HITACHI
Inspire the Next

uCosminexus Application Server

EJB Container Functionality Guide

3020-3-Y06-10(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

■ 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

Abbreviation			Full name or meaning
SQL Server	SQL Server 2005		Microsoft (R) SQL Server
	SQL Server 2008		Microsoft (R) SQL Server 2008
			Microsoft(R) SQL Server 2008 R2
	SQL Server 2012		Microsoft(R) SQL Server 2012
JDBC driver of 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 Enterprise 32-bit
			Microsoft (R) Windows Server (R) 2008 Standard 32-bit
		Windows Server 2008 x64	Microsoft (R) Windows Server (R) 2008 Enterprise
			Microsoft (R) Windows Server (R) 2008 Standard
		Windows Server 2008 R2	Microsoft (R) Windows Server (R) 2008 R2 Enterprise
			Microsoft (R) Windows Server (R) 2008 R2 Standard
			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 (32bit)
		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.

Printed in Japan.

■ Issued

Aug. 2013: 3020-3-Y06-10(E)

■ Copyright

Summary of amendments

The following table lists changes in the manual 3020-3-Y06-10(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, and uCosminexus Service Platform(64) 09-50 and product changes related to the manual:

Additions and Changes	Change Location
Precautions to be taken while executing multiple threads when you invoke a remote interface of EJB were added.	*2.13.6*
You can now compare the requests at the invocation source and the invocation destination by using the root application information of the PRF trace.	*2.17.3*
Notes on the remote interface for annotations while implementing asynchronous methods were added.	*2.17.8*
Precautions to be taken when you start the J2EE sever by specifying the `-nosecurity` option in the `cjstartsv` command were added.	*3.7.3*
Notes on multi-byte characters in the classes, which configure EJB were added.	*4.2.15*
Precautions to be taken when acquiring a system exception with the `getCause()` method were added.	*4.2.21*
The description on notes was moved from the release notes.	*2.17.9*, *4.2.15*, *4.2.17*, *4.2.21*, *4.2.22* and *4.2.23*

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

Preface

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

■ Non-supported functionality

Some functionality described in this manual are not supported. The 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 the 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 (cjjsp2java) with the 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 the EJB 2.1 and Servlet 2.4 annotations

Contents

1 Application Server Functionality

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

1.1 Classification of functionality

Application Server is a product used for building an environment to execute applications mainly on a J2EE server that is compliant with Java EE 6 and to develop applications that run in the execution environment. You can use the variety of functionality, such as the functionality compliant with Java EE standard specifications and the functionality independently expanded on Application Server. By selecting and using the functionality according to the purpose and intended use, you can build and operate a highly reliable system with excellent processing performance.

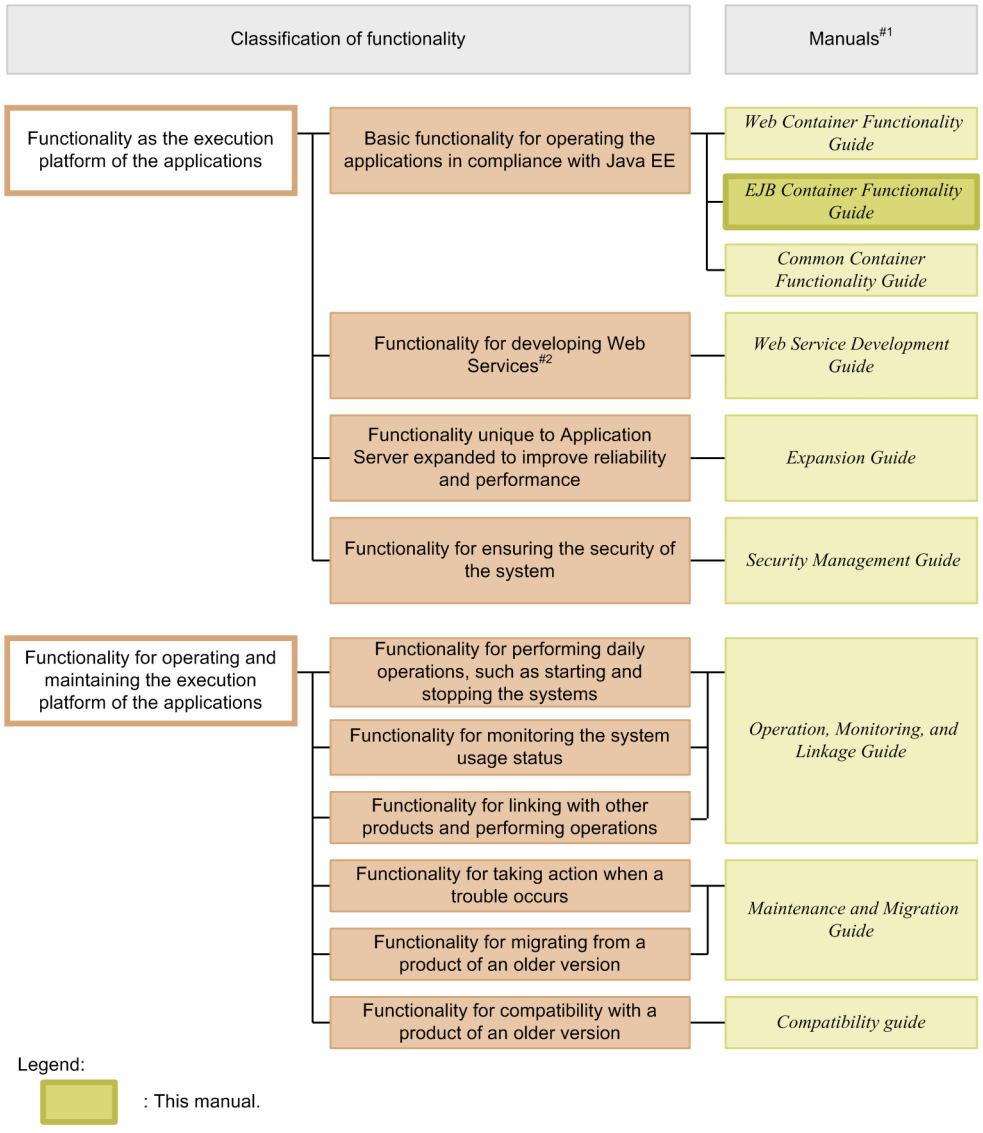
The broad classifications of Application Server functionality are as follows:

- Functionality that serves as an execution platform for the applications
- Functionality that is used for operating and maintaining an execution platform for the applications

The above-mentioned functionality can be further classified according to the positioning and the intended usage of the functionality. Application Server manuals are provided according to the classifications of the functionality.

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

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



#1

uCosminexus Application Server is omitted from the manual names.

#2

With Application Server, you can execute SOAP Web Services and RESTful Web Services. Depending on the objective of usage, see the following manuals other than the *uCosminexus Application Server Web Service Development Guide*.

When developing and executing SOAP applications

- *uCosminexus Application Server SOAP Application Development Guide*

When ensuring the security of SOAP Web Services and SOAP applications

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

For details about the XML processing

- *uCosminexus Application Server XML Processor User Guide*

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

1.1.1 Functionality that serves as a platform for executing applications

This functionality works as a platform for executing online business and batch business implemented as applications. You select the functionality that you want to use according to the intended use of the system and your requirements.

You must determine whether you want to use the functionality that serves as a platform for executing the applications before building the system or developing the applications.

The classification-wise descriptions of the functionality that serves as the platform for executing applications are as follows:

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

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

Application Server provides a Java EE 6 compliant J2EE server. The J2EE server provides functionality that is compliant with the standard specifications as well as functionality that is unique to Application Server.

The basic development functionality can be further classified into three types according to the types of the J2EE applications for which you use the functionality. The Application Server functionality guide manuals have been arranged according to this classification.

The following is an overview of each classification:

- **Functionality for executing Web applications (Web containers)**

  This classification includes the Web container functionality that serves as an execution platform for Web applications and the functionality executed by linking Web containers and Web servers.
- **Functionality for executing Enterprise Beans (EJB containers)**

  This classification includes the EJB container functionality that serves as a platform for executing Enterprise Beans. This classification also includes the EJB client functionality for invoking Enterprise Beans.
- **Functionality used in both Web applications and Enterprise Beans (Container common functionality)**

  This classification includes the functionality that can be used in the Web applications and the Enterprise Beans that run Web containers and EJB containers respectively.

(2) Functionality to develop Web Services

This classification corresponds to the functionality that serves as an execution environment and a development environment for Web Services.

Application Server provides the following engines:

- *JAX-WS engine*, which binds SOAP Messages according to the JAX-WS specifications
- *JAX-RS engine*, which binds RESTful HTTP messages according to the JAX-RS specifications

(3) Application Server unique functionality expanded for improving reliability and performance (Expansion functionality)

This classification includes functionality that is expanded uniquely for Application Server unique . This classification also includes the functionality executed by using non-J2EE server processes such as a batch server, CTM, or database.

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

(4) Functionality for ensuring the system security (Security management functionality)

This classification includes the functionality for maintaining security of a system centered around Application Server. This classification comprises functionalities such as authentication functionality for preventing access by unauthorized users and encryption functionality for preventing information leakage in the communication channel.

1.1.2 Functionality for operating and maintaining a platform for executing applications

This functionality is used for effectively operating and maintaining a platform for executing applications. You use this functionality, after starting the system operations, as and when required. However, depending on the functionality, you might have to implement the settings and applications in advance.

The classification-wise descriptions of the functionality used for operating and maintaining the platform for executing applications are as follows:

(1) Functionality used for daily operations, such as starting and stopping the systems (operation functionality)

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

(2) Functionality for monitoring the system usage (watch functionality)

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

(3) Functionality for operating the system by linking with other products (linkage functionality)

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

(4) Functionality for troubleshooting (maintenance functionality)

This classification includes the functionality used for troubleshooting. This classification also includes the functionality used for displaying information that will be referenced during the troubleshooting.

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

This classification includes the functionality used for migrating a system from an older Application Server to a new Application Server.

(6) Functionality for compatibility with products of older version (compatibility functionality)

This classification includes the functionality used for the compatibility with the older versions of Application Server. For the compatibility functionality, we recommend that you migrate a system using the corresponding recommended functionality.

1.1.3 Correspondence between functionality and manuals

The functionality guides for Application Server have been arranged according to the classifications of the functionality.

The following table describes the classifications of the functionality and the manual corresponding to each functionality.

Table 1-1: Classification of functionality and the manuals corresponding to each functionality

Category	Functionality	Manual[#1]
Basic development functionality	Web container	*Web Container Functionality Guide*
	Using JSF and JSTL	
	Web server linkage	
	In-process HTTP server	
	Servlet and JSP implementation	
	EJB container	*EJB Container Functionality Guide* [#2]
	EJB client	
	Precautions during Enterprise Bean implementation	
	Naming management	*Common Container Functionality Guide*
	Resource connection and transaction management	
	Invoking Application Server from OpenTP1 (TP1 in-bound linkage functionality)	
	JPA usage on Application Server	
	Cosminexus JPA provider	
	Cosminexus JMS provider	
	Using JavaMail	
	Using CDI on Application Server	
	Using Bean Validation with Application Server	
	Application property management	
	Using annotations	
	Formatting and deploying J2EE applications	
	Container extension library	
Extended functionality	Executing applications using a batch server	*Expansion Guide*
	Scheduling and load balancing requests using CTM	
	Scheduling batch applications	
	Inheriting the session information between the J2EE servers (Session failover functionality)	
	Database session failover functionality	

Category	Functionality	Manual[#1]
Extended functionality	EADs session failover functionality	*Expansion Guide*
	Controlling full garbage collection using the explicit heap functionality	
	Output of application user logs	
	Asynchronous parallel processing of threads	
Security management functionality	Authentication by integrated user management	*Security Management Guide*
	Authentication by application settings	
	Using TLSv1.2 in SSL/TLS communication	
	Controlling with the operation management functionality of the load balancer that is directly connected to API	
Operation functionality	Starting and stopping the system	*Operation, Monitoring, and Linkage Guide*
	Managing J2EE applications	
Watch functionality	Monitoring the operation information (Statistics collection functionality)	
	Monitoring resource depletion	
	Database audit trail linkage functionality	
	Output of operation information using the management commands	
	Automatic execution of processing by using management event report and management action	
	Collecting statistical information of CTM	
	Output of the console log	
Linkage functionality	Operating a JP1 integrated system	
	Centralized monitoring of the system (Integrating with JP1/IM)	
	Automatic operation of system according to job (Integrating with JP1/AJS)	
	Audit log collection and unitary management (Integrating with JP1/ Audit Management - Manager)	
	1-to-1 node switching system (linking with cluster software)	
	Mutual node switching system (linking with cluster software)	
	Node switching system for host unit management models (integrating with cluster software)	
Maintenance functionality	Troubleshooting related functionality	*Maintenance and Migration Guide*
	Analyzing the performance using performance analysis trace	
	Hitachi-specific JavaVM (hereafter, it is also abbreviated as JavaVM) functionality	
Migration functionality	Migrating from an older version of Application Server	
	Migrating to a recommended functionality	
Compatibility functionality	Basic mode	*Compatibility Guide*
	Servlet engine mode	
	Compatibility functionality with the basic and development functionality	
	Compatibility functionality with the extended functionality	

#1

uCosminexus Application Server is omitted from the manual names mentioned in the *Manual* column.

