -b option.

(A4)

Type

q (Queue) or t (Topic)

Name

Queue name or Topic name

host-name:port-number

The host name and port number on which CJMSP Broker is running.

If you omit the port number when CJMSP Broker is started, the default port number 7676 is used. In this case, you can omit the -b option.

Initial-timeout-value

The default initial timeout value is set at 10 seconds. Adjust the value if the processing times out with the default value.

(A5)

host-name:port-number

The host name and port number on which CJMSP Broker is running.

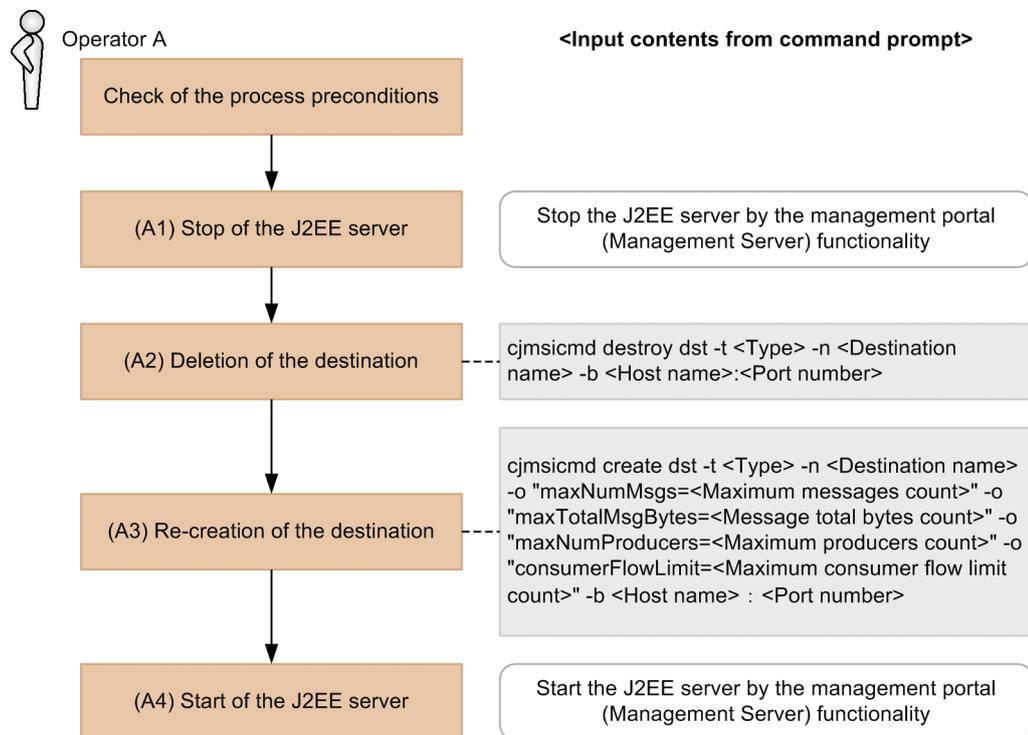
If you omit the port number when CJMSP Broker is started, the default port number 7676 is used. In this case, you can omit the -b option.

E.13 Changing the destination size

This subsection describes the procedure for changing the size of the destinations to be used with Cosminexus JMS Provider.

The following figure shows the procedure for changing the destination size.

Figure E-12: Procedure for changing the destination size



(1) Process preconditions

- The J2EE server process is running.

- The CJMSP resource adapter is running.
- The CJMSP Broker process is running.
- The application is running.

(2) Information and supplementary explanation required for execution

The coding, such as (A1), corresponds to the coding in the figure.

(A1)

No specific value.

(A2)

Type

q (Queue) or t (Topic)

Destination-name

Queue name or Topic name to be deleted

host-name:port-number

The host name and port number on which CJMSP Broker is running.

If you omit the port number when CJMSP Broker is started, the default port number 7676 is used. In this case, you can omit the `-b` option.

(A3)

Type

q (Queue) or t (Topic)

Destination-name

Queue name or Topic name to be re-created

maximum-number-of-messages

Maximum number of messages stored in Queue or Topic

total-number-of-message-bytes

Total number of message bytes stored in Queue or Topic

maximum-number-of-producers

Maximum number of producers for a destination

maximum-consumer-flow-limit

Maximum number of messages that one processing unit can deliver to a consumer

host-name:port-number

The host name and port number on which CJMSP Broker is running.

If you omit the port number when CJMSP Broker is started, the default port number 7676 is used. In this case, you can omit the `-b` option.

An example of the execution result is as follows:

Example: Example of creating a Queue where the maximum number of messages is 100000, total number of message bytes is 12 MB, maximum number of producers is 1000, and the maximum consumer flow limit is 1000

```
C:\>cjmsicmd create dst -t q -n Queue1 -o "maxNumMsgs=100000" -o
"maxTotalMsgBytes=12m" -o "maxNumProducers=1000" -o "consumerFlowLimit=1000" -b
localhost:7676
KDAN34140-I Successfully created the destination.
```

(A4)

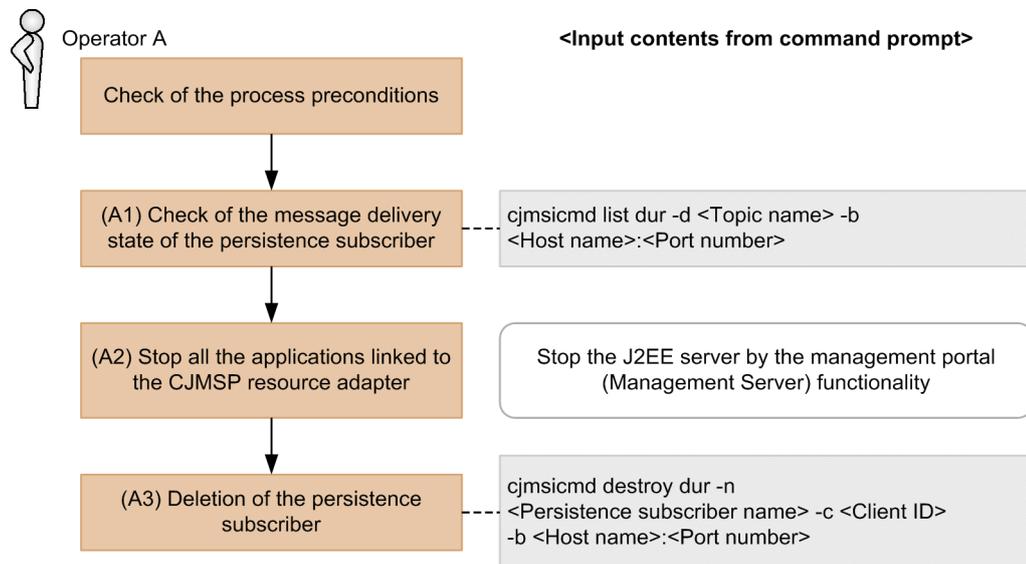
No specific value.

E.14 Deleting the persistence subscribers

This subsection describes the procedure for deleting the persistence subscribers used with Cosminexus JMS Provider.

The following figure shows the procedure for deleting the persistence subscribers.

Figure E-13: Procedure for deleting the persistence subscribers



(1) Process preconditions

- The J2EE server process is running.
- The CJMSP resource adapter is running.
- The CJMSP Broker process is running.
- The application is running.

(2) Information and supplementary explanation required for execution

The coding, such as (A1), corresponds to the coding in the figure.

(A1)

Topic-name

Topic name containing the persistence subscriber you want to check

host-name:port-number

The host name and port number on which CJMSP Broker is running.

If you omit the port number when CJMSP Broker is started, the default port number 7676 is used. In this case, you can omit the `-b` option.

An example of display is as follows:

```

-----
Durable Sub. Name Client ID Number of Messages Durable Sub. State
-----
DurableSUB1 clientId 9998 INACTIVE
KDAN34323-I Successfully listed durable subscriptions.
-----
  
```

(A2)

No specific value.

(A3)

Persistence-subscriber-name

Persistence subscriber name checked in (A1)

Client-ID

Client ID checked in (A1)

host-name:port-number

The host name and port number on which CJMSP Broker is running.

If you omit the port number when CJMSP Broker is started, the default port number 7676 is used. In this case, you can omit the `-b` option.

E.15 Monitoring the CJMSP Broker status

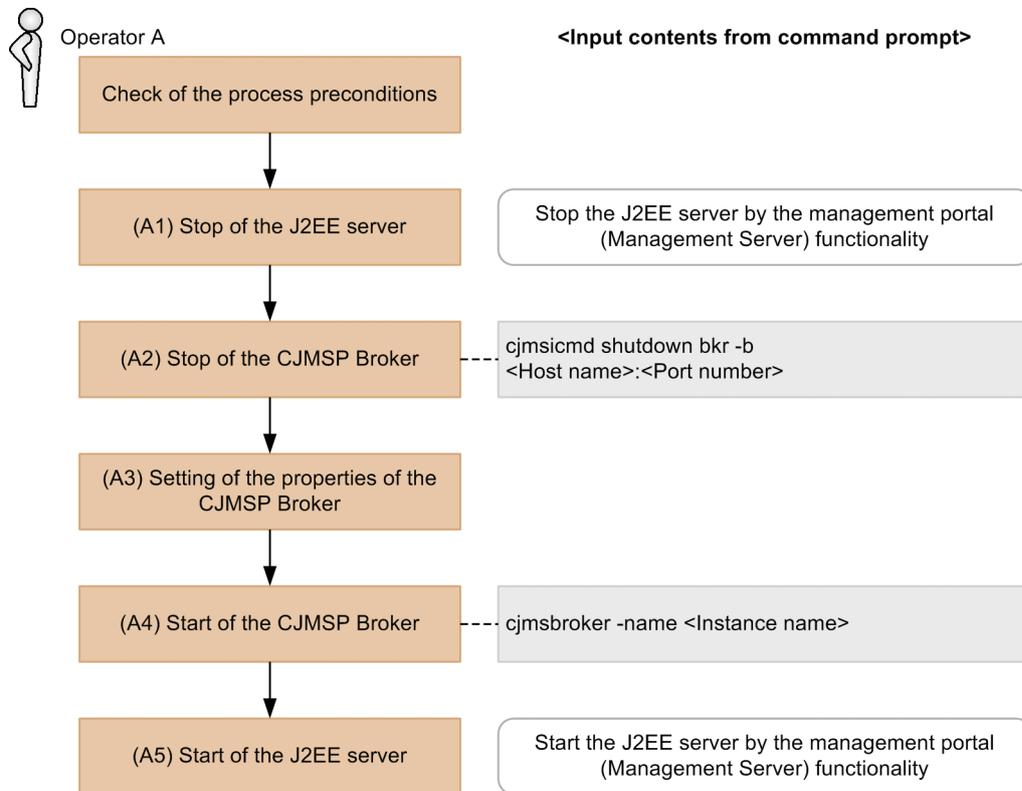
This subsection describes the procedure for starting CJMSP Broker in the monitoring mode due to debugging or error analysis.

By restarting CJMSP Broker with this procedure, the metrics information is output to the command prompt that executes CJMSP Broker.

To terminate this monitoring state, see *Appendix E.7 Starting the Cosminexus JMS Provider services (for restarting a running system)* to restart CJMSP Broker in the normal mode.

The following figure shows the procedure for monitoring the CJMSP Broker status.

Figure E-14: Procedure for monitoring the CJMSP Broker status



(1) Process preconditions

- The J2EE server process is running.
- The CJMSP resource adapter is running.
- The CJMSP Broker process is running.
- The application is running.

(2) Information and supplementary explanation required for execution

The coding, such as (A1), corresponds to the coding in the figure.

(A1)

No specific value.

(A2)

host-name:port-number

The host name and port number on which CJMSP Broker is running.

If you omit the port number when CJMSP Broker is started, the default port number 7676 is used. In this case, you can omit the `-b` option.

(A3)

The CJMSP Broker Property File is stored in `<VAR_HOME>/instances/instance-name/props/config.properties`.

To start CJMSP Broker in the status monitoring mode, specify the following property.

For details, see 7.4 *config.properties (CJMSP Broker Property File)* in the *uCosminexus Application Server Definition Reference Guide*.`img.metrics.interval`

Specifies the time interval in seconds for the output of the metrics information to the log and console.

(A4)

*instance-name*If omitted, `cjmsbroker` is used by default.

If you want to use multiple instances, make sure that the names do not overlap.

To overlap the names, specify the `-name` option, and assign any non-existent name.