#2

This manual.

1.2 Functionality corresponding to the purpose of the system

With Application Server, you must select the applicable functionality according to the purpose of the system to be built and operated.

This subsection describes each functionality for executing Enterprise Beans and describes which functionality is best used for which system. The functionality-wise support is described for the following:

- **Reliability**
  This functionality is best used for a system from which high reliability is recommended.
  This functionality includes functionality for enhancing the system availability (stable operations) and fault tolerance, and functionality for enhancing the security such as user authentication.
- **Performance**
  This functionality is best used for a system that adds value to performance.
  This functionality is used for performance tuning of the system.
- **Operation and maintenance**
  This functionality is best used when efficient operation and maintenance is to be performed.
- **Extendibility**
  This functionality is best used when a system is to be flexibly expanded or reduced, and when the system configuration is to be changed.
- **Others**
  The functionality is used to comply with other individual purposes.

The functionality for executing Enterprise Beans includes the Java EE standard functionality and the functionality uniquely expanded on Application Server. When you select the functionality, also confirm the compliance with the Java EE standard, as and when required.

1.2.1 EJB container functionality

The following table describes the functionality of an EJB container. Select the functionality according to the purpose of the system. For details on the functionality, see the *Reference location* column.

Table 1-2: Correspondence between the EJB container functionality and the purpose of the system

Functionality	Purpose of the system					Compliance with the Java EE Standard		Reference location
	Reliability	Performance	Operation and Maintenance	Extendibility	Others	Standard	Extended	
Executing an Enterprise Bean	--	--	--	--	--	Y	Y	*2.2*
Checking the compliance with EJB specifications	--	--	--	--	--	Y	Y	*2.3*
Mapping of CMP field and data types	--	--	--	--	--	Y	Y	*2.4*
Registering a reference in the JNDI name space of the EJB container[#]	--	--	--	Y	--	Y	Y	*2.5*
Connecting to an external resource	--	--	--	Y	--	Y	--	*2.6*
Transaction management in an Enterprise Bean	--	--	--	--	--	Y	Y	*2.7*

Functionality	Purpose of the system					Compliance with the Java EE Standard		Reference location
	Reliability	Performance	Operation and Maintenance	Extendibility	Others	Standard	Extended	
Cache models of an Entity Bean (Specifying the commit option)	--	Y	--	--	--	Y	--	*2.8*
Managing the Pool of Stateless Session Bean, Entity Bean	--	Y	--	--	--	Y	Y	*2.9*
Controlling the access to the Enterprise Beans	Y	--	--	--	--	Y	--	*2.10*
Setting up a timeout in the EJB container	--	Y	--	--	--	Y	Y	*2.11*
Timer Service functionality	--	--	--	--	--	Y	Y	*2.12*
Invoking the remote interface of EJB	--	Y	--	--	--	Y	Y	*2.13*
Fixing the communication ports and IP address of the EJB container (TPBroker options)	Y	--	--	--	--	--	Y	*2.14*
Using the default interceptor	--	--	--	--	--	Y	Y	*2.15*
Omitting the local business interface (using No-Interface view)	--	--	--	--	--	Y	--	*2.16*
Asynchronous invocation of a Session Bean	--	--	--	--	--	Y	--	*2.17*
Specifying in the annotation of the Session Synchronization	--	--	--	--	--	Y	--	*2.18*
Using a Singleton Session Bean	--	--	--	--	--	Y	--	*2.19*

Legend:

Y: Supported

--: Not supported

Note:

The functionality for which `Y` is specified in both the *Standard* and *Extended* columns of the *Compliance with the Java EE Standard* column indicates that functionality unique to Application Server has been added to extend the functionality beyond the Java EE standard functionality. The functionality for which `Y` is specified only in the Extended column indicates functionality unique to Application Server.

#

This is implemented by using naming management.

1.2.2 Functionality of EJB Client

The following table describes the functionality of EJB clients. Select the functionality according to the purpose of the system. For details on the functionality, see the *Reference location* column.

Table 1-3: EJB client functionality and the purpose of the system

Functionality	Purpose of the system					Compliance with the Java EE Standard		Reference location
	Reliability	Performance	Operation and maintenance	Extendibility	Others	Standard	Extended	
Starting EJB client applications	--	--	--	Y	--	Y	Y	*3.3*
Invoking an Enterprise Bean	--	--	--	Y	--	Y	Y	*3.4*
Implementing a transaction in an EJB client application	--	--	--	Y	--	Y	Y	*3.5*
Implementing security in an EJB client application	--	--	--	Y	--	Y	Y	*3.6*
Acquiring RMI-IIOP stubs and interfaces	--	--	--	Y	--	Y	Y	*3.7*
System log output of an EJB client application	--	--	--	Y	--	Y	Y	*3.8*

Legend:

Y: Supported

--: Not supported

Note:

The functionality for which `Y` is specified in both the *Standard* and *Extended* columns of the *Compliance with the Java EE Standard* column indicates that functionality unique to Application Server has been added to expand the functionality beyond the Java EE standard functionality. The functionality for which `Y` is specified only in the Extended column indicates functionality unique to Application Server.

1.3 Explanation of the functionality described in this manual

This section describes the meaning of the classification that is used for describing the functionality in this manual and also provides an example table describing the classification.

1.3.1 Meaning of the classification

In this manual, each functionality is described by divided into the following five classifications. Depending on the purpose for referencing the manual, you can select and read the required section.

- Description

  This part describes the functionality. This describes the purpose, characteristics, and mechanism of the functionality. Read the description, if you want to understand an overview of the functionality.
- Implementation

  This part describes how to perform coding and how to describe a DD. Read this, when you develop applications.
- Setup

  This part describes how to set up the property required for building systems. Read this, when you build a system.
- Operation

  This part describes how to perform operations. This part describes the procedure for performing operations and also the execution examples of the commands to be used. Read this, when you operate the system.
- Notes

  This part describes the overall precautions to be taken when using the functionality. Make sure to read the descriptions of the precautions to be taken.

1.3.2 Example table describing the classification

The classification of the functionality is described in a table. The title of the table is *Organization of this chapter* or *Organization of this section*.

The following is an example table describing the classification of the functionality:

Example table describing the classification of the functionality

Table X-1 Organization of this chapter (xx functionality)

Category	Title	Reference location
Description	What is the `xx` functionality?	*X.1*
Implementation	Implementing an application	*X.2*
	Defining in the DD and `cosminexus.xml`#	*X.3*
Setup	Setting up in the execution environment	*X.4*
Operation	Performing the operation using the `xx` functionality	*X.5*
Notes	Precautions when using the `xx` functionality	*X.6*

#

For details on `cosminexus.xml`, see *11. Managing Application Properties* in the *uCosminexus Application Server Common Container Functionality Guide*.

Tip

Setting up the property of an application that does not include `cosminexus.xml`

For an application that does not include `cosminexus.xml`, set up or change the property after importing the application into the execution environment. You can also change the already specified property in the execution environment.

An application is set up in the execution environment using the server management commands and property files. For details on how to set up applications by using the server management commands and property files, see *3.5.2 Procedure for setting up J2EE application properties* in the *uCosminexus Application Server Application Setup Guide*.

The tags specified in the property files correspond to either DD or `cosminexus.xml`. For details on the correspondence between DD or `cosminexus.xml` and the tags of the property files, see *3. Property Files used for Setting up J2EE Applications* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

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

1.4 Main updates in the functionality of Application Server 09-50

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

The following contents are described in this section:

- This section gives an overview of the main updates in the functionality of Application Server 09-50. For details on the functionality, check the description in the references. *Reference manual* and *Reference location* columns describe the main features of a particular functionality.
- *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 have been changed for improving development productivity.

Table 1–4: Changes made for improving development productivity

Item	Overview of changes	Reference manual	Reference location
Simplifying Eclipse setup	Enabled the setup of the Eclipse environment by using GUI.	*Application Development Guide*	*1.1.5*, *2.4*
Debugging support using the user extended performance analysis trace	Enabled the creation of the user extended performance analysis trace configuration file in a development environment.	*Application Development Guide*	*1.1.3*, *6.5*

(2) Simplifying implementation and setup

The following table describes the items that have been changed to simplify implementation and setup:

Table 1–5: Changes made for simplifying implementation and setup

Item	Overview of changes	Reference manual	Reference location
Expanding the system configuration pattern in a virtual environment	The types of tier (http-tier, j2ee-tier and ctm-tier) that can be used in a virtual environment have increased. This enables the set up of the following system configuration patterns: • A pattern in which the Web server and J2EE server are placed on separate hosts • A pattern in which the front-end (Servlet, JSP) and back-end (EJB) are deployed separately • A pattern in which CTM is used	*Virtual System Setup and Operation Guide*	*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 1–6: Changes made for supporting the standard and existing functionality

Item	Overview of changes	Reference manual	Reference location
Supporting the JDBC 4.0 specifications	DB Connector now supports the HiRDB Type4 JDBC Driver with JDBC 4.0 specifications and the JDBC Driver of SQL Server.	*Common Container Functionality Guide*	*3.6.3*

Item	Overview of changes	Reference manual	Reference location
Modifying naming conventions in the Portable Global JNDI names	Added characters that can be used in the Portable Global JNDI names.	*Common Container Functionality Guide*	*2.4.3*
Supporting the Servlet 3.0 specifications	Updates of the HTTP Cookie name of Servlet 3.0 and the path parameter name of URL can now be used even in versions earlier than Servlet 2.5	*Web Container Functionality Guide*	*2.7*
Expanding the applicability of an application which can be integrated with Bean Validation	Enabled validation using Bean Validation with CDI and user applications.	*Common Container Functionality Guide*	*Chapter 10*
Supporting JavaMail	Enabled the use of the mail sending and receiving functionality that uses APIs, and is compliant with JavaMail 1.4.	*Common Container Functionality Guide*	*Chapter 8*
Expanding the applicability of OSs on which you can use the `javacore` command	Enabled acquisition of a thread dump of Windows by using the `javacore` command	*Command Reference Guide*	*javacore (acquiring thread dump / in Windows)*

(4) Maintaining and improving reliability

The following table describes the items that have been changed for maintaining and improving reliability:

Table 1–7: Changes made for maintaining and improving reliability

Item	Overview of changes	Reference manual	Reference location
Preventing exhaustion of the code cache area	Enabled prevention of area exhaustion by checking the size of the code cache area used in the system and changing the threshold value before the area is exhausted.	*System Design Guide*	*7.1.2*
		Maintenance and Migration Guide	*5.7.2*, *5.7.3*
		Definition Reference Guide	*16.1*, *16.2*, *16.4*
Supporting an effective application of the explicit heap functionality	Added functionality that can control the objects to be moved to the Explicit heap as functionality to shorten the automatic release processing time and efficiently apply the explicit heap functionality. • Functionality that controls the object movement to the Explicit memory blocks • Functionality for specifying classes for which application of the explicit heap functionality is to be excluded • Output of the object release rate information to the Explicit heap information	*System Design Guide*	*7.13.6*
		Expansion Guide	*8.2.2*, *8.6.5*, *8.10*, *8.13.1*, *8.13.3*
		Maintenance and Migration Guide	*5.5*
Expanding the output range of the class-wise statistical information	Enabled the output of reference relations based on the static field to the extended thread dump, that includes the class-wise statistical information.	*Maintenance and Migration Guide*	*9.6*

(5) Maintaining and improving the operation performance

The following table describes the items that have been changed for maintaining and improving operation performance:

Table 1-8: Changes made for maintaining and improving operation performance

Item	Overview of changes	Reference manual	Reference location
Supporting the EADs session failover functionality	Supported the EADs session failover functionality, which implements the session failover functionality by integrating with EADs.	*Expansion Guide*	*Chapter 5*, *Chapter 7*
Operation using WAR	Enabled deploying of the WAR application that consists of only the WAR files, on the J2EE server.	*Web Container Functionality Guide*	*2.2.1*
		Common Container Functionality Guide	*13.9*
		Command Reference Guide	*cjimportwar (Importing WAR application)*
Starting and stopping by synchronous execution of the operation management functionality	Added an option that executes the synchronous starting and stopping of the operation management functionality (Management Server and Administration Agent).	*Operation, Monitoring, and Linkage Guide*	*2.6.1*, *2.6.2*, *2.6.3*, *2.6.4*
		Command Reference Guide	*adminagentctl (start or stop Administration Agent)*, *mngautorun (set up/ canceling the set up of autostart and autorestart)*, *mngsvrctl (start, stop, or set up Management Server)*
Forcefully releasing the Explicit memory blocks in the explicit management heap functionality	Enabled the execution of the process of releasing the Explicit memory blocks with the `javagc` command at any timing.	*Expansion Guide*	*8.6.1*, *8.9*
		Command Reference Guide	*javagc (Force generation of garbage collection)*

(6) Other purposes

The following table describes the items that have been changed for other purposes:

Table 1-9: Changes made for other purposes

Item	Overview of changes	Reference manual	Reference location
Acquiring the definition information	Enabled collection of only the definition file with the `snapshot` (snapshot log collection) command.	*Maintenance and Migration Guide*	*2.3*
		Command Reference Guide	*snapshotlog (collecting snapshot log)*
Log output of the `cjenvsetup` command	Enabled output of the execution information of the setup (`cjenvsetup` command) of Component Container Administrator to the message log.	*System Setup and Operation Guide*	*4.1.4*
		Maintenance and Migration Guide	*4.20*

Item	Overview of changes	Reference manual	Reference location
Log output of the `cjenvsetup` command	Enabled output of the execution information of the setup (`cjenvsetup` command) of Component Container Administrator to the message log.	*Command Reference Guide*	*cjenvsetup (setup of Component Container Administrator)*
Supporting BIG-IP v11	Added BIG-IP v11 to the types of the available load balancers	*System Setup and Operation Guide*	*4.7.2*
		Virtual System Setup and Operation Guide	*2.1*
Performing the output of the CPU time to the event log of the explicit heap functionality	Enabled the output of the CPU time taken for the Explicit memory block release processing, to the event log of the explicit heap functionality.	*Maintenance and Migration Guide*	*5.11.3*
Extending the user extended performance analysis trace functionality	Added the following functionality to the user extended performance analysis trace: • Enabled the specification of the trace target in a package unit or class unit in addition to the usual method unit. • Extended the range of the available event IDs. • Released the restriction on the number of rows that can be specified in the user extended performance analysis trace configuration file. • Enabled the specification of the trace acquisition level in the user extended performance analysis trace configuration file.	*Maintenance and Migration Guide*	*7.5.2*, *7.5.3*, *8.28.1*
Improving the information analysis when using an asynchronous invocation of Session Bean	Enabled the comparison of the requests of the invocation source and invocation destination by using the root application information of the PRF trace.	This manual	*2.17.3*

2 EJB Container

This chapter explains the functionality that can be used in an EJB container that is the execution base of the Enterprise Beans. You use the EJB container functionality during the execution of J2EE applications that use Enterprise Beans.

Note that among the EJB container functionality, the JNDI name space functionality and the functionality for Enterprise Bean transaction setting and access control, use the J2EE service functionality. Also, see *3. Resource Connection and Transaction Management* in the *uCosminexus Application Server Common Container Functionality Guide*.

2.1 Organization of this chapter

The EJB container is an execution environment that controls the execution of Enterprise Beans and provides various services to the Enterprise Beans.

The following table describes the functionality and reference locations for the EJB container.

Table 2-1: Functionality of EJB container and the section describing each functionality

Functionality	Reference location
Executing an Enterprise Bean	*2.2*
Checking the compliance with EJB specifications	*2.3*
Mapping of CMP fields and data types	*2.4*
Registering a reference in the JNDI name space of the EJB container#1	*2.5*
Connecting to an external resource	*2.6*
Transaction management in an Enterprise Bean#2	*2.7*
Cache models of an Entity Bean	*2.8*
Managing the Enterprise Bean pool	*2.9*
Controlling the access to the Enterprise Beans	*2.10*
Setting up a timeout in the EJB container	*2.11*
Timer Service functionality	*2.12*
Invoking the remote interface of EJB	*2.13*
Fixing the communication ports and IP address of the EJB container (TPBroker options)	*2.14*
Using the interceptor	*2.15*
Omitting the local business interface (using No-Interface view)	*2.16*
Asynchronous invocation of a Session Bean	*2.17*
Specifying in Session Synchronization annotation	*2.18*
Using a Singleton Session Bean	*2.19*

#1

This functionality is implemented based on the use of the naming management functionality of the J2EE service. For an overview of the management functionality, see *2. Naming Management* in the *uCosminexus Application Server Common Container Functionality Guide*.

#2

This functionality is implemented based on the use of the transaction management functionality of the J2EE service. For an overview of the transaction management of J2EE services, see *3. Resource Connection and Transaction Management* in the *uCosminexus Application Server Common Container Functionality Guide*.

Note that the EJB container functionality provided by Application Server includes Application Server-specific functionality extended beyond the functionality provided by J2EE and Application Server-specific functionality. To determine whether the functionality is unique to Application Server, see *1. Application Server Functionality*.

For details on the functionality that you can use in EJB clients, see *3. EJB Client*.

2.2 Executing an Enterprise Bean

This section describes the functionality used for executing Enterprise Beans.

An Enterprise Bean is a program in which the business logic is created based on the EJB architecture. It is applicable to business-processing programs. An Enterprise Bean runs in an EJB container.

The following table describes the organization of this section:

Table 2-2: Organization of this section (Executing an Enterprise Bean)

Category	Title	Reference location
Description	Types of Enterprise Bean	*2.2.1*
	Interface for Enterprise Bean	*2.2.2*
	Lifecycle of Enterprise Beans	*2.2.3*

Note:

There is no specific description of *Implementation*, *Setup*, *Operation*, and *Notes* for this functionality.

2.2.1 Types of Enterprise Bean

The following table describes the Enterprise Beans that you can execute in the EJB container provided with Application Server:

Table 2-3: Classification of Enterprise Bean that can be executed in EJB container

Major classification	Minor classification
Session Bean	Stateless Session Bean
	Stateful Session Bean
	Singleton Session Bean
Entity Bean	BMP (Bean Managed Persistence)
	CMP (Container-Managed Persistence)
Message-driven Bean	None

The characteristics of Enterprise Beans are as follows:

(1) Session Bean

A Session Bean is an Enterprise Bean that is generated for each session from the client and expires when the client exits. The lifecycle of the Session Bean does not exceed the range from the beginning until the end of the usage of the system by the user. Session Beans are classified into Stateless Session Beans, Stateful Session Beans, and Singleton Session Beans.

(a) Stateless Session Bean

This is a model in which the state of the session is not managed. Each session from the client needs to be concluded in one invocation of the Bean business method.

(b) Stateful Session Bean

This is a model in which the state of the session is managed. The EJB container manages the state of the session. Even when one session from the client invokes multiple EJB business methods, the state of each session is saved in between the invocation of the business methods.

(c) Singleton Session Bean

This is a model in which the state of the session is shared among multiple clients. One instance which is shared among all the sessions is created for an application. You must determine the Bean lifecycle according to the application.

For details on Singleton Session Beans that you can use with Application Server, see *2.19 Using Singleton Session Beans*.

(2) Entity Bean

An Entity Bean expresses the entity, and as a prerequisite, must be stored (persisted) in the database. As a result, even when the client exits, the state of the Entity Bean continues to exist in the database. The lifecycle of this Enterprise Bean is longer as compared to that of a Session Bean. The following two management models are defined in the EJB specifications:

(a) BMP (Bean Managed Persistence)

This is a model for managing the data persistence of Enterprise Bean business methods. The developer of the Enterprise Bean must implement processes such as connecting to the database, assembling and executing SQL statements.

(b) CMP (Container-Managed Persistence)

This is a model in which the EJB container manages the data persistence. The EJB container executes processes, such as connecting to the database and storing data, therefore, these processes need not be executed by the business methods of the Enterprise Bean. Use the method provided by the EJB container to define the mapping of the Enterprise Bean data and the tables and columns of the database in which the data is to be stored. At the same time, define the connection information, such as the host name and port number of the database to which you will connect, in a resource adapter or a data source. The EJB container references this definition information, assembles the SQL statements, and references and stores the data in the tables of the database to which you will connect.

Note that since EJB QL has been implemented in CMP 2.0 added in EJB 2.0, the process of searching the database can be coded in the DD with syntax such as SQL, independent of the database to be used. In addition, CMR (Container-Managed Relationship) has been implemented to establish a relationship between the Entity Beans, therefore, the relationship between the Entity Beans can be specified in the DD and managed by the EJB container.

For details on the Java data type of the Entity Bean of the CMP functionality in the EJB container provided by Application Server and the SQL data type of the database, see *2.4.2 Mapping the CMP field and database*.

(3) Message-driven Bean

A Message-driven Bean is a bean that integrates with JMS. The EJB container invokes a Bean, when a JMS message is received from the JMS Destination. Unlike a Session Bean or an Entity Bean, since the Message-driven Bean does not have a home interface and a component interface, it cannot be invoked directly from the client.

For a Message-driven Bean, the interfaces used for implementation in the EJB version 2.0 are different from the interfaces used for implementation in the EJB version 2.1 or later versions.

- In EJB 2.0, the following interfaces are implemented:
  - `javax.ejb.MessageDrivenBean` interface
  - `javax.jms.MessageListener` interface
- In EJB 2.1 or later versions, the following interfaces are implemented:
  - `javax.ejb.MessageDrivenBean` interface
  - Interface of any message listener provided by EIS

The versions of the corresponding connector are different in EJB 2.0 and in EJB 2.1 or later versions. The following table describes the correspondence between the versions of EJB and the connector:

Table 2-4: Correspondence between the EJB and connector versions

EJB version	Connector version	
	Connector 1.0	Connector 1.5
EJB 2.0	Y	N
EJB 2.1 or later	Y	--

Legend:

Y: Messages sent from Cosminexus Reliable Messaging or TP1/Message Queue - Access can be received.

--: Messages sent by using the listener interfaces of any format can be received.

N: Messages sent from the corresponding resource adapter cannot be received.

Furthermore, the following table describes the functionality that is different in EJB 2.0 and EJB 2.1 or later versions:

Table 2-5: Functionality that is different in EJB 2.0 and EJB 2.1 or later versions

Functionality	EJB 2.0	EJB 2.1 or later	
		When using the functionality of Connector 1.0	When using the functionality of Connector 1.5
Connectable resource adapters	Resource adapter in compliance with the Connector 1.0 specifications.	Resource adapter in compliance with the Connector 1.0 specifications.	Resource adapter that is compliant with the Connector 1.5 specifications, and in which `Inbound` has been defined.
Usable EIS	• Cosminexus Reliable Messaging • TP1/Message Queue - Access	• Cosminexus Reliable Messaging[#2] • TP1/Message Queue - Access[#2]	Any EIS (including JMS) that supports Connector 1.5.[#1]
Defining a queue	Define in queue definition file.	Define in queue definition file.	Define in the objects to be managed within the DD of resource adapters (`ra.xml`).
JMS version	JMS1.0.2b	JMS1.0.2b	JMS1.1
Managing a connection in a message listener	Specify in the application attribute (pooled-instance). Same as the method-ready pool of the Message-driven Bean.	Specify in the application attribute (pooled-instance). Same as the method-ready pool of the Message-driven Bean.	Differ depending on the used resource adapters.

#1

Cosminexus Reliable Messaging or TP1/Message Queue - Access does not support the Connector 1.5 specifications. Therefore, you cannot use the functionality of Connector 1.5.

#2

You cannot specify a *message-selector* tag in the Deployment Descriptor of EJB2.1. When using the message selector, make changes to receive messages from the Cosminexus JMS provider.

2.2.2 Interface for Enterprise Bean

This subsection describes the interfaces that you can use for implementing an Enterprise Bean. The following table describes the list of usable interfaces:

Table 2-6: List of usable interfaces

Interface name	Description
Remote home interface[#1]	This interface that is specified in EJB specifications 1.1 or later versions inherits `javax.ejb.EJBHome` and is used for a remote client. This interface is mainly used for acquiring Enterprise Bean instances.
Remote component interface[#2]	This interface that is specified in EJB specifications 1.1 or later versions inherits `javax.ejb.EJBObject` and is used for a remote client. This interface mainly defines the business methods.
Remote business interface	This interface is used to define the business methods for invoking the Enterprise Beans from a remote client. The defined interface need not be inherited.
Local home interface	This interface that is specified in EJB specifications 2.0 or later versions inherits `javax.ejb.EJBLocalHome` and is used for a local client. This interface is mainly used for acquiring Enterprise Bean instances.
Local component interface	This interface that is specified in EJB specifications 2.0 or later versions inherits `javax.ejb.EJBLocalObject` and is used for a local client. This interface mainly defines the business methods.
Local business interface	This interface is used to define the business methods for invoking the Enterprise Beans from a local client. The defined interface need not be inherited.

#1

This interface is called a home interface in EJB specifications 1.1.

#2

This interface is called a remote interface in EJB specifications 1.1.