An example display of the status monitoring mode is as follows:

```

Connections: 0 JVM Heap: 2932736 bytes (731352 free) Threads: 0 (14-1010)
In: 0 msgs (0 bytes) 0 pkts (0 bytes)
Out: 0 msgs (0 bytes) 0 pkts (0 bytes)
Rate In: 0 msgs/sec (0 bytes/sec) 0 pkts/sec (0 bytes/sec)
Rate Out: 0 msgs/sec (0 bytes/sec) 0 pkts/sec (0 bytes/sec)

```

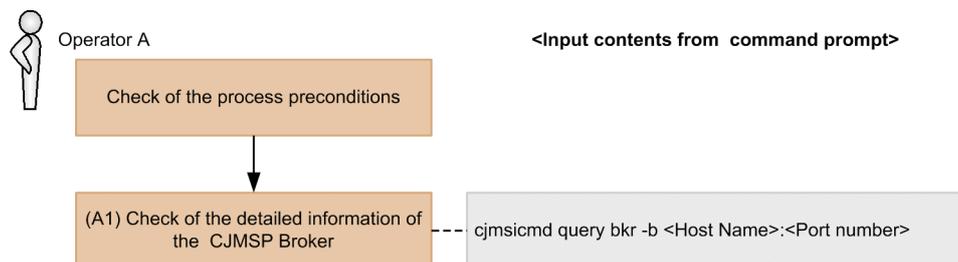
(A5)

No specific value.

E.16 Checking the CJMSP Broker details

The following figure shows the procedure for checking the latest CJMSP Broker details.

Figure E-15: Procedure for checking the CJMSP Broker details



(1) Process preconditions

- The J2EE server process is running.
- The CJMSP resource adapter is running.
- The CJMSP Broker process is running.
- The application is running.

(2) Information and supplementary explanation required for execution

The coding, such as (A1), corresponds to the coding in the figure.

(A1)

host-name:port-number

The host name and port number on which CJMSP Broker is running.

If you omit the port number when CJMSP Broker is started, the default port number 7676 is used. In this case, you can omit the `-b` option.

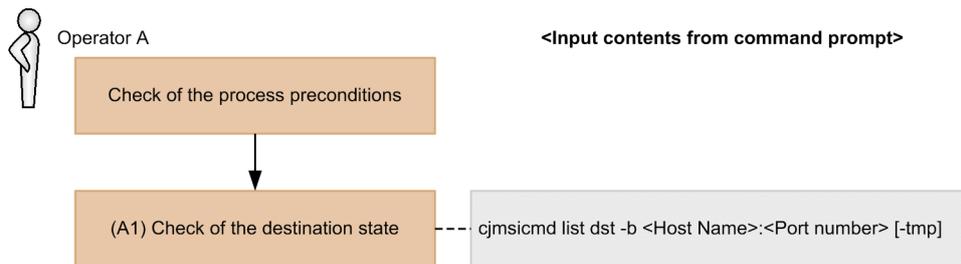
An example of display is as follows:

```
Instance Name  cjmsbroker
Primary Port   7676
Broker is Embedded false
Instance Configuration/Data Root Directory D:\Hitachi\CC\cjmsp\var
Current Number of Messages in System 843
Current Total Message Bytes in System 177030
Current Number of Messages in Dead Message Queue 0
Current Total Message Bytes in Dead Message Queue 0
KDAN34295-I Successfully queried the broker.
```

E.17 Checking the destination status

The following figure shows the procedure for checking the latest destination status.

Figure E-16: Procedure for checking the destination status



(1) Process preconditions

- The J2EE server process is running.
- The CJMSP resource adapter is running.
- The CJMSP Broker process is running.
- The application is running.

(2) Information and supplementary explanation required for execution

The coding, such as (A1), corresponds to the coding in the figure.

(A1)

host-name:port-number

The host name and port number on which CJMSP Broker is running.

If you omit the port number when CJMSP Broker is started, the default port number 7676 is used. In this case, you can omit the `-b` option.

If you use the `-tmp` option, you can check the status of the temporary destination.

The status of the temporary destination can only be checked for the duration for which the temporary destination is enabled after the application starts.

An example of display when the `-tmp` option is used is as follows:

```

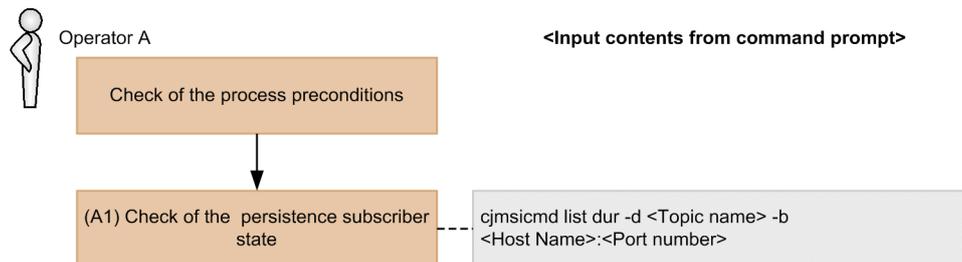
-----
Name Type State Producers Consumers Msgs
Total Count UnAck Avg Size
-----
Queue1 Queue RUNNING 0 0 0 0 0.0
mq.sys.dmq Queue RUNNING 0 0 0 0 0.0
temporary_destination://queue/10.209.10.192/2139/1 Queue (temporary) 0 1 0 0 0.0
KDAN34110-I Successfully listed destinations.
-----

```

E.18 Checking the persistence subscriber status

The following figure shows the procedure for checking the latest status of the persistence subscriber.

Figure E-17: Procedure for checking the persistence subscriber status



(1) Process preconditions

- The J2EE server process is running.
- The CJMSP resource adapter is running.
- The CJMSP Broker process is running.
- The application is running.

(2) Information and supplementary explanation required for execution

The coding, such as (A1), corresponds to the coding in the figure.

(A1)

Topic-name

Topic name containing the persistence subscriber you want to check

host-name:port-number

The host name and port number on which CJMSP Broker is running.

If you omit the port number when CJMSP Broker is started, the default port number 7676 is used. In this case, you can omit the `-b` option.

An example of display is as follows:

```

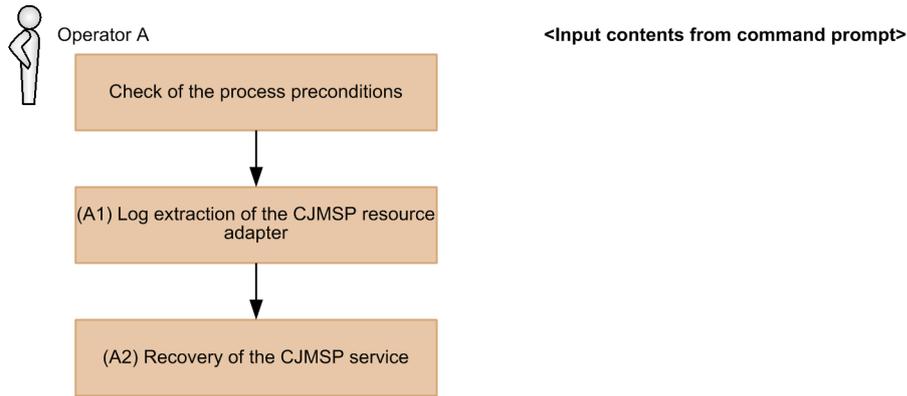
-----
Durable Sub. Name Client ID Number of Messages Durable Sub. State
-----
DurableSUB1 clientId 9998 INACTIVE
KDAN34323-I Successfully listed durable subscriptions.
-----

```

E.19 Analysis of errors in the CJMSP resource adapters

The following figure shows the procedure of analysis from the error status of the CJMSP resource adapter.

Figure E-18: Procedure for analyzing errors in the CJMSP resource adapters



(1) Process preconditions

- The J2EE server process is running.
- The CJMSP resource adapter is running.
- The CJMSP Broker process is running.
- The application is running.

(2) Information and supplementary explanation required for execution

The coding, such as (A1), corresponds to the coding in the figure.

(A1)

Check the log beneath *Cosminexus-installation-directory*/CC/server/public/ejb/server-name/logs/cjms/connector-name, and analyze the problem.

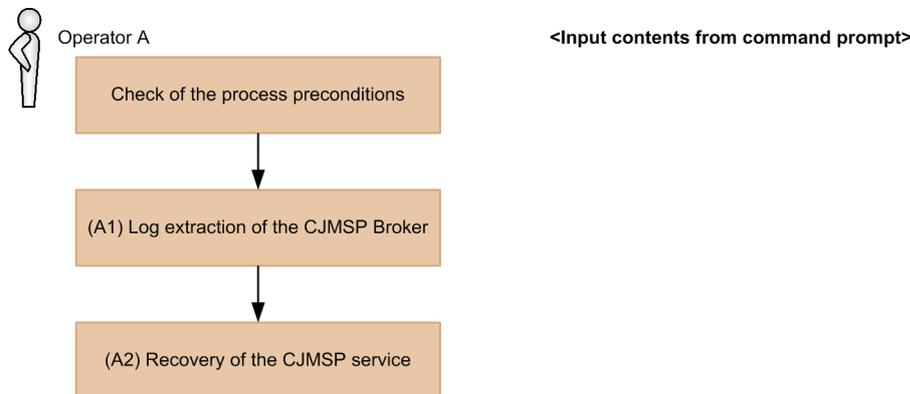
(A2)

Recover the Cosminexus JMS Provider service according to the procedure described in *Appendix E.22 Recovery when an error occurs in a CJMSP resource adapter*.

E.20 Analysis when CJMSP Broker stops due to an error

The following figure shows the procedure of analysis from the status when CJMSP Broker stops due to an error.

Figure E-19: Procedure of analysis from the status when CJMSP Broker stops due to an error



(1) Process preconditions

- The J2EE server process is running.
- The CJMSP resource adapter is running.
- The CJMSP Broker process is running.
- The application is running.

(2) Information and supplementary explanation required for execution

The coding, such as (A1), corresponds to the coding in the figure.

(A1)

Check the log beneath *Cosminexus-installation-directory/CC/cjmsp/var/instances/instance-name/log*, and analyze the problem.

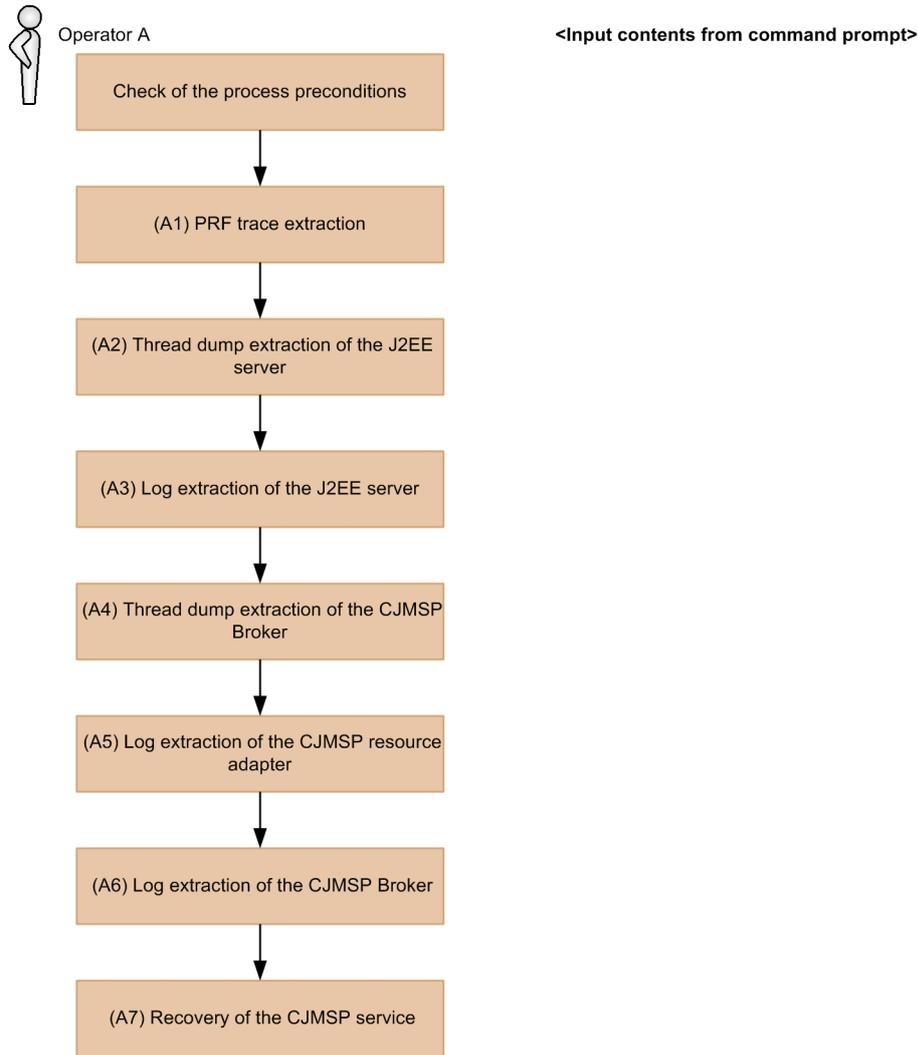
(A2)

Recover the Cosminexus JMS Provider service according to the procedure described in *Appendix E.23 Recovery when CJMSP Broker stops due to an error*.