In this manual, multiple interfaces are grouped together and are given a general name. The following table describes the general names for the interfaces that are used in the descriptions of this manual:

Table 2-7: General names of interfaces

General name of the interface	Description
Home interface	This is the general name for the following interfaces: • Remote home interface • Local home interface
Component interface	This is the general name for the following interfaces: • Remote component interface • Local component interface
Business interface	This is the general name for the following interfaces: • Remote business interface • Local business interface
Remote interface	This is the general name for the following interfaces: • Remote home interface • Remote component interface • Remote business interface This name may, however, indicate only the remote component interface and the remote business interface.
Local interface	This is the general name for the following interfaces: • Local home interface

General name of the interface	Description
Local interface	• Local component interface • Local business interface This name may, however, indicate only the local component interface and the local business interface.

These interfaces are implemented for a Session Bean or an Entity Bean. A Message-driven Bean does not have these interfaces.

With a Session Bean, you can use the No-Interface view to invoke from the local interface. In such cases, you can omit the implementation of the interface. For details on the No-Interface view, see *2.16 Omitting local business interfaces (Using No-Interface view)*.

(1) Remote interface

In a remote interface, the Enterprise Beans are invoked by RMI-IIOP communication, based on the provisions in the Java RMI interface. The Enterprise Beans existing in a JavaVM of another client can be invoked, but that causes a communication overhead at run time. The arguments and the return values during execution of the method are passed by value.

(2) Local interface

In a local interface, the Enterprise Beans are invoked by invoking the Java methods and communication does not occur. This interface can be used only when the clients exist in the same J2EE application. Furthermore, when using a local interface, unlike a remote interface, the arguments and the return values during execution of the method are passed by reference.

(3) Functionality supporting a business interface

The following functionality can be used even when you use a business interface:

- Functionality for optimizing local invocation
  For details on the functionality, see *2.13.1 Optimizing local invocation in the EJB remote interface*.
- Pass by reference functionality of remote interface values
  For details on the functionality, see *2.13.2 Referencing and passing the values of the EJB remote interface*.
- Timeout during RMI-IIOP communication
  For details on the functionality, see *2.11.5 Timeout of RMI-IIOP communication*.
  Specify the communication timeout in the definition file or in API. The precautions while setting the timeout, when a business interface is used, are described below:
  - When specifying in the definition file
    When using the DI functionality, the communication timeout property of JNDI is also enabled, since JNDI is used in the EJB container.
  - When specifying in API
    You can specify in the thread, but not in the object.

  When a timeout occurs, exception `java.rmi.RemoteException` (`org.omg.CORBA.TIMEOUT`) is thrown if the business interface inherits `java.rmi.Remote`, and exception `javax.ejb.EJBException` (RemoteException (`org.omg.CORBA.TIMEOUT`)) is thrown if the business interface does not inherit `java.rmi.Remote`.
- EJB check functionality
  For details on the functionality, see *2.3 Checking the compliance with EJB specifications*.
- Method timeout functionality for monitoring the J2EE application execution time
  For details on the functionality, see *5.3.2 Monitoring the execution time of the J2EE applications* in the *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*.
  When invoking the EJB methods with annotation specified, you can apply the timeout value for each method as described in the table below:

Table 2-8: Applicability of a timeout for Enterprise Bean

Interface	Method	Stateless SessionBean	Stateful SessionBean	Singleton Session Bean	Message DrivenBean	Persistence API
Business interface	Business method[#1]	Y	Y	Y	N/A	N/A
Home or component interface[#2]	`create`	N	Y	N/A	N/A	N/A
	Business method	Y	Y	N/A	N/A	N/A
	`remove`	N	Y	N/A	N/A	N/A
`javax.ejb.TimerObject`	`ejbTimeout`	Y	N/A	Y	N/A	N/A

Legend:

Y: Applied

N: Not applied

N/A: Not applicable

#1

Includes methods, wherein the `@Timeout` annotation and the `@Remove` annotation are appended.

#2

Indicates interfaces that use `@RemoteHome` or `@LocalHome`.

(4) Invoking the Enterprise Bean of a business interface

The case of invoking a business interface running in another J2EE application on the same J2EE server and that of invoking a business interface running in another J2EE server are explained below:

In either case, the invoking EJB-JAR file or the WAR file contains the business interface of the invoked Enterprise Bean as well as the user-created classes used in the interface. Furthermore, lookup is performed by using either the name that is bound automatically to the JNDI name space (Portable Global JNDI name or a name starting with HITACHI_EJB) or the optional name set up in the user-specified name space functionality. For details on the names used for lookup, see *2.5 Look up by the name starting with HITACHI_EJB* in the *uCosminexus Application Server Common Container Functionality Guide*.

- **When invoking the Enterprise Bean of a business interface running in another J2EE application on the same J2EE server**

  Include the business interface of the invoked Enterprise Bean as well as the user-created classes used in the interface in the invoking EJB-JAR file or the WAR file.

  If the `ejbserver.rmi.localinvocation.scope` key of the user property file for the J2EE server is "None", acquire the stubs by using the `cjgetstubsjar` command for the invoked Enterprise Bean and include these stubs in the invoking EAR file. #

- **When invoking the Enterprise Bean of a business interface running in another J2EE server**

  Acquire the stubs by using the `cjgetstubsjar` command for the invoked Enterprise Bean and include these stubs in the invoking EAR file.#

  #

  If you use the dynamic class loading functionality for invoking the business interface between applications, you need not include the stubs. However, from the performance perspective, Hitachi does not recommend the use of the dynamic class loading functionality.

2.2.3 Lifecycle of Enterprise Beans

The lifecycle for various types of Enterprise Beans is explained below:

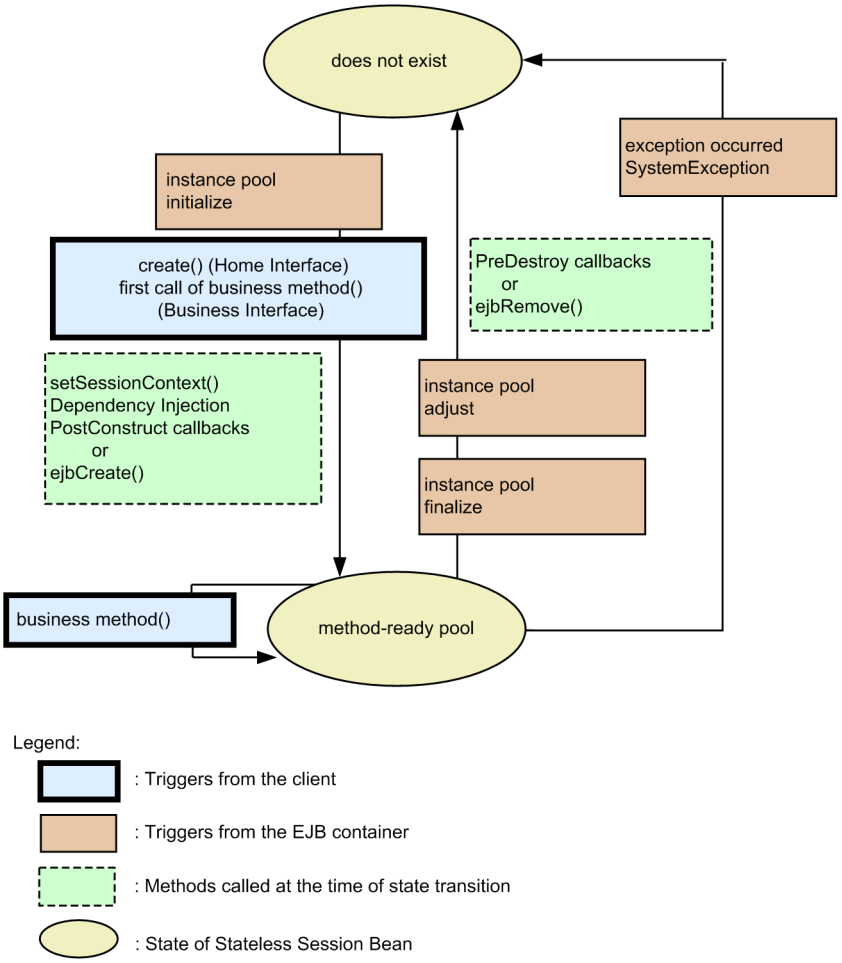
(1) Lifecycle of Session Beans

The lifecycle of a Session Bean is different for a Stateless Session Bean and for a Stateful Session Bean.

(a) In the case of a Stateless Session Bean

The following figure illustrates the lifecycle of a Stateless Session Bean.

Figure 2-1: Lifecycle of a Stateless Session Bean



does not exist:

State when the Stateless Session Bean does not exist

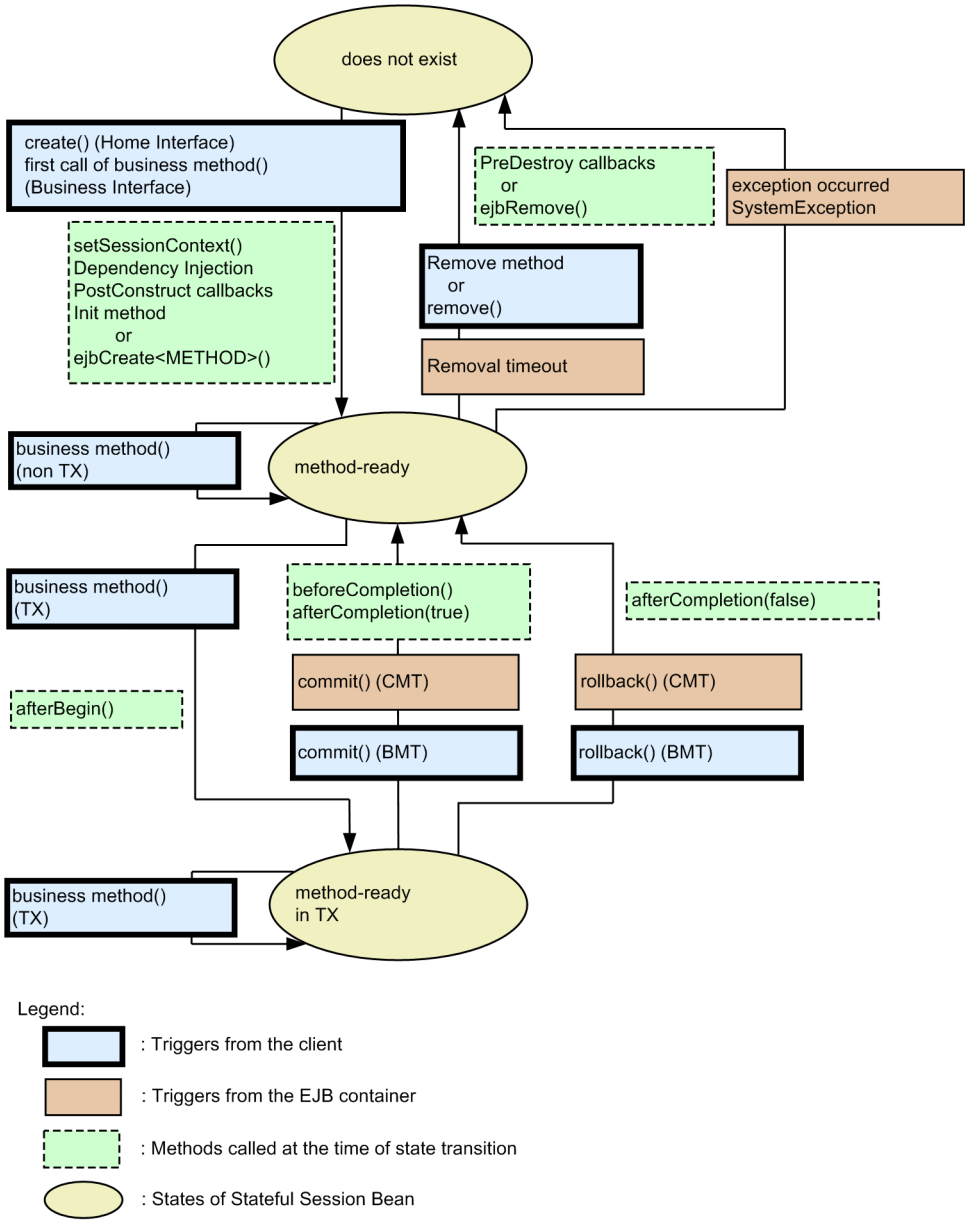
method-ready pool:

State when the Stateless Session Bean exists in an executable state in the method-ready pool

(b) In the case of a Stateful Session Bean

The following figure illustrates the lifecycle of a Stateful Session Bean.

Figure 2-2: Lifecycle of a Stateful Session Bean



does not exist:

State when the Stateful Session Bean does not exist

method-ready:

State when the Stateful Session Bean is activated and exists in an executable state (no transaction) in the method-ready pool

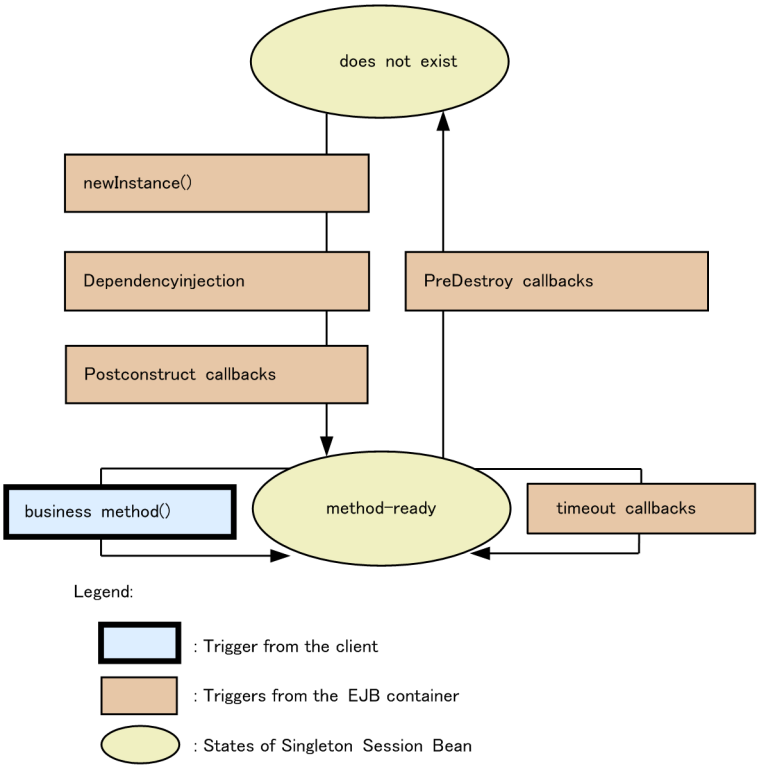
method-ready in TX:

State when the Stateful Session Bean is activated and exists in an executable state (transaction exists) in the method-ready pool

(c) In the case of a Singleton Session Bean

The following figure illustrates the lifecycle of a Singleton Session Bean.

Figure 2-3: Lifecycle of a Singleton Session Bean



does not exist:

A state when the Singleton Session Bean does not exist

method-ready:

A state when the Singleton Session Bean is ready for execution

An EJB container initializes Singleton Session Bean. Note that you can explicitly define the initialization time during an application development, by specifying the annotation.

- To initialize a Singleton Session Bean in the starting process of an application, specify the `@Startup` annotation in the Singleton Session Bean class. With this, the initialization process is executed before a request is sent from an external client.
- By specifying the `@DependsOn` annotation, you can define the creation order of a Singleton Session Bean and other components of the Singleton Session Bean.

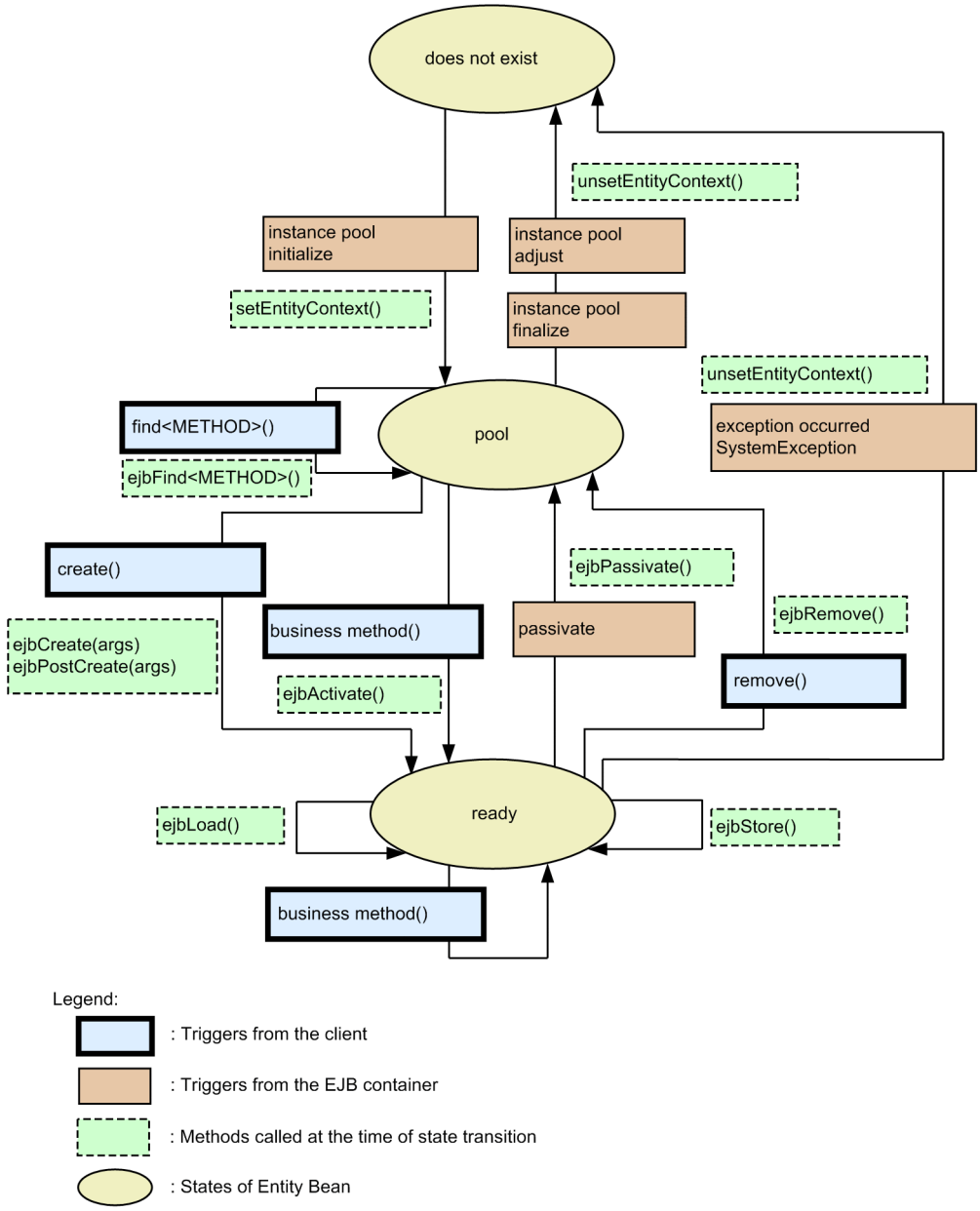
Once initialized, the Singleton Session Bean instance exists until the application stops. Operations of an EJB container, when a Singleton Session Bean is destroyed, are as follows:

- If the callback interceptor method that executes the `PreDestroy` process exists, it invokes the appropriate method. During the invocation of this method, the state is maintained so that the EJB container can use all the Beans for which the dependency is defined using `DependOn`.
- On the completion of the `PreDestroy` process, the EJB container deletes the instances of the Singleton Session Bean.

(2) Lifecycle of Entity Beans

The following figure shows the lifecycle of an Entity Bean:

Figure 2-4: Lifecycle of a Entity Bean



does not exist:

State when the Entity Bean does not exist

pool:

State when the Entity Bean is passivated and exists in the passive pool

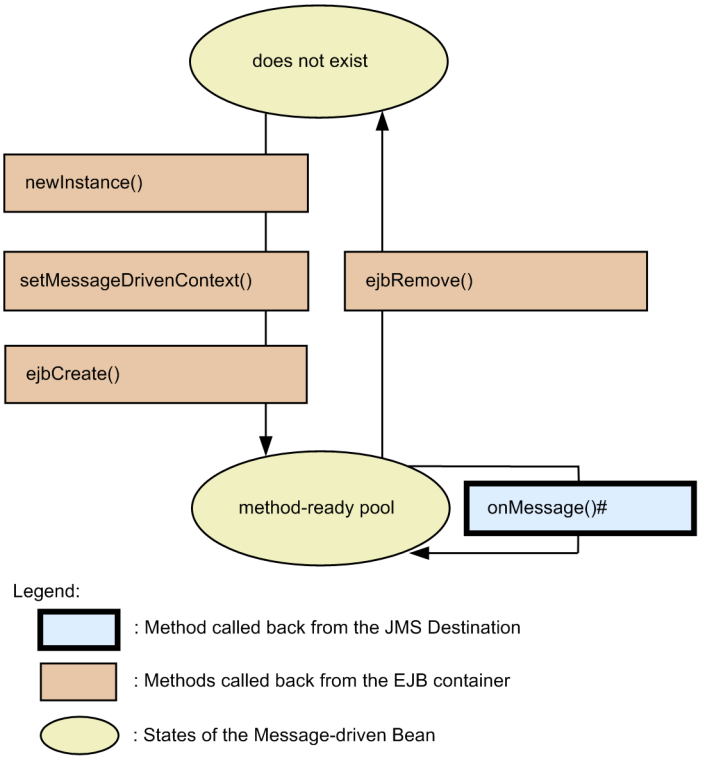
ready:

State when the Entity Bean is activated and exists in the ready pool

(3) Lifecycle of Message-driven Beans

The following figure shows the lifecycle of a Message-driven Bean.

Figure 2-5: Lifecycle of a Message-driven Bean



In case of Message-driven Bean which does not use javax.jms.MessageListener interface, method any MessageListener interface method is called back.

does not exist:

State when the Message-driven Bean does not exist

method-ready pool:

State when the Message-driven Bean is in the method-ready state and exists in the ServerSession pool

2.3 Checking the compliance with EJB specifications

In the EJB container, whenever a J2EE application is started, a check is performed to see whether each Enterprise Bean conforms to the specifications of EJB. As a result of this check, if a J2EE application contains an Enterprise Bean that does not conform to the specifications, the J2EE application fails to start. At this point, an error message is displayed.

The EJB checking process runs at the following times:

- When a J2EE application is imported for the first time on a J2EE server and started.
- When the configuration of a J2EE application is changed (including changes by using the redeploy functionality) and then the application is started.
- When a J2EE server is started for the first time after upgrade installation.

If the J2EE application starts successfully at these times, the EJB check does not run during the subsequent start and stop operation of the J2EE application.

2.4 Mapping of CMP fields and data types

This section describes the mapping of the CMP fields and data types.

In a CMP field, the specifiable range and the availability to specify in the primary key is fixed for each Java data type. Furthermore, the support for the Java data type and the SQL data type of the database will differ for each type of the database.

The following table describes the organization of this section:

Table 2-9: Organization of this section (Mapping CMP field and data type)

Category	Title	Reference location
Description	Range of Java data type supported in CMP	*2.4.1*
	Mapping the CMP field and database	*2.4.2*
Notes	Precautions for using CMP	*2.4.3*

Note:

There is no specific explanation of *Implementation*, *Setup*, and *Operation* for this functionality.

2.4.1 Range of Java data type supported in CMP

The following table describes the range of Java data types supported in the CMP Entity Bean of an EJB container and also describes the availability of specifications for the primary key:

Table 2-10: Range of Java data types supported in CMP

Java data type	Range of values	Specification to the primary key
`boolean`	true, false	N
`java.lang.Boolean`		Y
`byte`	-128 to 127	N
`java.lang.Byte`		Y
`char`	'\u0000' to '\uffff'(0 to 65535)	N
`java.lang.Character`		Y
`short`	-32768 to 32767	N
`java.lang.Short`		Y
`int`	-2147483648 to 2147483647	N
`java.lang.Integer`		Y
`long`	-9223372036854775808 to 9223372036854775807	N
`java.lang.Long`		Y
`float`#	± 1.40239846e^{-45} to ± 3.40282347e^{38}	N
`java.lang.Float`#		Y
`double`#	± 4.94065645841246544e^{-324} to ± 1.79769313486231570e^{308}	N
`java.lang.Double`#		Y
`byte[]`	1Byte to 2147483647Byte	N
`java.lang.String`	--	Y

Java data type	Range of values	Specification to the primary key
`java.math.BigDecimal`	--	N
`java.sql.Date`	--	N
`java.sql.Time`	00:00:00 to 23:59:59	N
`java.sql.TimeStamp`	--	N
Serializable type	--	N

Legend:

Y: Can be specified for the primary key

N: Cannot be specified for the primary key

--: Not applicable

#

The floating points might be rounded off.

2.4.2 Mapping the CMP field and database

This section describes the mapping of the CMP field and database. The mapping will differ for each type of database.

(1) Mapping for HiRDB

The following table describes the mapping of the CMP field and database for HiRDB.