E.21 Analysis when there is no response from a Cosminexus JMS Provider service

The following figure shows the procedure for analysis when there is no response from a Cosminexus JMS Provider service.

Figure E-20: Procedure for analysis when there is no response from a Cosminexus JMS Provider service



(1) Process preconditions

- The J2EE server process is running.
- The CJMSP resource adapter is running.
- The CJMSP Broker process is running.
- The application is running.

(2) Information and supplementary explanation required for execution

The coding, such as (A1), corresponds to the coding in the figure.

(A1)

Collect the PRF trace.

(A2)

Collect the thread dump for the J2EE server.

(A3)

Check the log beneath *Cosminexus-installation-directory/CC/server/public/ejb/server-name/logs*, and eliminate the error systematically.

(A4) Collect the thread dump for CJMSP Broker.

In Windows

There are two methods for obtaining the CJMSP Broker thread dump:

- Use the JavaVM command `jheapprof`.
For details on how to use this command, see *jheapprof (Output of extended thread dump with Hitachi class-wise statistical information)* in the *uCosminexus Application Server Command Reference Guide*.
- At the command prompt where you start CJMSP Broker, enter the `Ctrl + Break` keys.
In this case, if you obtain the thread dump and then stop CJMSP Broker, the message `Do you want to terminate the batch job? (Y/N)?` is displayed. At this time, specify `N`.

In UNIX

Use the `kill` command.

Example of execution

```
kill -3 < process ID>
```

(A5) Check the log beneath `Cosminexus-installation-directory/CC/server/public/ejb/server-name/logs/cjms/connector-name`, and eliminate the error systematically.

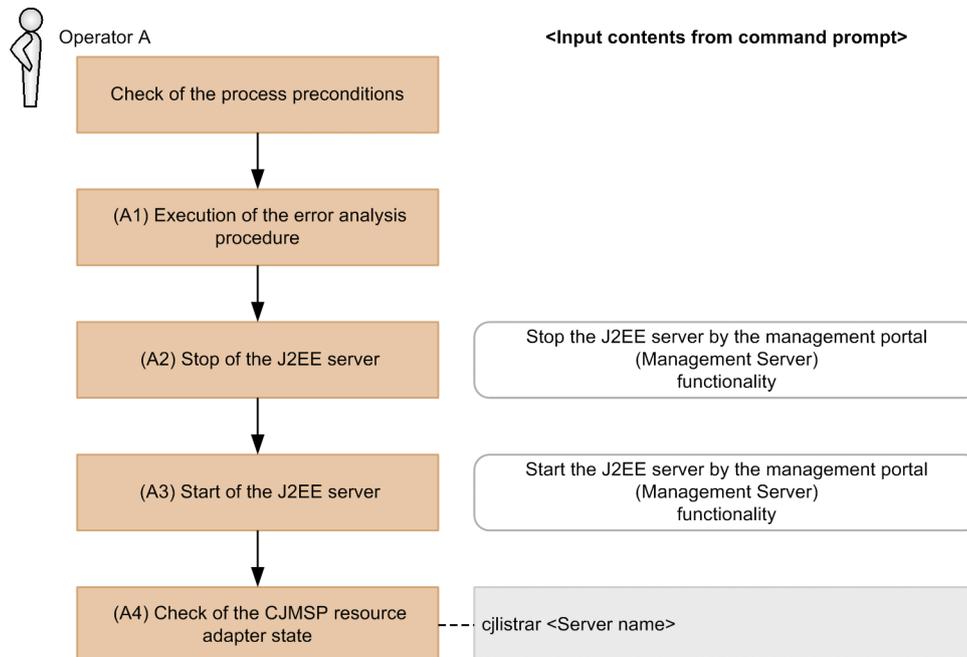
(A6) Check the log beneath `<VAR_HOME>/instances/instance-name/log`, and eliminate the error systematically.

(A7) Recover the Cosminexus JMS Provider service according to the procedure described in *Appendix E.24 Recovery when there is no response from a Cosminexus JMS Provider service*.

E.22 Recovery when an error occurs in a CJMSP resource adapter

The following figure shows the procedure for recovery when an error occurs in a CJMSP resource adapter.

Figure E-21: Procedure for recovery when an error occurs in a CJMSP resource adapter



(1) Process preconditions

- The J2EE server process is running.
- The CJMSP resource adapter is running.
- The CJMSP Broker process is running.
- The application is not running.

(2) Information and supplementary explanation required for execution

The coding, such as (A1), corresponds to the coding in the figure.

(A1)

See *Appendix E.19 Analysis of errors in the CJMSP resource adapters*, and analyze the error.

(A2)

No specific value.

(A3)

No specific value.

(A4)

server-name

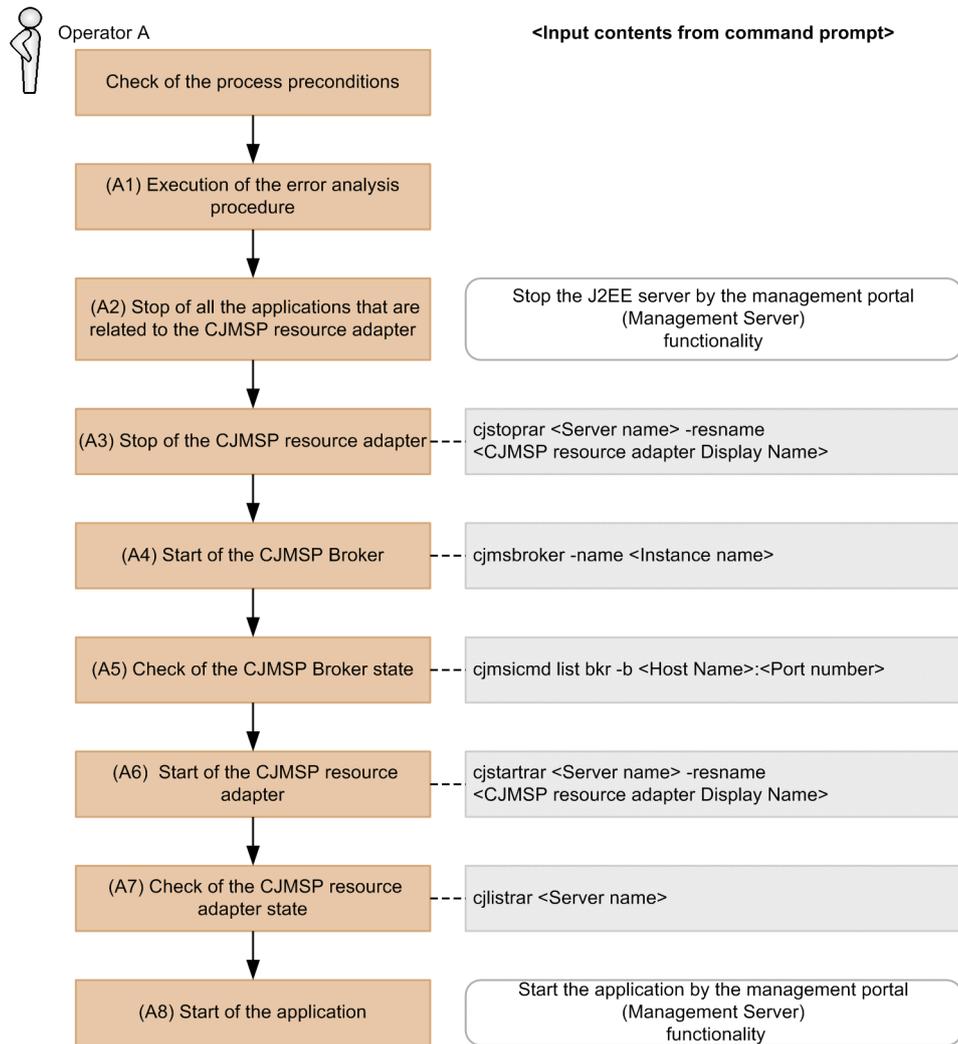
Server name set up with Management Server

Make sure that running *CJMSP-resource-adapter-display-name* is displayed.

E.23 Recovery when CJMSP Broker stops due to an error

The following figure shows the procedure for recovery when CJMSP Broker stops due to an error.

Figure E-22: Procedure for recovery when CJMSP Broker stops due to an error



(1) Process preconditions

- The J2EE server process is running.
- The CJMSP resource adapter is running.
- The CJMSP Broker process is not running.
- The application is running.

(2) Information and supplementary explanation required for execution

The coding, such as (A1), corresponds to the coding in the figure.

(A1)

See *Appendix E.20 Analysis when CJMSP Broker stops due to an error*, and analyze the error.

(A2)

No specific value.

(A3)

server-name

Server name set up with Management Server

CJMSP-resource-adapter-display-name

Display name of the CJMSP resource adapter

In Cosminexus JMS Provider, `Cosminexus_JMS_Provider_RA` is set up by default.

(A4)

instance-name

If omitted, `cjmsbroker` is used by default.

If you want to use multiple instances, make sure that the names do not overlap.

To overlap the names, specify the `-name` option, and assign any non-existent name.

(A5)

Make sure that the State is `OPERATING`.

host-name:port-number

The host name and port number on which CJMSP Broker is running.

If you omit the port number when CJMSP Broker is started, the default port number 7676 is used. In this case, you can omit the `-b` option.

(A6)

server-name

Server name set up with Management Server

CJMSP-resource-adapter-display-name

Display name of the CJMSP resource adapter

In Cosminexus JMS Provider, `Cosminexus_JMS_Provider_RA` is set up by default.

(A7)

server-name

Server name set up with Management Server

Make sure that running *CJMSP-resource-adapter-display-name* is displayed.

(A8)

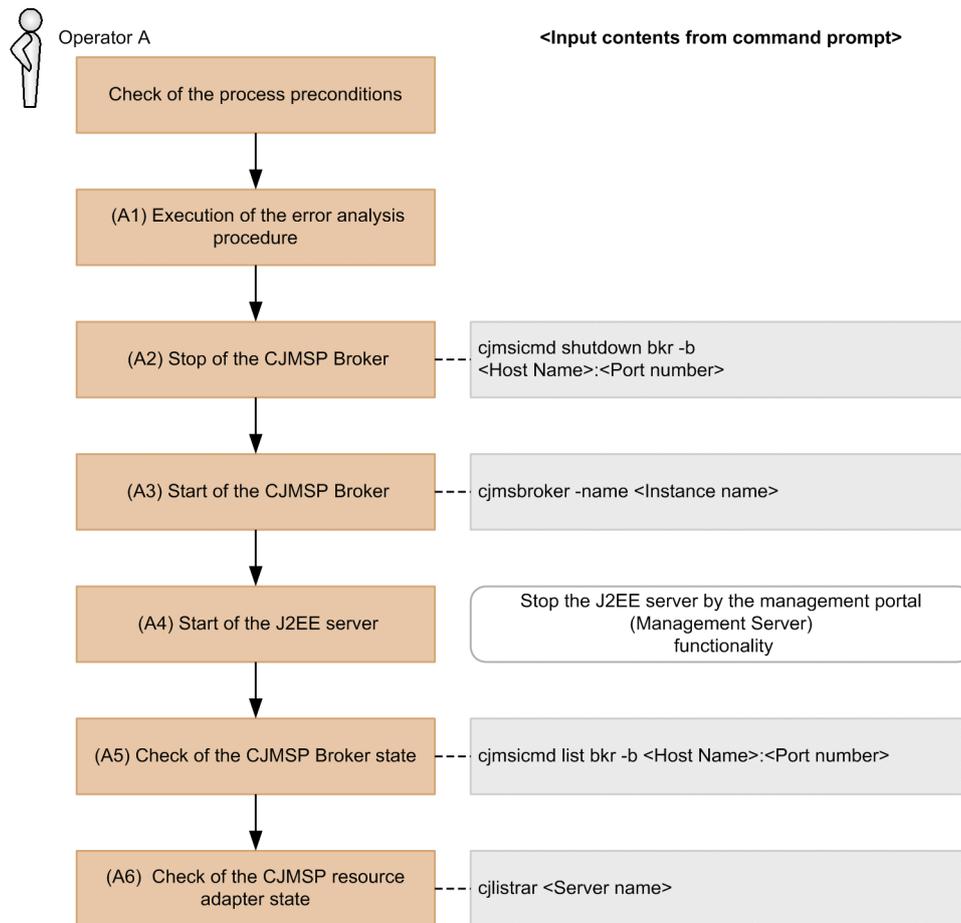
No specific value.