In the following table, *Java data type* is the data type of Java supported by CMP, *JDBC data type* is the `java.sql.Types`. data type of the JDBC corresponding to the data type of Java, and *SQL data type* is the DB column type recommended for mapping with the Java data type:

Table 2-11: Mapping of the CMP field and database (when using HiRDB)

Java data type	JDBC data type	SQL data type
`boolean`	SMALLINT	SMALLINT
`java.lang.Boolean`		
`byte`	SMALLINT	SMALLINT
`java.lang.Byte`		
`char`#1	CHAR	CHAR(4)
`java.lang.Character`#1		
`short`	SMALLINT	SMALLINT
`java.lang.Short`		
`int`	INTEGER	INTEGER
`java.lang.Integer`		
`long`	DECIMAL	DECIMAL(22)
`java.lang.Long`		
`float`	REAL	REAL, SMALLFLT
`java.lang.Float`		
`double`	FLOAT	DOUBLE PRECISION

Java data type	JDBC data type	SQL data type
`java.lang.Double`	FLOAT	DOUBLE PRECISION
`byte[]`#2	LONGVARBINARY	BLOB
`java.lang.String`#1	VARCHAR	VARCHAR(m) CHAR(n) MVARCHAR(m) MCHAR(n) NVARCHAR(x) NCHAR(y)#3
`java.math.BigDecimal`	DECIMAL	DECIMAL(m,n)#4
`java.sql.Date`	DATE	DATE#5
`java.sql.Time`	TIME	TIME
`java.sql.TimeStamp`#6	CHAR	CHAR(29)
Serializable type#2	LONGVARBINARY	BLOB

#1

For details on the precautions to be taken when using the Fixed-length string SQL type, see *2.4.3 Precautions for using CMP*.

#2

HiRDB BLOB maximum value 2147483647 bytes

However, the maximum data size that can be handled depends on the upper limit for the JDBC driver. When you use HiRDB Type4 JDBC Driver, there are no JDBC driver-based restrictions.

#3

The following are the respective ranges of m, n, x, and y:

m: 1 to 32000, n: 1 to 30000, x: 1 to 16000, y: 1 to 15000

#4

The following are the respective ranges of m and n:

m: 1 to 29, n: 1 to 29

#5

The DATE range is from 0001/01/01 to 9999/12/31.

#6

Saved as a string with the format `yyyy-mm-dd hh:mm:ss.fffffffff` (JDBC date escape).

(2) Mapping list for Oracle

The following table describes the mapping of the CMP field and the database for Oracle.

In the following table, *Java data type* is the data type of Java supported by CMP, *JDBC data type* is the `java.sql.Types.` data type of the JDBC corresponding to the data type of Java, and *SQL data type* is the DB column type that is recommended for the mapping with the Java data type:

Table 2-12: Mapping of the CMP field and database (when using Oracle)

Java data type	JDBC data type	SQL data type
`boolean`	NUMERIC	NUMBER(38)
`java.lang.Boolean`		
`byte`	NUMERIC	NUMBER(38)
`java.lang.Byte`		
`char`#1	CHAR	CHAR(4)

Java data type	JDBC data type	SQL data type
`java.lang.Character`[#1]	CHAR	CHAR(4)
`short`	NUMERIC	NUMBER(38)
`java.lang.Short`		
`int`	NUMERIC	NUMBER(38)
`java.lang.Integer`		
`long`	NUMERIC	NUMBER(22)
`java.lang.Long`		
`float`	NUMERIC	NUMBER
`java.lang.Float`		
`double`[#2]	FLOAT	FLOAT(126)
`java.lang.Double`[#2]		
`byte[]`[#3]	LONGVARBINARY	LONG RAW
`java.lang.String`[#1]	VARCHAR	VARCHAR(m) CHAR(n) LONG[#4]
`java.math.BigDecimal`	NUMERIC	NUMBER(m,n)[#5]
`java.sql.Date`	DATE	DATE[#6#7]
`java.sql.Time`	CHAR	CHAR(8)[#8]
`java.sql.TimeStamp`	TimeStamp	DATE[#7 #9]
Serializable type[#3]	LONGVARBINARY	LONG RAW

Note:

The BLOB data type mapped in `java.sql.Types.BLOB` and the CLOB data type mapped in `java.spl.Types.CLOB` cannot be handled.

#1

For details on the precautions to be taken for using the Fixed-length string SQL type, see *2.4.3 Precautions for using CMP*.

#2

Range of Oracle FLOAT(126) 1E-125 to 9.9 ... 9E125

The value might be rounded off.

#3

Oracle LONG RAW maximum value 2 gigabytes.

However, the maximum data size that can be handled depends on the upper limit for the JDBC driver.

#4

The following are the respective ranges of m and n:

m = 1 to 4000, n = 1 to 2000

Furthermore, if "" (blank string of zero length) is saved in Oracle, the value will be converted to `NULL`.

#5

The following are the respective ranges of m and n:

m = 1 to 38, n = -84 to 127

#6

The DATE range is from `-4712/01/01` to `9999/12/31`.

#7

The minus value cannot be handled correctly between Java-JDBC drivers for data corresponding to B.C (Before Christ). Therefore, the value cannot be guaranteed.

#8

Saved as a string with format `hh:mm:ss` (JDBC date escape).

#9

The range of DATE is from `-4712/01/01 00:00:00` to `9999/12/31 23:59:59`.

2.4.3 Precautions for using CMP

This section describes the precautions for using CMP in the EJB container of Application Server.

- **When using a fixed-length string SQL type in the CMP field**

  When using a fixed-length string SQL type (CHAR type of HiRDB or Oracle), the number of characters that are less than the required number of digits might be filled up with spaces when they are saved in the database. Therefore, the spaces might be added at the end of the data during the data creation. Pay proper attention when using this type.

- **When using a fixed-length string SQL type in the primary key**

  When using a fixed-length string SQL type (CHAR type of HiRDB or Oracle) in a key, the number of characters that are less than the required number of digits will be filled up with spaces when they are saved in the database. Therefore, the value might differ from the data during the data creation, and the target Entity Bean might not be acquired. Pay proper attention when using this type.

- **Support when an exception occurs because of the `remove` method**

  If an exception occurs while the `remove` method is running, data is retained on the database. Delete the data manually.

2.5 Registering a reference in the JNDI name space of the EJB container

This section describes the registration of a reference in the JNDI name space of the EJB container.

The EJB container supports registration of references in the java:comp/env name space.

The following table describes the organization of this section:

Table 2-13: Organization of this section (Registering a reference in the JNDI name space of the EJB container)

Category	Title	Reference location
Description	Registering a reference in the java:comp/env name space	*2.5.1*
Implementation	Defining in cosminexus.xml	*2.5.2*
Setup	Setting up in the execution environment	*2.5.3*

Note:
There is no specific explanation of *Operation* and *Notes* for this functionality.

For details on the naming management functionality that can be used in Application Server, see *2. Naming Management* in the *uCosminexus Application Server Common Container Functionality Guide*.

2.5.1 Registering a reference in the java:comp/env name space

The EJB container supports registration of references in the java:comp/env name space. Hitachi recommends that you register the corresponding reference in the name space shown within brackets (()).

- Environment entry (`java:comp/env`)
- EJB home object reference (`java:comp/env/ejb`)
- Business interface reference (`java:comp/env/ejb`)
- JMS (`java:comp/env/jms`)
- JDBC data source (`java:comp/env/jdbc`)
- JavaMail session (`java:comp/env/mail`)
- uCosminexus TP1 Connector (`java:comp/env/eis`)
- JavaBeans resource (`java:comp/env/bean`)

As a result, indirect lookup can be performed by using the JNDI name space of `java:comp/env`.

2.5.2 Defining in cosminexus.xml

The definition of reference mapping is specified in the `<ejb-jar>` tag of `cosminexus.xml`.

When using `java:comp/env`, you must define the reference mapping in the reference-source application. The tag to be specified is different for each type of target Enterprise Bean.

The following table describes the definition of reference mapping in `cosminexus.xml`:

Table 2-14: Definition of reference mapping in cosminexus.xml

Items	Tag to be specified	Setting contents
Resolving the references of the resources	In the case of a Session Bean `<session>-<resource-ref>` tag `<session>-<resource-env-ref>` tag	• Set up the information of the resource in the `<resource-ref>` tag.

Items	Tag to be specified	Setting contents
Resolving the references of the resources	In the case of an Entity Bean <entity>-<resource-ref> tag <entity>-<resource-env-ref> tag In the case of Message Driven Bean <message-driven>-<resource-ref> tag <message-driven>-<resource-env-ref> tag	• Set up the information of the resource environment in the <resource-env-ref> tag.

For details on the tags to be specified, see *2.2.2 Details of EJB-JAR properties* in the *uCosminexus Application Server Application and Resource Definition Reference Guide*.

2.5.3 Setting up in the execution environment

The reference mapping can also be set up in the execution environment. Specify the settings in the J2EE application imported into a J2EE server, and execute the setup only when you set up or change the property of a J2EE application that does not include `cosminexus.xml`.

A J2EE application is set up in the execution environment using the server management commands and the property files. Use the following property files to define the reference mapping:

Table 2-15: Property files used to define reference mapping

Setting target	Attribute files
Session Bean	Session Bean attribute file
Entity Bean	Entity Bean attribute file
Message-driven Bean	Message-driven Bean attribute file

The tags specified in the property files correspond to the DD or `cosminexus.xml`. Note that the tags to be specified differ depending on whether the reference of the Enterprise Bean or the reference of the resource is to be resolved.

For details on the `cosminexus.xml` settings, see *2.5.2 Defining in cosminexus.xml*.

2.6 Connecting to an external resource

For details on the resources that you can use in an EJB container and also about J2EE resources, see *3. Resource Connection and Transaction Management* in the *uCosminexus Application Server Common Container Functionality Guide*.

2.7 Transaction management in an Enterprise Bean

This section describes the transaction management in an Enterprise Bean.

The transactions of an Enterprise Bean are managed either by the Enterprise Bean or by the EJB container.

The following table describes the organization of this section:

Table 2-16: Organization of this section (Transaction management in an Enterprise Bean)

Category	Title	Reference location
Description	Types of transaction management methods in an Enterprise Bean	*2.7.1*
	BMT	*2.7.2*
	CMT	*2.7.3*
Implementation	Defining in cosminexus.xml	*2.7.4*
Setup	Settings in the execution environment	*2.7.5*

Note:
There is no specific description of *Operation* and *Notes* for this functionality.

For details on the transaction management functionality that you can use in Application Server, see *3.4 Managing transactions* in the *uCosminexus Application Server Common Container Functionality Guide*.

2.7.1 Types of transaction management methods in an Enterprise Bean

The following two methods of transaction management are used for the Enterprise Beans supported by the EJB container:

- BMT (Bean Managed Transaction)
- CMT (Container-Managed Transaction)

This functionality is implemented based on the transaction management functionality of the J2EE service.

Note that the methods of transaction management are set as attributes (properties) of the Session Beans or the Message-driven Beans included in a J2EE application. For details on the J2EE application settings, see *2.7.4 Defining in cosminexus.xml*.

For details on the transaction management, see *3. Resource Connection and Transaction Management* in the *uCosminexus Application Server Common Container Functionality Guide*.

2.7.2 BMT

BMT is a model that performs transaction management in an Enterprise Bean. The BMT target is the Session Beans and the Message-driven Beans (BMT does not target the Entity Beans, the CMT target is always Entity Beans).

During BMT, use `javax.transaction.UserTransaction` in the business methods of the Enterprise Bean and perform the following operations:

1. Start the transaction
2. Update the resource manager
3. Commit or roll back the transaction

In a Stateless Session Bean, one transaction needs to be concluded (committed or rolled back) in one business method. In a Message-driven Bean, a transaction needs to be concluded (committed or rolled back) in one of the following methods: The methods differ depending on the versions of EJB.

- `onMessage` method (for EJB 2.0)

- Method of any message listener (for EJB 2.1 or later versions)

On the other hand, in a Stateful Session Bean, many business methods can be included in a single transaction scope. At this point, the EJB container maintains the relationship between the Bean instance and the transaction. The following figure shows the Stateful Session Bean in BMT:

Figure 2-6: Stateful Session Bean in BMT

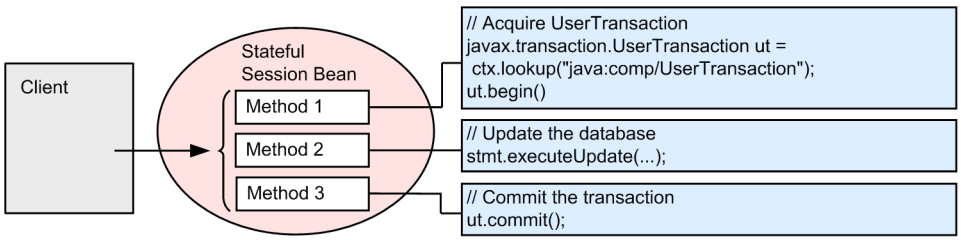


Table 2-17: Transaction control in BMT

Client-side transactions	Transactions of Bean instance	Transactions connected to the methods
--	--	--
T1	--	--
--	T2	T2
T1	T2	T2

Legend:

--: Either a transaction is not started or is not mapped

T1: Transaction started in the client

T2: Transaction started in the Bean

Important note

If a transaction is not concluded and a new transaction is started by using the `UserTransaction.begin` method, the EJB container throws `javax.transaction.NotSupportedException`.

2.7.3 CMT

CMT is a model that performs transaction management in the EJB container. The CMT target is the Session Beans, Entity Beans, and the Message-driven Beans. In CMT, you specify the transaction attributes for each method of Bean. Note that the transaction attributes are set as attributes (properties) of the Session Beans, Entity Beans, or the Message-driven Beans included in a J2EE application. For details on the J2EE application settings, see *2.7.4 Defining in cosminexus.xml*.

When a transaction cannot be committed, the EJB container performs the following processing:

1. Logging of application errors
2. Transaction rollback
3. Destroying Bean instances
4. Throwing exception `java.rmi.RemoteException` in the client invoked by the remote component interface, and exception `javax.ejb.EJBException` in the client invoked either by the local component interface or by the business interface. If, however, the remote business interface inherits `java.rmi.Remote`, the exception `java.rmi.RemoteException` is thrown.

(1) Types of transaction attributes and their behavior

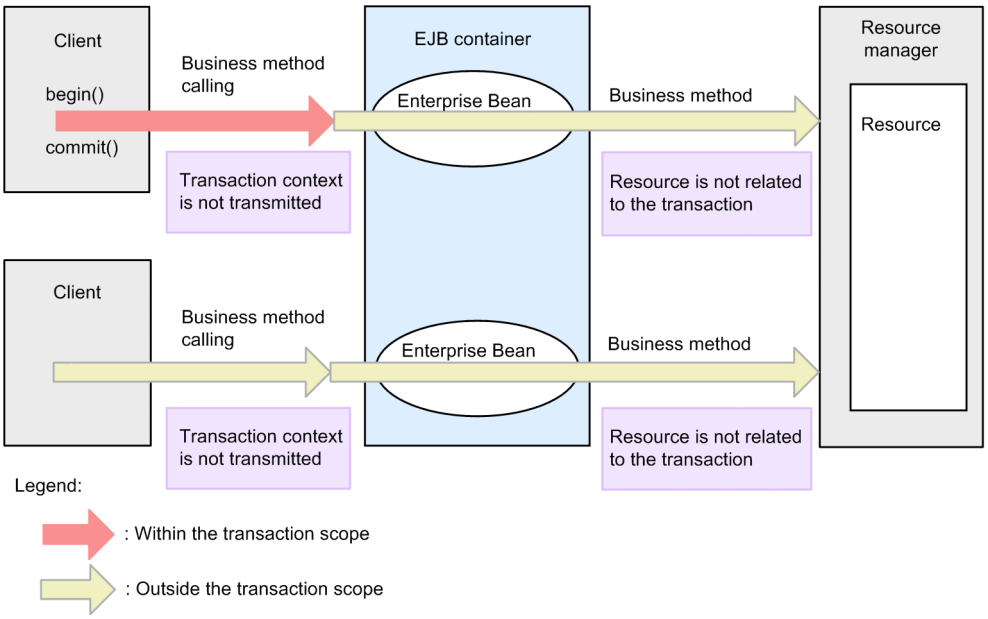
The behavior of transaction attributes is explained in the figures below, for each type of transaction attribute:

■ NotSupported attribute

If the client invokes the business methods of an Enterprise Bean within the transaction scope, the transaction context is not propagated in the Enterprise Bean. Also, if the client invokes the business methods of the Enterprise Bean outside the transaction scope, the transaction context is again not propagated in the Enterprise Bean.

The following figure shows the behavior of the `NotSupported` attribute:

Figure 2-7: NotSupported attribute

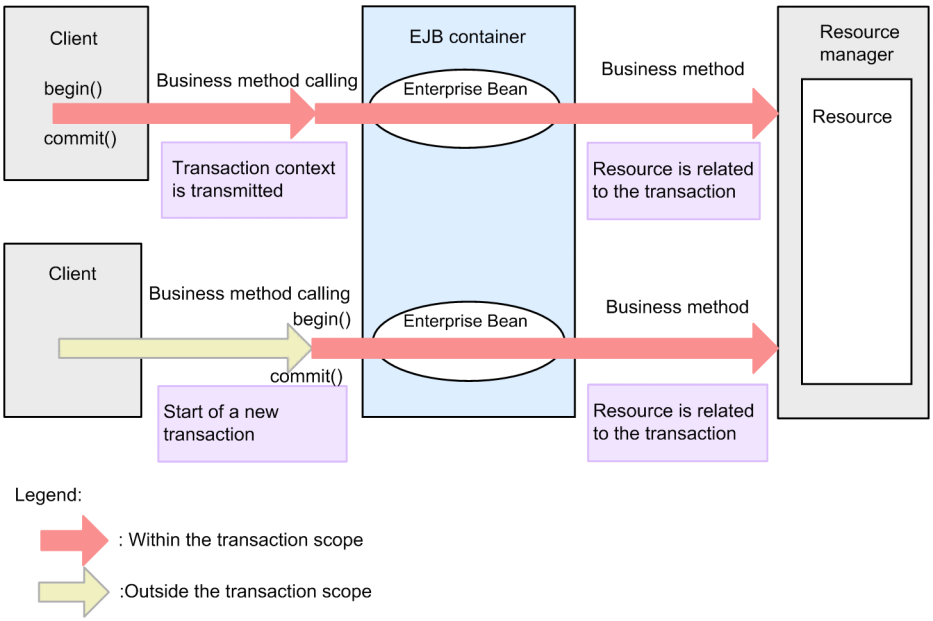


■ Required attribute

If the client invokes the business methods of an Enterprise Bean within the transaction, the transaction context is propagated in the Enterprise Bean, and the business methods of the Enterprise Bean enter the transaction scope of the caller. If the client invokes the business methods of the Enterprise Bean outside the transaction scope, a new transaction is started in the Enterprise Bean.

The following figure shows the behavior of the `Required` attribute:

Figure 2-8: Required attribute

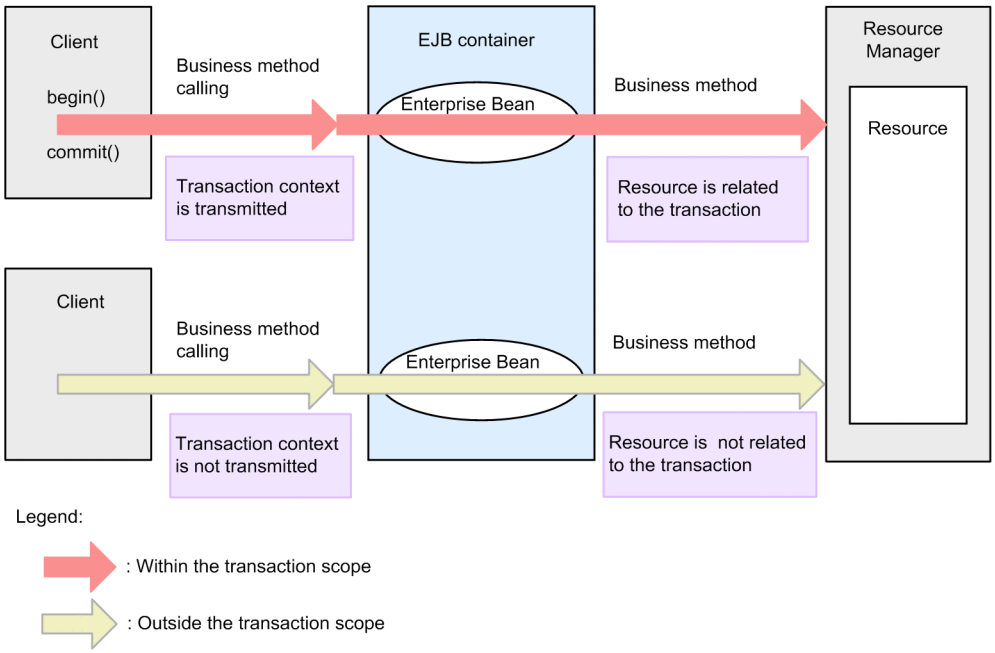


■ Supports attribute

If the client invokes the business methods of an Enterprise Bean within the transaction, the transaction context is propagated in the Enterprise Bean, and the business methods of the Enterprise Bean enter the transaction scope of the caller. If the client invokes the business methods of the Enterprise Bean outside the transaction scope, the transaction context is not propagated in the Enterprise Bean.

The following figure shows the behavior of the `Supports` attribute:

Figure 2-9: Supports attribute

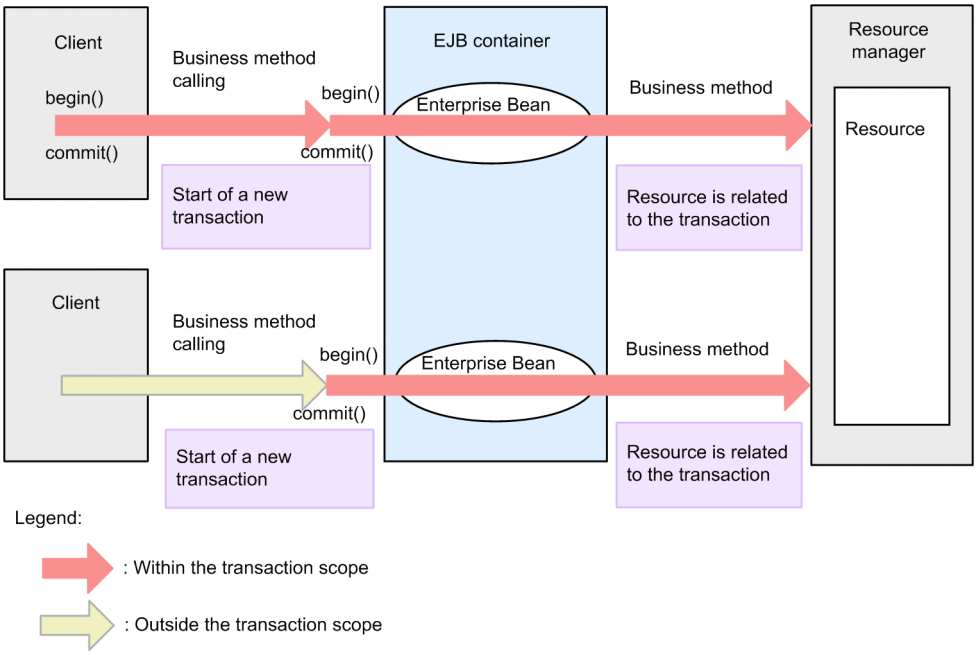


■ RequiresNew attribute

If the client invokes the business methods of an Enterprise Bean within the transaction scope, the EJB container starts a new transaction. Even if the client invokes the business methods of the Enterprise Bean outside the transaction scope, a new transaction is started by the EJB container, in the same way.

The following figure shows the behavior of the `RequiresNew` attribute:

Figure 2-10: RequiresNew attribute



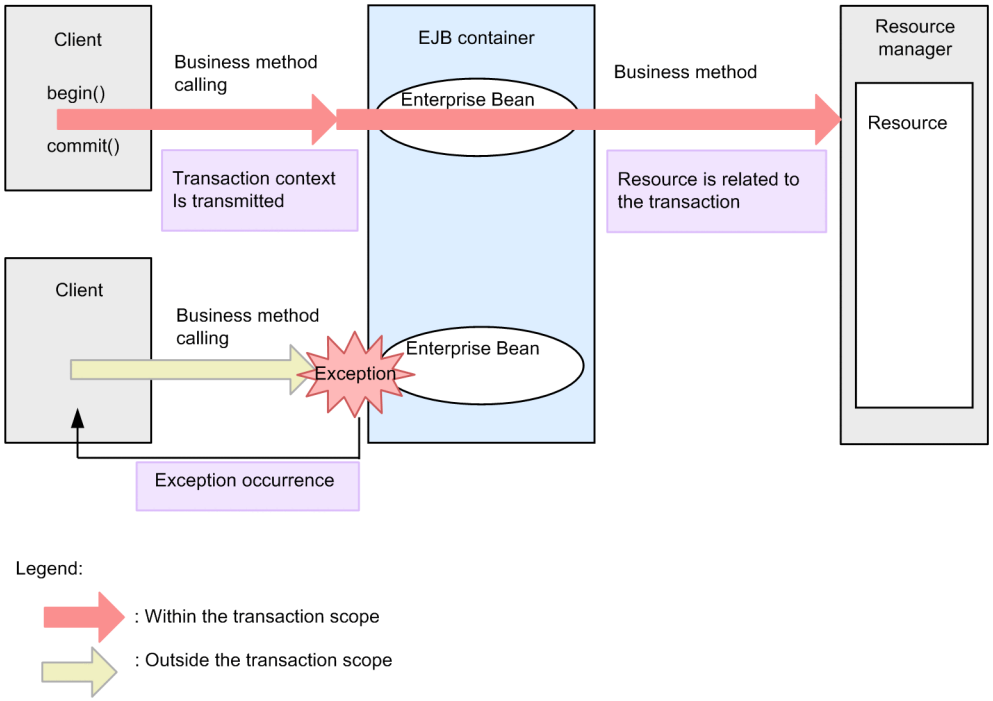
■ Mandatory attribute

If the client invokes the business methods of an Enterprise Bean within the transaction, the transaction context is propagated in the Enterprise Bean, and the business methods of the Enterprise Bean enter the transaction scope of the caller. If the client invokes the business methods of the Enterprise Bean outside the transaction scope, the EJB container throws the following exceptions in the client:

- When business interface is used, exception `javax.ejb.EJBTransactionRequiredException` is thrown. In the case of remote business interface that inherits `java.rmi.Remote`, exception `javax.transaction.TransactionRequiredException` is thrown.
- When remote component interface is used, exception `javax.transaction.TransactionRequiredException` is thrown.
- When local component interface is used, exception `javax.ejb.TransactionRequiredLocalException` is thrown.