E.24 Recovery when there is no response from a Cosminexus JMS Provider service

This subsection describes the procedure for recovery from the no-response state of a Cosminexus JMS Provider service.

The following figure shows the procedure for recovery when there is no response from a Cosminexus JMS Provider service.

Figure E-23: Procedure for recovery when there is no response from a Cosminexus JMS Provider service



(1) Process preconditions

- The J2EE server process is running.
- The CJMSP resource adapter is running.
- The CJMSP Broker process is running.
- The application is running.

(2) Information and supplementary explanation required for execution

The coding, such as (A1), corresponds to the coding in the figure.

(A1)

See *Appendix E.21 Analysis when there is no response from a Cosminexus JMS Provider service*, and analyze the error.

(A2)

host-name:port-number

The host name and port number on which CJMSP Broker is running.

If you omit the port number when CJMSP Broker is started, the default port number 7676 is used. In this case, you can omit the `-b` option.

(A3)

instance-name

If omitted, `cjmsbroker` is used by default.

If you want to use multiple instances, make sure that the names do not overlap.
 To overlap the names, specify the `-name` option, and assign any non-existent name.

(A4)

No specific value.

(A5)

host-name:port-number

The host name and port number on which CJMSP Broker is running.

If you omit the port number when CJMSP Broker is started, the default port number 7676 is used. In this case, you can omit the `-b` option.

Make sure that the State is OPERATING.

(A6)

server-name

Server name set up with Management Server

Make sure that running *CJMSP-resource-adapter-display-name* is displayed.

E.25 Deleting the Cosminexus JMS Provider service instances

This subsection describes the procedure for deleting the running instances of a Cosminexus JMS Provider service. You perform this procedure when the previous instances are no longer required because the instance name was changed.

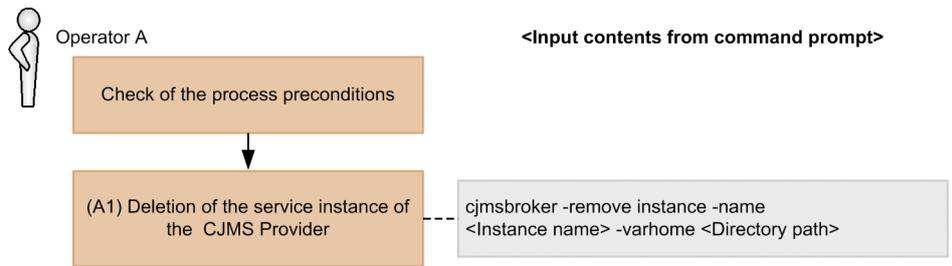
! Important note

This operation deletes all the information existing for a destination.

Before you execute this operation, make sure that there is no un-processed persistent message and persistent subscriber information.

The following figure shows the procedure for deleting the Cosminexus JMS Provider service instances.

Figure E-24: Procedure for deleting the Cosminexus JMS Provider service instances



(1) Process preconditions

- The J2EE server process is not running.
- The CJMSP resource adapter is not running.
- The CJMSP Broker process is not running.
- The application is not running.

(2) Information and supplementary explanation required for execution

The coding, such as (A1), corresponds to the coding in the figure.

(A1)

instance-name

If omitted, `cjmsbroker` is used by default.

If you want to use multiple instances, make sure that the names do not overlap.

To overlap the names, specify the `-name` option, and assign any non-existent name.

Directory-path

`VAR_HOME` path containing the instances you want to delete

Deletes the directory (default name `cjmsbroker`) corresponding to the CJMSP Broker instance name.

If you perform this processing, the files and directories related to all the instances are deleted, except the `prop` and `log` directories.

If the instance directory is not required, go to the following directory through the explorer and delete the directory manually:

`<VAR_HOME>/instances`

When starting CJMSP Broker next time, if you use the deleted instance name to start CJMSP Broker again, the instance is re-created.

F. Main Functionality Changes in Each Version

This section describes the main functionality changes in Application Server versions prior to 09-50 and the purpose of each change. For details on the main functionality changes in 09-50, see *1.4 Main functionality changes in Application Server 09-50*.

The contents described in this section are as follows:

- This section gives an overview and describes the main changes in the functionality of various Application Server versions. For details on the functionality, see the description in the *Reference location* column in the *Reference manual* column. The main locations where the functionality is described in the 09-50 manual are mentioned in the *Reference manual* and *Reference location* columns.
- The term *uCosminexus Application Server* is omitted from the manual names mentioned in the *Reference manual* column.

F.1 Main functionality changes in 09-00

(1) Simplifying installation and setup

The following table describes the items that are changed to simplify installation and setup.

Table F-1: Changes made for simplifying installation and setup

Item	Overview of changes	Reference manual	Reference section
Changing the units to be set up and operated in the virtual environment	The units to be operated when you set up and operate the virtual environment have been changed from the virtual server to the virtual server group. You can now use the file in which the virtual server group is defined and register multiple virtual servers into a management unit in a batch.	<i>Virtual System Setup and Operation Guide</i>	1.1.2
Cancelling the restrictions on the setup environment by using Setup Wizard	The restrictions on the environments that can be set up with Setup Wizard have been removed. Even if an environment has been set up with another functionality, you can now unset up the environment and use Setup Wizard for the setup.	<i>System Setup and Operation Guide</i>	2.2.7
Simplifying the procedure for deleting the setup environment	The deletion procedure has now been simplified by adding the functionality to delete the system environment set up with Management Server (mngunsetup command).	<i>System Setup and Operation Guide</i>	4.1.37
		<i>Management Portal User Guide</i>	3.6, 5.4
		<i>Command Reference Guide</i>	mngunsetup (Deleting the Management Server configuration environment)

(2) Supporting standard and existing functionality

The following table describes the items that are changed to support standard and existing functionality.

Table F-2: Changes made for supporting standard and existing functionality

Item	Overview of changes	Reference manual	Reference section
Supporting Servlet 3.0	Servlet 3.0 is now supported.	<i>Web Container Functionality Guide</i>	<i>Chapter 6</i>
Supporting EJB 3.1	EJB 3.1 is now supported.	<i>EJB Container Functionality Guide</i>	<i>Chapter 2</i>
Supporting JSF 2.1	JSF 2.1 is now supported.	<i>Web Container Functionality Guide</i>	<i>Chapter 3</i>
Supporting JSTL 1.2	JSTL 1.2 is now supported.	<i>Web Container Functionality Guide</i>	<i>Chapter 3</i>
Supporting CDI 1.0	CDI 1.0 is now supported.	This manual	<i>Chapter 9</i>
Using the Portable Global JNDI names	You can now look up objects using a Portable Global JNDI name.	This manual	<i>2.4</i>
Supporting JAX-WS 2.2	JAX-WS 2.2 is now supported.	<i>Web Service Development Guide</i>	<i>1.1, 16.1.5, 16.1.7, 16.2.1, 16.2.6, 16.2.10, 16.2.12, 16.2.13, 16.2.14, 16.2.16, 16.2.17, 16.2.18, 16.2.20, 16.2.22, 19.1, 19.2.3, 37.2, 37.6.1, 37.6.2, 37.6.3</i>
Supporting JAX-RS 1.1	JAX-RS 1.1 is now supported.	<i>Web Service Development Guide</i>	<i>1.1, 1.2.2, 1.3.2, 1.4.2, 1.5.1, 1.6, 2.3, Chapter 11, Chapter 12, Chapter 13, Chapter 17, Chapter 24, Chapter 39</i>

(3) Maintaining and improving reliability

The following table describes the items that are changed for maintaining and improving reliability.

Table F-3: Changes made for maintaining and improving reliability

Item	Overview of changes	Reference manual	Reference section
Using TLSv 1.2 for the SSL/TLS communication	You can now use RSA BSAFE SSL-J to execute the SSL/TLS communication with a security protocol containing TLSv 1.2.	<i>Security Management Guide</i>	<i>7.3</i>

(4) Maintaining and improving operability

The following table describes the items that are changed with the purpose of maintaining and improving operability.

Table F-4: Changes made for maintaining and improving operability

Item	Overview of changes	Reference manual	Reference section
Monitoring the total pending queues of the entire Web container	You can now output the total pending queues of the entire Web container in the operation information and monitor the number of queues.	<i>Operation, Monitoring, and Linkage Guide</i>	Chapter 3
Output of trace-based performance analysis for applications (user-extended trace)	The trace-based performance analysis used for analyzing the processing performance of the user-developed applications can now be output without changing the applications.	<i>Maintenance and Migration Guide</i>	Chapter 7
Operations performed by using the user script in a virtual environment	The user-created script (user script) can now be executed on the virtual server at any time.	<i>Virtual System Setup and Operation Guide</i>	7.8
Improving the management portal	Changes have been made so that the messages describing procedures are now displayed on the following management portal windows: <ul style="list-style-type: none"> • Deploy Preference Information window • Start windows for the Web server, J2EE server, and SFO server • Batch start, batch restart, and start windows for Web server cluster and J2EE server cluster 	<i>Management Portal User Guide</i>	10.11.1, 11.9.2, 11.10.2, 11.11.2, 11.11.4, 11.11.6, 11.12.2, 11.13.2, 11.13.4, 11.13.6
Adding the restart functionality in the management functionality	You can now specify automatic restart for the management functionality (Management Server and Administration Agent) and continue operations even when an error occurs in the management functionality. The procedure for automatic start settings has also been changed.	<i>Operation, Monitoring, and Linkage Guide</i>	2.4.1, 2.4.2, 2.6.3, 2.6.4
		<i>Command Reference Guide</i>	<i>mngautorun (Setting up/ canceling the setup of autostart and autorestart)</i>

(5) Other purposes

The following table describes the items that are changed for other purposes.

Table F-5: Changes made for other purposes

Item	Overview of changes	Reference manual	Reference section
Changing the file switching units for log output	The output destination files are now switched by date when the log is output.	<i>Maintenance and Migration Guide</i>	3.2.1
Changing the Web server name	The name of the Web server included in Application Server is changed to HTTP Server.	<i>HTTP Server User Guide</i>	--
Supporting direct connection by using the BIG-IP APIs (SOAP architecture)	Direct connection is now supported by using APIs (SOAP architecture) in BIG-IP (load balancer). Also, the procedure for setting up the connection environment of the load balancer has been changed for using direct connection through APIs.	<i>System Setup and Operation Guide</i>	4.7.3, Appendix K
		<i>Virtual System Setup and Operation Guide</i>	2.1, Appendix C
		<i>Security Management Guide</i>	8.2, 8.4, 8.5, 8.6, 18.2, 18.3, 18.4

Legend:

--: Reference the entire manual.

F.2 Main functionality changes in 08-70

(1) Simplifying installation and setup

The following table describes the items that are changed to simplify installation and setup.

Table F-6: Changes made for simplifying installation and setup

Item	Overview of changes	Reference manual	Reference location
Improving the management portal	Changes have been made to enable users to set the property (settings of the HITACHI Connector Property file) for defining the resource adapter attributes and perform the connection test in the management portal window. Also, J2EE applications (ear files and zip files) can now be uploaded on Management Server using the management portal window.	<i>First Step Guide</i>	3.5
		<i>Management Portal User Guide</i>	--
Adding the functionality for implicitly importing the <code>import</code> attribute for the <code>page/tag</code> directive	The functionality for implicitly importing the <code>import</code> attribute of the <code>page/tag</code> directive can now be used.	<i>Web Container Functionality Guide</i>	2.3.7
Supporting automation of the environment settings for the JP1 products in a virtual environment	Changes have been made so that when Application Server is set up on a virtual server, the environment settings of JP1 products can be automatically set for the virtual server by using the hook script.	<i>Virtual System Setup and Operation Guide</i>	7.7.2
Improving the Integrated user management functionality	When using a database in a user information repository, you can now connect to the database by using the JDBC driver of the database products. The database connection through the JDBC driver of Cosminexus DABroker Library is no longer supported. You can now set the integrated user management functionality using the Easy Setup definition file and the management portal window. Active Directory now supports double byte characters such as the Japanese language in the DN.	<i>Security Management Guide</i>	Chapter 5, 14.3
		<i>Management Portal User Guide</i>	3.5, 10.9.1
Enhancing HTTP Server settings	You can now directly set the directive (settings of <code>httpsd.conf</code>) that defines the operation environment of HTTP Server, using the Easy Setup definition file and the management portal window.	<i>System Setup and Operation Guide</i>	4.1.21
		<i>Management Portal User Guide</i>	10.10.1
		<i>Definition Reference Guide</i>	4.13

Legend:

--: Reference the entire manual.

(2) Supporting standard and existing functionality

The following table describes the items that are changed to support standard and existing functionality.

Table F-7: Changes made for supporting standard and existing functionality

Item	Overview of changes	Reference manual	Reference location
Adding items to be specified in <code>ejb-jar.xml</code>	A class level interceptor and a method level interceptor can now be specified in <code>ejb-jar.xml</code> .	<i>EJB Container Functionality Guide</i>	2.15
Supporting the parallel copy garbage collection	The parallel copy garbage collection can now be selected.	<i>Definition Reference Guide</i>	16.5

F. Main Functionality Changes in Each Version

Item	Overview of changes	Reference manual	Reference location
Supporting global transactions with the Inbound resource adapters conforming to the Connector 1.5 specifications	<code>Transacted Delivery</code> can now be used in the resource adapters conforming to the Connector 1.5 specifications. This enables participation of the EIS invoking the Message-driven Beans in a global transaction.	This manual	3.16.3
Supporting the MHP with a TP1 inbound adapter	The MHP can now be used as the <code>OpenTP1client</code> that invokes <code>Application Server</code> by using the TP1 inbound adapter.	This manual	Chapter 4
Supporting the FTP inbound adapter with the <code>cjrarupdate</code> command	The FTP inbound adapter has been added to the resource adapters that can be upgraded by using the <code>cjrarupdate</code> command.	<i>Command Reference Guide</i>	2.2

(3) Maintaining and improving reliability

The following table describes the items that are changed for maintaining and improving reliability.

Table F-8: Changes made for maintaining and improving reliability

Item	Overview of changes	Reference manual	Reference location
Improving the database session failover functionality	Users can now select a mode that does not lock the database in which the global session information is stored, in a performance-centric system. Also, exclusive requests for references can now be defined without updating the database.	<i>Expansion Guide</i>	Chapter 6
Expansion of the processing for the OutOfMemory handling functionality	Processing has been added for the OutOfMemory handling functionality.	<i>Maintenance and Migration Guide</i>	2.5.7
		<i>Definition Reference Guide</i>	16.2
Adding the memory saving functionality for the Explicit heap used in an HTTP session	Functionality has been added to control the amount of Explicit heap memory used in an HTTP session.	<i>Expansion Guide</i>	8.11

(4) Maintaining and improving operability

The following table describes the items that are changed with the purpose of maintaining and improving operability.

Table F-9: Changes with the purpose of maintaining and improving operability

Item	Overview of changes	Reference manual	Reference location
Supporting user authentication using the JP1 products in a virtual environment (supporting cloud operations)	The management and authentication of users using a virtual server manager can now be performed by using the authentication server of the JP1 products during JP1 integration.	<i>Virtual System Setup and Operation Guide</i>	1.2.2, Chapter 3, Chapter 4, Chapter 5, Chapter 6, 7.9

(5) Other purposes

The following table describes the items that are changed for other purposes.

Table F-10: Changes due to other purposes

Item	Overview of changes	Reference manual	Reference location
Supporting the direct connection using APIs (REST Architecture) for the load balancers	Direct connection using APIs (REST architecture) is now supported as a method to connect to the load balancers. Also, ACOS (AX2500) is added to the types of available load balancers.	<i>System Setup and Operation Guide</i>	4.7.2, 4.7.3
		<i>Virtual System Setup and Operation Guide</i>	2.1
		<i>Definition Reference Guide</i>	4.5
Supporting timeout during snapshot log collection and improving the collection targets	You can now terminate snapshot log collection (timeout) at a specified time. The contents collected as primarily sent data have been changed.	<i>Maintenance and Migration Guide</i>	<i>Appendix A</i>

F.3 Main functionality changes in 08-53

(1) Simplifying installation and setup

The following table describes the items that are changed to simplify installation and setup.

Table F-11: Changes made for simplifying installation and setup

Item	Overview of changes	Reference manual	Reference location
Setting up a virtual environment supporting various hypervisors	Application Server can now be set up on a virtual server implemented by using various hypervisors. An environment in which multiple hypervisors co-exist is also supported now.	<i>Virtual System Setup and Operation Guide</i>	<i>Chapter 2, Chapter 3, Chapter 5</i>

(2) Supporting standard and existing functionality

The following table describes the items that are changed to support standard and existing functionality.

Table F-12: Changes made for supporting standard and existing functionality

Item	Overview of changes	Reference manual	Reference location
Invocation from OpenTP1 supporting transaction integration	Transactions can now be integrated when the Message-driven Beans running on Application Server are invoked from OpenTP1	This manual	<i>Chapter 4</i>
JavaMail	The mail receiving functionality, which uses the APIs conforming to JavaMail 1.3 by integrating with the mail server conforming to POP3, is now available.	This manual	<i>Chapter 8</i>

(3) Maintaining and improving reliability

The following table describes the items that are changed for maintaining and improving reliability.

Table F-13: Changes made for maintaining and improving reliability

Item	Overview of changes	Reference manual	Reference location
Enhancing the JavaVM troubleshooting functionality	The following functionality can now be used as the JavaVM troubleshooting functionality:	<i>Maintenance and Migration Guide</i>	<i>Chapter 4, Chapter 5, Chapter 9</i>

Item	Overview of changes	Reference manual	Reference location
Enhancing the JavaVM troubleshooting functionality	<ul style="list-style-type: none"> The operations when OutOfMemoryError occurs can now be changed. You can now set up an upper limit for the amount of C heap allocated for JIT compilation. You can now set up the maximum thread count. The output items of the extended <code>verbosegc</code> information have been extended. 	<i>Maintenance and Migration Guide</i>	<i>Chapter 4, Chapter 5, Chapter 9</i>

(4) Maintaining and improving operability

The following table describes the items that are changed with the purpose of maintaining and improving operability.

Table F-14: Changes with the purpose of maintaining and improving operability

Item	Overview of changes	Reference manual	Reference location
Supporting JP1/ITRM	JP1/ITRM, a product that uniformly manages the IT resources, is now supported.	<i>Virtual System Setup and Operation Guide</i>	<i>1.3, 2.1</i>

(5) Other purposes

The following table describes the items that are changed for other purposes.

Table F-15: Changes due to other purposes

Item	Overview of changes	Reference manual	Reference location
Supporting Microsoft IIS 7.0 and Microsoft IIS 7.5	Microsoft IIS 7.0 and Microsoft IIS 7.5 are now supported as Web servers.	--	--
Supporting HiRDB Version 9 and SQL Server 2008	<p>The following database products are now supported:</p> <ul style="list-style-type: none"> HiRDB Server Version 9 HiRDB/Developer's Kit Version 9 HiRDB/Run Time Version 9 SQL Server 2008 <p>Also, SQL Server JDBC Driver is now supported as the JDBC driver for SQL Server 2008.</p>	This manual	<i>Chapter 3</i>

Legend:

--: Not applicable.

F.4 Main functionality changes in 08-50

(1) Simplifying installation and setup

The following table describes the items that are changed to simplify installation and setup.

Table F-16: Changes made for simplifying installation and setup

Item	Overview of changes	Reference manual	Reference location
Changing the mandatory tags for specifying <code>web.xml</code> in the Web Service provider	The specification of the <code>listener</code> tag, <code>servlet</code> tag, and <code>servlet-mapping</code> tag was changed from mandatory to optional in <code>web.xml</code> in the Web Service provider.	<i>Definition Reference Guide</i>	2.4
Using the network resources of the logical server	The functionality for accessing the network resources and network drive existing on another host from the J2EE applications was added.	<i>Operation, Monitoring, and Linkage Guide</i>	1.2.3, 5.2, 5.7
Simplifying the execution procedure of the sample program	The procedure of executing the sample program was simplified by providing a part of the sample program in the EAR format.	<i>First Step Guide</i>	3.5
		<i>System Setup and Operation Guide</i>	Appendix M
Improving the operations in the management portal window	The default update interval of the window was changed from "Do not update" to "3 seconds".	<i>Management Portal User Guide</i>	7.4.1
Improving the Setup Wizard completion window	The Easy Setup definition file and HITACHI Connector Property file used for setup are now displayed on the window displayed when the Setup Wizard finishes operations.	<i>System Setup and Operation Guide</i>	2.2.6
Setting up the virtual environment	The procedure of setting up Application Server on the virtual server, implemented by using a hypervisor, has been added. #	<i>Virtual System Setup and Operation Guide</i>	Chapter 3, Chapter 5

#

To set up in the 08-50 mode, see *Appendix D Settings to Use the Virtual Server Manager in the 08-50 Mode* in the *uCosminexus Application Server Virtual System Setup and Operation Guide*.

(2) Supporting standard and existing functionality

The following table describes the items that are changed to support standard and existing functionality.

Table F-17: Changes made for supporting standard and existing functionality

Item	Overview of changes	Reference manual	Reference location
Supporting invocation from OpenTP1	The Message-driven Beans running on Application Server can now be invoked from OpenTP1.	This manual	Chapter 4
Supporting JMS	The Cosminexus JMS Provider functionality conforming to the JMS 1.1 specifications can now be used.	This manual	Chapter 7
Supporting Java SE 6	The Java SE 6 functionality can now be used.	<i>Maintenance and Migration Guide</i>	5.5, 5.8.1
Supporting the use of generics	Generics can now be used with EJB.	<i>EJB Container Functionality Guide</i>	4.2.19

(3) Maintaining and improving reliability

The following table describes the items that are changed for maintaining and improving reliability.

Table F-18: Changes made for maintaining and improving reliability

Item	Overview of changes	Reference manual	Reference location
Improving the usability of the Explicit Memory Management functionality	The Explicit Memory Management functionality can now be used easily by using the automatic allocation configuration file.	<i>System Design Guide</i>	7.1.1, 7.6.3, 7.10.5, 7.11.1

F. Main Functionality Changes in Each Version

Item	Overview of changes	Reference manual	Reference location
Improving the usability of the Explicit Memory Management functionality	The Explicit Memory Management functionality can now be used easily by using the automatic allocation configuration file.	<i>Expansion Guide</i>	<i>Chapter 8</i>
Controlling the database session failover functionality for URIs	When using the database session failover functionality, you can now specify the requests that will be outside the scope of the functionality for URIs.	<i>Expansion Guide</i>	<i>5.6.1</i>
Monitoring errors in the virtual environment	In a virtual system, the virtual server is now monitored and the occurrence of errors can be detected.	<i>Virtual System Setup and Operation Guide</i>	<i>Appendix D</i>

(4) Maintaining and improving operability

The following table describes the items that are changed with the purpose of maintaining and improving operability.

Table F-19: Changes with the purpose of maintaining and improving operability

Item	Overview of changes	Reference manual	Reference location
Omitting the management user account	The login ID and password of the user can now be omitted in the management portal, Management Server commands, or Smart Composer functionality commands.	<i>System Setup and Operation Guide</i>	<i>4.1.15</i>
		<i>Management Portal User Guide</i>	<i>2.2, 7.1.1, 7.1.2, 7.1.3, 8.1, 8.2.1, Appendix F. 2</i>
		<i>Command Reference Guide</i>	<i>1.4, mngsvrctl (Starting/ stopping/ setting up Management Server), mngsvrutil (Management Server management commands), 8.3, cmx_admin_passwd (Setting up the management user account of Management Server)</i>
Virtual environment operations	The procedure of executing batch start, batch stop, scale in, and scale out for multiple virtual servers has been added in the virtual system. #	<i>Virtual System Setup and Operation Guide</i>	<i>Chapter 4, Chapter 6</i>

#

To operate in the 08-50 mode, see *Appendix D Settings to Use the Virtual Server Manager in the 08-50 Mode* in the *uCosminexus Application Server Virtual System Setup and Operation Guide*.

(5) Other purposes

The following table describes the items that are changed for other purposes.

Table F-20: Changes due to other purposes

Item	Overview of changes	Reference manual	Reference location
Unused objects statistical functionality in the Tenured area	Only the unused objects can now be identified in the Tenured area.	<i>Maintenance and Migration Guide</i>	9.8
Base object list output functionality for Tenured augmentation factors	The information of the objects that form the base of unused objects, identified by using the unused objects statistical functionality in the Tenured area, can now be output.		9.9
Class-wise statistical information analysis functionality	The class-wise statistical information can now be output in the CSV format.		9.10
Cluster node switching based on the detection of the exceeding of the auto-restart frequency of the logical server	Node switching can now be executed when the logical server is in an abnormally stopped state (state in which the auto-restart count is exceeded, or state in which an error is detected when the auto-restart count is set to 0) in a cluster configuration wherein Management Server is to be monitored for node switching.	<i>Operation, Monitoring, and Linkage Guide</i>	18.4.3, 18.5.3, 20.2.2, 20.3.3, 20.3.4
Node switching system for the host unit management model	Node switching can now be executed for the host unit management model in the system operations integrated with the cluster software.		Chapter 20
Supporting ACOS (AX2000, BS320)	ACOS (AX2000, BS320) was added to the types of load balancers that are available.	<i>System Setup and Operation Guide</i>	4.7.2, 4.7.3, 4.7.5, 4.7.6, Appendix K, Appendix K.2
		<i>Definition Reference Guide</i>	4.5, 4.6.2, 4.6.4, 4.6.5, 4.6.6, 4.10.1
Transaction attributes that can be specified in the Stateful Session Bean (SessionSynchronization) for CMT-managed transactions, have been added	Supports, NotSupported, and Never can now be specified as the transaction attributes in the Stateful Session Bean (SessionSynchronization), for CMT-managed transactions.	<i>EJB Container Functionality Guide</i>	2.7.3
Forcefully terminating Administration Agent when OutOfMemoryError occurs	Administration Agent is now forcefully terminated when OutOfMemoryError occurs in JavaVM.	<i>Maintenance and Migration Guide</i>	2.5.8
Asynchronous parallel processing of threads	Asynchronous timer processing and asynchronous thread processing can now be implemented by using <code>TimerManager</code> and <code>WorkManager</code> .	<i>Expansion Guide</i>	Chapter 10

F.5 Main functionality changes in 08-00

(1) Improvement of development productivity

The following table describes the items changed with the purpose of improving the development productivity.

Table F-21: Changes made with the purpose of improving the development productivity

Item	Overview of changes	Reference manual	Reference location
Easy migration from other application server products	The use of the following functionality was enabled for smooth migration from other Application Server products:	<i>Web Container Functionality Guide</i>	2.3, 2.7.5

F. Main Functionality Changes in Each Version

Item	Overview of changes	Reference manual	Reference location
Easy migration from other application server products	<ul style="list-style-type: none"> The upper limit of the HTTP sessions can now be determined using exceptions. If the JavaBeans ID is repeated and if the upper case/lower case in the custom tag attribute name and TLD definition are different, the occurrence of translation errors can now be controlled. 	<i>Web Container Functionality Guide</i>	2.3, 2.7.5
Provision of <code>cosminexus.xml</code>	By specifying attributes unique to Cosminexus Application Server in <code>cosminexus.xml</code> , the J2EE applications can now be started after being imported into the J2EE server without specifying the property settings.	This manual	11.3

(2) Supporting standard functionality

The following table describes the items changed with the purpose of supporting the standard functionality.

Table F-22: Changes made with the purpose of supporting the standard functionality

Item	Overview of changes	Reference manual	Reference location
Supporting Servlet 2.5	Servlet 2.5 is now supported.	<i>Web Container Functionality Guide</i>	2.2, 2.5.4, 2.6, Chapter 6
Supporting JSP 2.1	JSP 2.1 is now supported.	<i>Web Container Functionality Guide</i>	2.3.1, 2.3.3, 2.5, 2.6, Chapter 6
JSP debug	JSP debug can now be performed in the development environment in which MyEclipse is used. [#]	<i>Web Container Functionality Guide</i>	2.4
Storing the tag library in the library JAR and TLD mapping	When the tag library is stored in the library JAR, the TLD file in the library JAR is searched by the Web Container when the Web application starts and is automatically mapped.	<i>Web Container Functionality Guide</i>	2.3.4
Omitting <code>application.xml</code>	<code>application.xml</code> can now be omitted in the J2EE application.	This manual	11.4
Combined use of annotations and DD	The annotations and DD can now be used together and the contents specified in an annotation can be updated with the DD.	This manual	12.5
Conforming annotation to Java EE 5 standards (default interceptor)	The default interceptor can now be stored in the library JAR. Also, DI can now be executed from the default interceptor.	This manual	11.4
Resolving the <code>@Resource</code> references	The resource references can now be resolved with <code>@Resource</code> .	This manual	12.4
Supporting JPA	The JPA specifications are now supported.	This manual	Chapter 5, Chapter 6

[#] In version 09-00 and later, you can use the JSP debug functionality in the development environment using WTP.

(3) Maintaining and improving reliability

The following table describes the items changed with the purpose of maintaining and improving reliability.

Table F-23: Changes made for maintaining and improving reliability

Item	Overview of changes	Reference manual	Reference location
Persistence of session information	The session information of the HTTP session can now be saved in the database and then and inherited.	<i>Expansion Guide</i>	<i>Chapter 5, Chapter 6</i>
Controlling the full garbage collection	By deploying the object that causes full garbage collection outside the Java heap, the occurrence of full garbage collection can now be controlled.	<i>Expansion Guide</i>	<i>Chapter 8</i>
Client performance monitor	The time required for client processing can now be checked and analyzed.	--	--

Legend:

--: This functionality has been deleted in 09-00.

(4) Maintaining and improving operability

The following table describes the items that are changed with the purpose of maintaining and improving operability.

Table F-24: Changes with the purpose of maintaining and improving operability

Item	Overview of changes	Reference manual	Reference location
Improving the operability of applications on the management portal	The server management commands and management portal can now be interoperated for application and resource operations.	<i>Management Portal User Guide</i>	<i>1.1.3</i>

(5) Other purposes

The following table describes the items that are changed for other purposes.

Table F-25: Changes due to other purposes

Item	Overview of changes	Reference manual	Reference location
Deleting the disabled HTTP Cookies	The disabled HTTP Cookies can now be deleted.	<i>Web Container Functionality Guide</i>	<i>2.7.4</i>
Detecting errors in the Naming Service	When an error occurs in a Naming Service, the EJB client can now detect the error faster.	This manual	<i>2.9</i>
Connection error detection timeout	A timeout value can now be specified for connection error detection timeout.	This manual	<i>3.15.1</i>
Supporting Oracle11g	Oracle11g can now be used as the database.	This manual	<i>Chapter 3</i>
Scheduling of batch processing	The execution of batch applications can now be scheduled using the CTM.	<i>Expansion Guide</i>	<i>Chapter 4</i>
Batch processing log	The retry frequency and retry interval can now be specified for the error in log file size, number of log files, and log exclusion processing of the batch execution commands.	<i>Definition Reference Guide</i>	<i>3.6</i>
snapshot log	The collected contents of the snapshot log are changed.	<i>Maintenance and Migration Guide</i>	<i>Appendix A.1, Appendix A.2</i>
Publication of the protected area of method cancellation	The contents of protected area list that is outside the scope of method cancellation are published.	<i>Operation, Monitoring, and Linkage Guide</i>	<i>Appendix C</i>
Pre-statistical garbage collection select functionality	You can now select whether to execute garbage collection before the class-wise statistical information is output.	<i>Maintenance and Migration Guide</i>	<i>9.7</i>

F. Main Functionality Changes in Each Version

Item	Overview of changes	Reference manual	Reference location
Functionality to output the age distribution information of the Survivor area	The age distribution information of the Java objects in the Survivor area can now be output to the JavaVM log file.	<i>Maintenance and Migration Guide</i>	9.11
Finalize retention cancellation functionality	The JavaVM finalize processing status can now be monitored and the retention of processing can be cancelled.	--	--
Changing the maximum heap size of the server management commands	The maximum heap size used by the server management commands was changed.	<i>Definition Reference Guide</i>	5.2, 5.3
Action for cases when un-recommended display names are specified	A message is now displayed when an un-recommended display name is specified in the J2EE applications.	<i>Messages</i>	KDJE42374-W

Legend:

--: This functionality has been deleted in 09-00.

G. Glossary

Terminology used in this manual

For the terms used in the manual, see *uCosminexus Application Server and BPM/ESB Platform Terminology Guide*.

Index

A

Abstract entity class 351
Abstract schema name 367
Abstract schema type 367
Accessing detached entity 315
ActivationSpec settings 225
Adding business logic to accessor method 345
admin service 400
advantages of applications using JPA 261
advantages of using applications containing
 cosminexus.xml 470
afterDelivery method 221
annotation 494
annotation functionality 16
Annotations for relationships 319
annotations that can be specified in Enterprise Beans 495
annotations that can be specified in Web application 495
Applicable scope of Bean Validation 454
application directory 529
applications containing cosminexus.xml 469
Application Server-managed transactions 114
Application Server-specific Connector 1.5 API
 specifications 220
archive-format J2EE application 528
archive-format J2EE applications 528
Assigning optional names to EJB home object references 62
Assigning optional names to J2EE resources 63
attributes specified in <persistence-unit> tag 283
automatically closing connections 187, 196
automatically closing objects 195
Auto-numbering of primary key values 337
available components 265
available functionality 145
available J2EE components and functionality 129
available JPA providers 264
available resource adapters 267

B

bean validation functionality and bean validation operations
 455
Bean Validation functionality used with Application Server
 15
beforeDelivery method 221
behavior of naming service error detection functionality 80
Bi-directional ManyToMany relationship 323
Bi-directional ManyToOne/OneToMany relationship 322
Bi-directional OneToOne relationship 321
binding and looking up objects in JNDI name space 29
BNF for JPQL 631
Bulk DELETE statement 376
Bulk UPDATE statement 376

C

cache-clearing range 75

Cache functionality of entity objects 329
Cache reference forms and cache types 331
Cache registration and update timing 329
caching DataSource objects 176, 178
caching with naming management functionality 74
CDI functionality used with Application Server 15
changing application directory 533
changing output destination of session information file 573
changing settings for functionality for controlling
 annotation references 511
Character code conversion between browser and
 application 616
Character Codes 614
Character codes handled in application 614
check development log 447
checking connection pool state 247
checking running transaction 192
checking stopped transaction 192
checking transaction information 192
Checking versions during flush operation or transaction
 conclusion 340
Checking versions of persistence fields and relationships
 339
CJMSP Broker 385
CJMSP resource adapter 385
cjreloadapp command 570
classes to be loaded and class path required for loading 498
classes used for lifecycle management 199
classification of functionality 2
Class loader configuration for local call optimization 620
class loader configuration used for reloading 556
Class path set in class loader 621
clearing cache 75
clearing connection pool 187
client transaction operations when system exception occurs
 125
communication port for transaction recovery 252
communication port used by Smart Agent 252, 253
Complex type primary key 347
component-managed sign-on 177
Component-managed sign-on 127
Concurrent use of round-robin search functionality 79
Conditional expressions that can be used in WHERE clause
 372
conditions for connection association 172
conditions for connection sharing 171
configuration of application directory 531
configuration of applications that can be executed with
 Application Server 482
Configuration of Class Loader 618
configuring database connections using DB Connector for
 Cosminexus RM and Cosminexus RM 143
connectable databases 130, 146
Connectable databases 305
connecting to database 128
connecting to database queue 141
connecting to Oracle using Oracle RAC 234

connecting to SMTP server 154
 connection association 172, 178
 connection count adjustment functionality 168, 179
 connection definition identifier 217
 connection management threads 179, 196
 Connection pool clustering functionality 234
 connection pool clustering operations 241
 connection pooling 165
 connection pooling operations 167
 connection pool warming up 168, 179
 connection settings for OpenTP1 and Outbound 151
 connection settings for using components other than resource adapters 111
 connection sharing 170
 connection sharing or association 169
 connections using DB Connector for Cosminexus RM and Cosminexus RM 141
 connections using TP1/Message Queue - Access 151
 connections using uCosminexus TP1 Connector 150
 connection sweeper 189, 197
 connection sweeper operations 169
 connection test for resources 249
 connection test functionality 249
 connection with other resources 163
 constraints on using container extension library and server start and stop hook functionality 596
 Constructor expression 368
 consumer 385
 container extension library 585
 container extension library 587
 container extension library functionality 17, 588
 container-managed EntityManager 268
 container-managed sign-on 177
 Container-managed sign-on 127
 contents defined in DD (JavaBeans resources) 157
 content that can be specified in definition of JavaBeans resource properties 160
 Contract Between JPA Provider and EJB Container 623
 controlling annotation references 507
 CORBA Naming Service 28
 correspondence between functionality and manuals 5
 correspondence between purpose of system and functionality 8
 cosminexus.xml 469
 Cosminexus JMS Provider 385
 Cosminexus JMS Provider functionality 14
 Cosminexus JPA Provider 301
 Cosminexus JPA Provider 303
 Cosminexus JPA Provider functionality 13
 Cosminexus RM 94
 createEndpoint method 220
 Creating accessor method 345
 creating applications containing cosminexus.xml 472
 creating cosminexus.xml 472
 Creating entity class 343
 creating implementation class of JavaBeans resources 157

D

data access model when JPA is not used 261
 data access model when JPA is used 262
 Database operations based on query language 338

DB Connector-based connections 128
 DB Connector for Cosminexus RM 94
 dead message queue 401
 dead messages 401
 Declaring collection members 371
 Default class loader configuration 618
 Default mapping (bi-directional relationship) 321
 Default mapping (unidirectional relationship) 324
 Default mapping rules for persistence fields and persistence properties 349
 defining connection association 173
 defining connection sharing 171
 Defining mapping between entity class and database 343
 Defining persistence.xml 380
 definitions in DD (module settings) 510
 delayed reloading of Web applications 564
 deploying and un-deploying J2EE applications 540
 deploying and un-deploying J2EE applications in archive format 540
 deploying and un-deploying J2EE applications in exploded archive-format 540
 deploying and using resource adapter as J2EE resource adapter 95
 deploying EAR files and ZIP files in exploded archive format 541
 deploying message endpoint 214
 Deployment-related contract 627
 destination 385
 destinations created using commands 401
 destroying connections with exceptions 166
 detached 309
 detecting connection errors 181, 196
 detecting errors in naming service 78
 detecting updates and reloading J2EE applications 550
 determining usage of container extension library 588
 DI 501
 differences in functionality depending on presence of application.xml 483
 Differences in schema of resource adapters conforming to Connector 1.0 specifications and Connector 1.5 specifications 92
 Direction of relationships 320
 directly deploying resource adapter on J2EE server 107
 directory storing status file 125
 displaying connection pool information 186
 Display name specified when DD is omitted 489
 Duplication of optional names 57

E

EJB container-based auto-close connection 188
 ejbserver.client.transaction.clientName 255
 ejbserver.client.transaction.enabled 255
 ejbserver.connectionpool.applicationAuthentication.disabled 178
 ejbserver.connectionpool.association.enabled 178
 ejbserver.connectionpool.association.enabledDespiteUnshareableSetting 178
 ejbserver.connectionpool.sharingOutsideTransactionScope.enabled 178
 ejbserver.connectionpool.validation.timeout 125, 179, 196
 ejbserver.connector.statementpool.clear.backcompat 178

ejbserver.container.ejhome.sessionbean.reconnect.enabled
 84
 ejbserver.cui.optionalname.enabled 62, 63
 ejbserver.deploy.context.check_interval 571
 ejbserver.deploy.context.reload_scope 571
 ejbserver.deploy.context.update.interval 572
 ejbserver.deploy.session.work.directory 573
 ejbserver.distributedtx.ots.status.directory1 125, 256
 ejbserver.distributedtx.ots.status.directory2 125, 256
 ejbserver.distributedtx.recovery.completionCheckOnStopping.timeout 196
 ejbserver.distributedtx.recovery.port 196, 253, 255
 ejbserver.distributedtx.rollbackClientTxOnSystemException 125
 ejbserver.distributedtx.XATransaction.enabled 125
 ejbserver.jndi.cache 76
 ejbserver.jndi.cache.interval 76
 ejbserver.jndi.cache.interval.clear.option 76
 ejbserver.jndi.cache.reference 178
 ejbserver.jndi.namingservice.group.<Specify group name>.providerurls 69
 ejbserver.jndi.namingservice.group.list 69
 ejbserver.jndi.request.timeout 36
 ejbserver.jta.TransactionManager.defaultTimeOut 196, 256
 ejbserver.naming.host 36, 37
 ejbserver.naming.port 36, 37
 ejbserver.naming.protocol 37
 ejbserver.naming.startupMode 36
 ejbserver.webj2ee.connectionAutoClose.enabled 196
 elements specifiable in connection definitions and location of specification 218
 Embedded class 348
 enabling connection sharing outside Application Server-managed transactions 178
 entity classes 262
 Entity lifecycle management with EntityManager 354
 EntityManager 268
 EntityManager and persistence context 270
 Entity operations by EntityManager 309
 Entity relationships 319
 Estimating number of DB Connector connections 307
 example of creating application directory 532
 example of creating cosminexus.xml 474
 examples of property file specification 227
 examples of target files for update detection 558
 Exception processing when optimistic lock fails 340
 Exceptions thrown in API functions of Query interface 379
 Exceptions thrown in API functions of query-related interfaces in EntityManager 379
 Exceptions thrown when queries are used 379
 Excluding callback methods 358
 executability of commands based on connection pool status 245
 executable J2EE application formats 527
 Execution results of SELECT clause 369
 expanding EAR files and ZIP files 541
 exploded archive-format J2EE application 529
 exploded archive-format J2EE applications 529

F

features of connections using DB Connector for Cosminexus RM and Cosminexus RM 142
 features of JPA 261
 FetchType.EAGER 314
 FetchType.LAZY 314
 files for update detection 557
 for EJB applications 568
 formats and deployment of J2EE applications 525
 for Message-driven Beans and resource adapters using any message listener interface 230
 for Message-driven Beans and resource adapters using javax.jms.MessageListener interface 228
 for Web applications 569
 free files 405
 FROM clause 369
 Full 331
 functionality available for other resource connections 163
 functionality available with connection pooling 168
 functionality available with Inbound 164
 functionality available with Outbound 163
 functionality for controlling annotation references 507
 functionality for fault tolerance 181
 functionality for formatting and deploying J2EE applications 17
 functionality for invoking Application Server from OpenTP1 (TP1 inbound integrated functionality) 12
 functionality for managing resource connections and transactions 9
 functionality for operating and maintaining execution platform of applications 5
 functionality for operations in firewall environment 252
 functionality for performance tuning 165
 functionality other than resource adapter functionality 98
 Functionality provided by Cosminexus JPA Provider 304
 functionality provided with transaction services 117
 functionality serving as execution platform for applications 4
 Function expression 373

G

generating and initializing connection pool 166
 getInputStream method 617
 getParameter method 616
 getQueryString method 617
 getReader method 617
 global transactions 114
 Glossary 693
 GROUP BY clause 375

H

Hard reference 331
 HardWeak 332
 HAVING clause 375
 how to check JNDI name space 35
 how to connect to resources 90
 how to obtain application-managed EntityManager 279
 how to obtain container-managed EntityManager 274
 How to obtain Query object 359, 361

how to reload J2EE applications 550
 How to set up EntityManager and EntityManagerFactory 354
 how to set up resource adapters 99
 how to use -d option 157
 how to use resource adapters 95

I

implementation for connecting to resources 99
 Implementing callback methods 357
 implementing Enterprise Beans when annotation is specified 497
 implementing JavaBeans resources 156
 implementing lookup (JavaBeans resources) 157
 implementing server start and stop hook functionality 593
 Inbound 163
 inbound connection with OpenTP1 152
 including and using resource adapter in J2EE application 95
 Inheritance class types 351
 Inheritance mapping strategy 352
 inheriting session information when Web applications are reloaded 565
 in-process transaction service 176
 interval for updating J2EE application configuration file 562
 Invocation order in inheritance hierarchy 358
 invocation order of server start and stop hook processing 592
 invoking CORBA objects through Smart Agent 595
 isDeliveryTransacted method 221
 items checked when application is deployed 297

J

J2EE resource adapter 100
 J2EE resource adapters 95
 java.naming.factory.initial 69
 JavaBeans resource functionality 155
 JavaMail functionality 15
 javax.persistence.EntityManager interface 289
 javax.resource.spi.endpoint.MessageEndpointFactory interface 220
 javax.resource.spi.endpoint.MessageEndpoint interface 221
 JDBC specifications supported by DB Connector 131
 JMS application 386
 jms service 400
 JNDI 52, 53
 JOINED strategy 353
 Joins expression 371
 JPA functionality that can be used with Application Server 264
 JPA functionality used with Application Server 13
 JPA provider 262
 JPQL coding method 367
 JPQL syntax 367

K

key 514

L

LAZY fetch in @OneToOne and @ManyToOne 315
 Light transaction 176
 light transaction functionality 114
 local interface 55
 local transactions 114
 Location for specifying callback method 356
 logical connection 169
 logical naming service 66
 looking up Administered objects 216
 looking up with names beginning with HITACHI_EJB 31
 looking up with names using java:comp/env 30
 looking up with optional names assigned by user-specified name space functionality 31
 looking up with Portable Global JNDI names 30

M

Main functionality changes in Application Server 09-50 21
 managed 309
 managed entity 316
 managing application attributes 16
 Managing Application Attributes 465
 managing attributes 467
 managing EntityManager life cycle 272
 managing resource adapter lifecycle 198
 managing resource adapter work 202
 managing transactions 113
 Mapped superclass 351
 mapping and looking up JNDI Name Space 32
 mapping usage and transaction levels of resource adapters 164
 Mechanism for referencing Enterprise Beans and method of usage (ejb-ref) 32
 Mechanism for referencing resources and method of usage (resource-ref) 34
 merge processing of entity 315
 merits of using annotations and annotations that can be specified 495
 message inflow 211
 Message listener method 222
 Message Selector 396
 method of adding resource adapter in imported J2EE application 105
 method of importing J2EE application containing resource adapter into J2EE server 104
 method of injecting EntityManagerFactory in application 279
 method of injecting EntityManager in application 274
 method of looking up EntityManagerFactory from application 280
 method of looking up EntityManager from application 276
 method of obtaining resource references by using DI 99
 Method of using @IdClass 347
 Method of using embedded class 347
 Method signature rules of accessor method 345
 method specified with cosminexus.xml or other property files 504
 method specified with mappedName attribute of @Resource annotation 503

method specified with name attribute of `@Resource` annotation 502

methods provided with `EntityManager` 268

Methods used for character code conversion between browser and application 616

method that confirms execution of auto-close connection 188

minimum and maximum number of connections 179

N

naming management 25

Naming management 26

naming management functionality 8, 27

naming service 28

naming services 28

new 309

NONE 333

Non-entity class of entity inheritance hierarchy 351

notes and restrictions related to reloading 574

Notes on adding resources in J2EE applications when `application.xml` is omitted 491

Notes on API functions of `EntityManager` 354

notes on application development 254

notes on caching in naming 77

notes on character code conversion in system 137

notes on connecting to HiRDB 133

notes on connecting to Oracle 135

notes on connecting to SQL Server 137

Notes on creating query string using JavaScript 617

Notes on data source specification 380

notes on enabling timeout for detecting connection errors 184

Notes on executing query 366

notes on implementation of bean validation 464

notes on implementation of CORBA object invocation processing 595

Notes on including file in JSP 617

notes on inheriting session information 566

notes on JTA-based transaction implementation 124

Notes on multiple persistence contexts using cache 334

notes on naming service error detection functionality 82

notes on packaging of CORBA object invocation processing 595

Notes on path expressions 375

notes on performing round-robin search 72

notes on replacing J2EE applications by redeploying 548

notes on resource adapters 111

notes on re-using EJB home object references 84

notes on specifying `selectMethod` property of DB Connector 138

notes on starting transaction with EJB client applications 254

notes on system operations 256

notes on system setup 254

Notes on updating or deleting data in query 334

notes on using annotations 521

Notes on using cache functionality 334

Notes on using callback methods 357

notes on using connection pool 166

notes on using exploded archive format 538

notes on using JPA with Application Server 299

Notes on using JPQL 377

Notes on using JPQL in HiRDB 378

notes on using lifecycle management functionality 202

Notes on using literals 375

Notes on using optimistic lock 340

notes on using resource adapters conforming to Connector 1.5 specifications 233

notes on using statement pooling functionality 175

notes on using Synchronization 120

notes on using transaction manager 120

notes on using work management functionality 210

O

objects that can be assigned optional name 55

Obtaining and executing native query results 365

Obtaining and executing query results 361

Obtaining entities from database 312

obtaining transaction manager 119

omitting DD 482

OpenTP1 SPP 95

Operations and state transition for entities 309

operations during DI failure 505

Operations for creating `application.xml` when `application.xml` is omitted 490

Operations for the entities 309

Operations for Web applications in which `web.xml` is omitted 488

operations of applications containing `cosminexus.xml` 476

operations that are reloaded 560

operations when error occurs 557

operations when performing round-robin search 67

Optimistic lock 339

Optimistic lock processing 339

optimizing container-managed sign-on for DB Connector 177, 178

optional names for Enterprise Beans 55

optional names for J2EE resources 56

ORDER BY clause 376

Order of invoking callback methods 358

other resource adapter functionality (for resource adapters conforming to Connector 1.5 specifications) 198

Outbound 163

Output of information when `validation.xml` contents are invalid 463

output of the SQL statement for troubleshooting 193

overriding `@PersistenceUnit` definition using DD 282

Overview of Bean Validation 453

overview of connection pool clustering 236

overview of exploded archive format 529

overview of naming management 27

overview of processing and points to remember when transactions are not used 123

overview of processing and points to remember when using container-managed transactions (CMT) 120

overview of processing and points to remember when using resource adapter-specific transaction management interface 122

overview of processing and points to remember when using `UserTransaction` interface 121

overview of resource connections and transaction management 89

overwriting annotations using DD 514

P

Path expression 370
 persistence context when application-managed
 EntityManager is used 272
 persistence context when container-managed
 EntityManager is used 271
 persistence field 344
 persistence property 344
 persistence subscriber 392
 persistence unit 269
 persist operation for entities 311
 Pessimistic lock in JPQL 342
 physical connection 169
 points to remember when connection test is performed for
 resources 219
 points to remember when operation information and
 information for resource depletion monitoring is output
 220
 points to remember when transaction is recovered 219
 points to remember when using connection pool 219
 pool size of CallableStatement 180
 pool size of PreparedStatement 180
 Portable Global JNDI name 30
 precautions for using user-specified name space
 functionality 64
 preconditions and notes on connecting to HiRDB 132
 preconditions and notes on connecting to Oracle 134
 preconditions and notes on connecting to SQL Server 136
 preconditions and notes on connecting to XDM/RD E2 140
 preconditions for connecting to HiRDB queue 147
 preconditions for connecting to Oracle queue 148
 preconditions for HiRDB Version 8 132, 147
 preconditions for HiRDB Version 9 133, 148
 preconditions for Oracle11g 134
 preconditions for SQL Server 136
 Preconditions for using Cosminexus JPA Provider 305
 Primary key type 346
 Priority of operators 374
 priority of reference resolution methods 504
 procedure for changing resource adapter settings 101
 procedure for changing settings of JavaBeans resource 159
 Procedure for checking whether data is updated 339
 procedure for controlling message inflow (for Non-
 Transacted Delivery) 212
 procedure for controlling message inflow (for Transacted
 Delivery) 212
 Procedure for inheriting entity class 351
 Procedure for processing cache functionality 329
 Procedure for referencing and updating database with JPQL
 359
 Procedure for referencing and updating database with
 native query 361
 Procedure for referencing and updating database with query
 language 359
 procedure for replacing JavaBeans resource 160
 procedure for replacing resource adapters 102
 procedure for resource adapter settings (To deploy and use
 resource adapter as J2EE resource adapter) 100

procedure for resource adapter settings (To include and use
 resource adapter in J2EE applications) 103
 procedure for resource adapter settings (To use connection
 pool clustering) 109
 procedure for resource adapter settings (To use resource
 adapter with Inbound) 107
 procedure for setting up new JavaBeans resource 158
 procedure for setting up new resource adapter 100
 Procedure for specifying callback method 356
 procedure for starting JavaBeans resources 155
 procedure for stopping or starting connection pool manually
 246
 Procedure for updating cache 330
 procedure of caching 74
 procedure of creating and using container extension library
 589
 procedure of migrating from applications not containing
 cosminexus.xml 481
 procedures for using bean validation 459
 procedures for using bean validation from CDI 460
 procedures for using bean validation from JSF 459
 Processing in Cosminexus JPA Provider 303
 Processing of cache functionality 329
 processing of resource adapter for delivering messages 215
 producer 385
 Propagation of operations to entities 311
 PTP messaging model 390
 Pub/Sub messaging model 391
 publisher 391
 purpose and scope of functionality for controlling
 annotation references 507

Q

Query domain 367
 queue 390

R

Range variable declaration and identification variables 369
 Reading entity information from database 314
 receiver 390
 Redeploy 546
 redeploying J2EE applications 546
 reference scope of persistence unit name 296
 referencing and updating definitions with server
 management commands 519
 Relation between JPQL and cache 330
 relation between settings for client source and search
 destination object 59
 relationship between reloading and controlling concurrently
 executed threads 567
 relationship between reloading and monitoring J2EE
 application execution time 568
 Relationship types 319
 release method 222
 reload 570
 reloading by detecting updates 550
 reloading by using commands 553
 reloading J2EE applications using commands 570
 reloading JSPs 566
 reloading Web applications 564

- remote interface 55
 - removed 309
 - remove operation for entities 312
 - replacing J2EE applications 545
 - replacing J2EE applications by redeploying 546
 - replacing JavaBeans resources 161
 - Requirements for creating entity classes 343
 - resolving resource references using @Resource annotation 502
 - resource adapter functionality 96
 - resource adapter lifecycle management 198
 - resource adapters available in each usage method 95
 - resource adapters conforming to Connector 1.0 specifications 92
 - resource adapters conforming to Connector 1.5 specifications 92
 - resource adapters used 238
 - resource adapters used for connection with other resources 163
 - Resource adapter work management 202
 - resource connections 90
 - Resource Connections and Transaction Management 87
 - resource sign-on method 127
 - resources that do not use resource adapters for connection 91
 - resources that use resource adapters for connection 90
 - restarting connection pool 247
 - Result set mapping 363
 - retrying to obtain connection 185, 197
 - re-using EJB home object references (functionality for re-connecting to EJB home objects) 84
 - round-robin search 66
 - Rules applied to callback methods 357
 - rules for assigning optional names 56
 - rules for determining modules when application.xml does not exist 486
 - rules for determining modules when application.xml exists 485
 - Runtime-related contract 623
- ## S
-
- Scope of cache functionality 334
 - scope of reloading 555
 - scope of round-robin search 66
 - searching CORBA Naming Service by using round-robin policy 66
 - searching Enterprise Beans 59
 - searching from the client 59
 - searching J2EE resource 60
 - SELECT clause 368
 - SELECT statement 367
 - sender 390
 - Separate and merge operations of entity from persistence context 315
 - server start and stop hook functionality 592
 - server start and stop hook functionality 587
 - session information file 565
 - setCharacterEncoding method 616
 - setFirstResult method 365
 - Set function 368
 - setMaxResults method 365
 - setting interval for configuration file update 572
 - setting optional names for Enterprise Beans 60
 - setting optional names for J2EE resources 62
 - setting scope for reload functionality 571
 - settings for delayed reloading 573
 - settings for detecting updates and reloading J2EE applications 570
 - settings for JavaBeans resource property file 156
 - settings for JSP pre-compilation 574
 - settings for mapping Message-driven Beans and resource adapters 224
 - settings for monitoring J2EE application execution time 574
 - settings for operations in firewall environment 252
 - settings for using caching functionality 76
 - settings for using container extension library functionality 589
 - settings for using exploded archive-format J2EE applications (changing security settings) 537
 - settings for using resource adapters conforming to Connector 1.5 specifications 222
 - settings recommended for using round-robin search functionality 72
 - settings required for clustering connection pool 248
 - settings required for performing round-robin search 68
 - setting up Administered objects 224
 - setting up Administered objects to be looked up 216
 - setting update detection interval 571
 - setting up interfaces used by Message-driven Beans 225
 - setting up J2EE application properties 543
 - setting up JavaBeans resources 158
 - sharing application directory across multiple J2EE servers 530
 - SINGLE TABLE strategy 352
 - Soft reference 331
 - SoftWeak 332
 - Specifying access methods for entity class fields 344
 - specifying annotations 495
 - Specifying callback listener in O/R mapping file 357
 - Specifying callback method in annotation 356
 - specifying class path for using server start and stop hook functionality 594
 - Specifying flush mode 366
 - specifying multiple connection definitions 217
 - Specifying primary key in entities 346
 - Specifying query hint 366
 - Specifying range of query result items 365
 - SQL statement for troubleshooting 193
 - statement cancellation 189, 190, 197
 - statement initialization in statement pooling functionality 178
 - statement pooling 173
 - statement pooling operations 174
 - statuses and replacement of J2EE applications 548
 - status of member connection pools 245
 - subscriber 391
 - supported application formats 266
 - supported class loader configuration 266
 - suspending connection pool 247
 - switching CORBA Naming Services 83
 - Synchronization with database 313

T

target files for update detection 557
 temporary destinations created using APIs 401
 terminating connection pool 166
 thread pool 207
 thread pooling 207
 timeout during checking of unconcluded transactions 196
 Timeout in communication with naming service 36
 timeout in error detection 125, 182, 196
 Timeout in error detection 179
 timing at which error detection is implemented 182
 yiming for clearing cache 75
 timing for referencing annotations 509
 timing for registering or deleting optional name 58
 timing for registering or deleting optional names of J2EE resources 58
 Timing when entity information is read from database 314
 to change the security policy settings 538
 to include and use resource adapter in J2EE application 108
 topic 391
 to release SecurityManager 538
 to use DB Connector for Cosminexus RM and Cosminexus RM 94
 TPI/Message Queue 95
 transaction control and EntityManager 269
 transaction inflow 215
 transaction management methods for resource connections 113
 transaction operations during system exceptions 118
 transaction recovery 190, 196, 253
 transaction support level 126
 transaction support levels 115
 transaction timeout 189
 transaction timeout (for J2EE servers) 196
 Transaction timeout settings 189
 transaction types 115, 125
 transition of member connection pool state (when pool is suspended automatically) 244
 transition of member connection pool state (when pool is suspended manually) 244
 types of DB Connector for Cosminexus RM (RAR file) 147
 types of DB Connectors (RAR file) 132
 types of EntityManager 268
 Types of entity states 309
 types of names used for lookup 29
 types of persistence context 270
 Types of persistence fields and persistence properties of entities 345
 types of RAR files for each resource adapter 93
 types of resource adapters 91
 types of resources that can be specified in @Resource annotation 501

U

undeploying message endpoint 214
 Unidirectional ManyToMany relationship 327
 Unidirectional ManyToOne relationship 325
 Unidirectional Multi-Valued relationship 326
 Unidirectional OneToMany relationship 326
 Unidirectional OneToOne relationship 324

Unidirectional Single-Valued relationship 324
 update detection interval for J2EE applications 561
 updating annotations 512
 updating contents defined in annotations 512
 Updating database using entities 308
 user-managed transactions (Transactions not managed by Application Server) 114
 user-specified name space functionality 55
 user-specified name space functionality 31
 User-specified name space functionality 63
 using annotations 493
 using container extension library 587
 Using debug log (check development log) 462
 using DI 501
 using JavaBeans resources 155
 using library JAR class with declared annotation 496
 USRCONF_JVM_CLASSPATH 590
 USRCONF_JVM_CLPATH 590
 USRCONF_JVM_LIBPATH 590

V

vbroker.agent.port 253

W

waiting for connection when connections deplete 184, 197
 waiting time until database connection is established 140
 Weak 333
 Weak reference 331
 web container-based auto-close connection 188
 webserver.context.check_interval 571
 webserver.context.reload_delay_timeout 573
 webserver.context.update.interval 572
 webserver.jsp.check_interval 571
 webserver.jsp.update.interval 572
 What is naming service error detection functionality 78
 when concurrently executed threads are controlled for Web applications or URL groups 567
 when concurrently executed threads are controlled for Web containers 568
 when different databases are accessed with JMS interface and JDBC interface 144
 when JMS interface is used alone 144
 when running J2EE application is replaced 548
 when same database is accessed with JMS interface and JDBC interface 144
 when stopped J2EE application is replaced 548
 WHERE clause 372
 work management start and termination processing 210
 Writing entity information to database 313