The following figure shows the behavior of `Mandatory` attribute:

Figure 2-11: Mandatory attribute



■ Never attribute

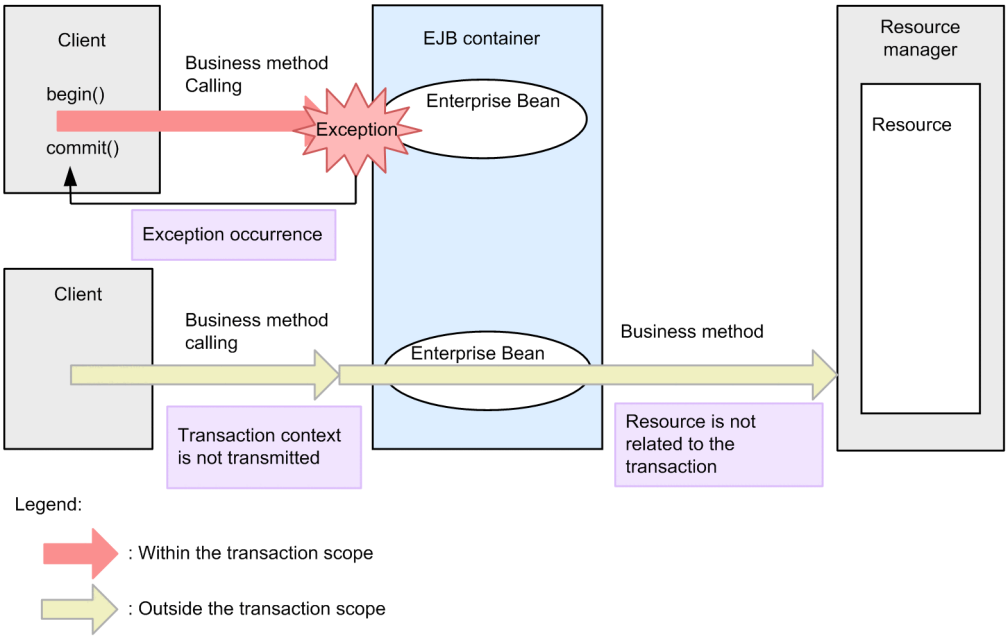
If the client invokes the business methods of an Enterprise Bean within the transaction scope, the EJB container throws the following exceptions in the client:

- When business interface is used, exception `javax.ejb.EJBException` is thrown. In the case of remote business interface that inherits `java.rmi.Remote`, exception `java.rmi.RemoteException` is thrown.
- When remote component interface is used, exception `java.rmi.RemoteException` is thrown.
- When local component interface is used, exception `javax.ejb.EJBException` is thrown.

If the client invokes the business methods of the Enterprise Bean outside the transaction scope, the transaction context is not propagated in the Enterprise Bean.

The following figure shows the behavior of the `Never` attribute:

Figure 2-12: Never attribute



(2) Transaction attributes that can be specified for each type of Enterprise Bean

The following table describes the transaction attributes and the default values that can be specified for each type of Enterprise Bean. The transaction attributes that can be specified for each type of Bean are defined in the EJB specifications. CMP 2.0 is considered as optional in the EJB specifications, and other transaction attributes cannot be specified. The default value is not defined in the EJB specifications.# For an application server, when CMT is specified in the DD of EJB, and the transaction attribute is not specified, the default setting will be as shown in the table below:

Table 2-18: Transaction attributes and default values that can be specified for each type of Enterprise Bean

Type	Transaction attribute that can be specified	Default value
Stateless Session Bean Stateful Session Bean (other than SessionSynchronization) Entity Bean (BMP, CMP 1.1)	• `Supports` • `NotSupported` • `Required` • `RequiresNew` • `Mandatory` • `Never`	`Supports`
Stateful Session Bean (SessionSynchronization)	• `Supports` • `NotSupported` • `Required` • `RequiresNew` • `Mandatory` • `Never`	`Required`
Singleton Session Bean (Except `PostConstruct/PreDestroy`) #1	• `Supports` • `NotSupported` • `Required` • `RequiresNew` • `Mandatory`	`Required`

Type	Transaction attribute that can be specified	Default value
Singleton Session Bean (Except `PostConstruct/PreDestroy`) #1	• `Never`	`Required`
Entity Bean (CMP 2.0)	• `Required` • `RequiresNew` • `Mandatory`	`Required`
Message-driven Bean (when using a resource adapter in compliance with Connector 1.0)	• `Required` • `NotSupported`	`Required`
Message-driven Bean (when using a resource adapter in compliance with Connector 1.5)	• `Required`#2 • `NotSupported`	`Required`#2
Session Bean without a DD using annotations	• `Supports` • `NotSupported` • `Required` • `RequiresNew` • `Mandatory` • `Never`	`Required`

Note:

The default value (`Required`) is, however, defined for an Enterprise Bean that does not have a DD.

#1

With Application Server, even if you specify a transaction attribute in the lifecycle callback method (`PostConstruct` or `PreDestroy`) of Singleton Session Bean, it is not effective.

#2

Cannot be specified for the CJMSP resource adapter or FTP inbound adapter.

(3) Transaction attributes of a Stateful Session Bean (SessionSynchronization)

`SessionSynchronization` is the interface used for reporting messages, when you start or stop a transaction. According to the EJB specifications, you can specify `Required`, `RequiresNew`, or `Mandatory` in the transaction attributes of Stateful Session Beans in which `SessionSynchronization` is implemented. Correspondingly, you can specify `Supports`, `NotSupported`, and `Never` with Application Server.

This subsection describes the correspondence between the specified transaction attributes and availability of method invocation, and also the timing of invoking callback methods. The callback methods are:

afterBegin methods

`afterBegin` methods are either of the following methods:

- Methods that implement the `afterBegin` method of the `javax.ejb.SessionSynchronization` interface
- Methods in which `@AfterBegin` is specified

beforeCompletion methods

`beforeCompletion` methods are either of the following methods:

- Methods that implement the `beforeCompletion` method of the `javax.ejb.SessionSynchronization` interface
- Methods in which `@BeforeCompletion` is specified

afterCompletion methods

`afterCompletion` methods are either of the following methods:

- Methods that implement the `afterCompletion` method of the `javax.ejb.SessionSynchronization` interface

- Methods in which `@AfterCompletion` is specified

For details on using annotations, see *2.18 Specifications in Session Synchronization annotation*.

(a) Transaction attributes and availability of method invocation

The following table describes the correspondence between transaction attributes of Stateful Session Beans in which `SessionSynchronization` is implemented and the availability of the invocation of business methods or callback methods.

Table 2-19: Availability of method invocation for business methods or callback methods

Transaction attribute	Client transaction availability	Business method invocation	Invocation of callback method of SessionSynchronization
`Supports`	Yes	Y#1	Y#1
	No	Y#1	--
`NotSupported`	Yes	Y#1	--
	No	Y#1	--
`Required`	Yes	Y	Y
	No	Y	Y
`RequiresNew`	Yes	Y	Y
	No	Y	Y
`Mandatory`	Yes	Y	Y
	No	--#2	--
`Never`	Yes	--#2	--
	No	Y#1	--

Legend:

Y: Method is invoked.

--: Method is not invoked.

#1

This operation is unique to Application Server.

#2

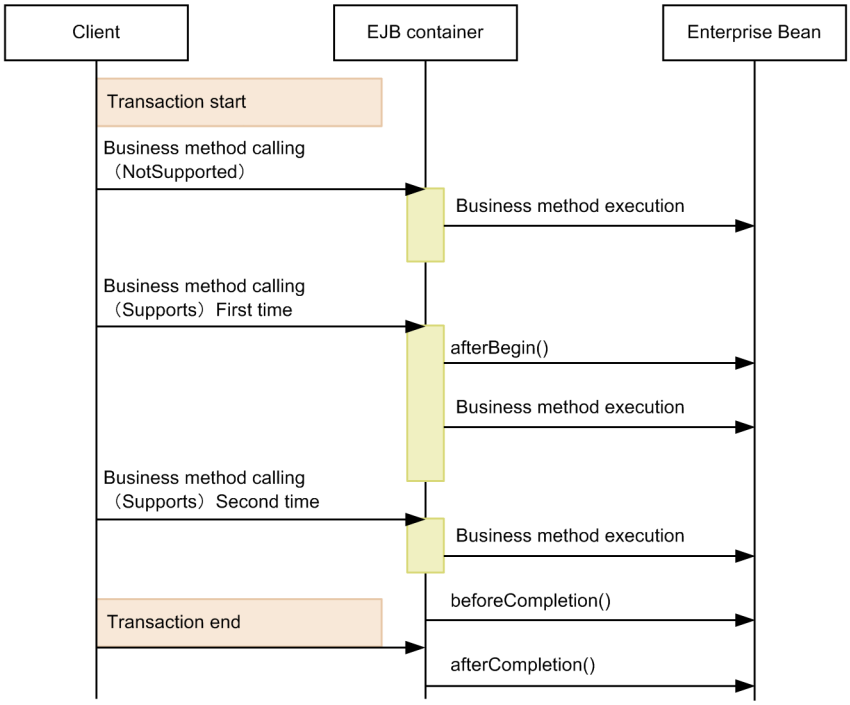
An exception, specified in the EJB specifications, is thrown.

(b) Timing of invoking callback methods

If the business method of the same Enterprise Bean is invoked more than once, the `afterBegin` method that is a callback method is invoked, when the Enterprise Bean first participates in the transaction.

The following is an example of invoking the `afterBegin` method. The following example shows the operation when a business method participating in a transaction is executed after executing the business method that does not participate in the transaction.

Figure 2-13: Example of invoking the callback method



- Enterprise Bean does not participate in the transaction, during the Business method call of initial NotSupported, and hence afterBegin method is not called.
- Enterprise Bean participates in the transaction for the first time during the business method calling In the Supports of the first time, For this reason, afterBegin method is called in this timing.
- afterBegin method is already executed during the business method calling in the Supports of the second time, afterBegin method is not called.

2.7.4 Defining in cosminexus.xml

The definition of the transaction management method of an Enterprise Bean is specified in the <ejb-jar> tag of cosminexus.xml. The tag to be specified will differ for each type of Enterprise Bean to be set up.

The following table describes the definition of the transaction management method of an Enterprise Bean in cosminexus.xml:

Table 2-20: Definition of the transaction management method of an Enterprise Bean in cosminexus.xml

Items	Tag to be specified	Setting contents
Selecting the transaction management method of an Enterprise Bean (BMT or CMT)#	In the case of a Session Bean <session>-<transaction-type> tag In the case of Message Driven Bean <message-driven>-<transaction-type> tag	Specify whether to select Bean (BMT) or Container (CMT).
Transaction attributes allocated to the method (for CMT)	In the case of Session Bean or Entity Bean <assembly-descriptor>-<container-transaction>-<trans-attribute> In the case of Message Driven Bean <message-driven>-<container-transaction>-<trans-attribute>	Specify the transaction attributes allocated to the method.

#

When BMT is selected, the transaction must be controlled with an API (such as a method of the `javax.transaction.UserTransaction` class).

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

2.7.5 Settings in the execution environment

The transaction management method of an Enterprise Bean can also be set up in the execution environment. Specify the settings in a J2EE application imported into a J2EE server, and execute the setup only to specify or change the property of the J2EE application that does not include `cosminexus.xml`.

Set up the J2EE applications in the execution environment using the server management commands and property files. Use the following property files to define the reference mapping:

Table 2-21: Property files used to define the transaction management methods of an Enterprise Bean

Setting target	Attribute files
Session Bean	Session Bean attribute file
Entity Bean	Entity Bean attribute file
Message-driven Bean	Message-driven Bean attribute file

The tags specified in the property files correspond to either DD or `cosminexus.xml`. For details on the `cosminexus.xml` settings, see *2.7.4 Defining in cosminexus.xml*.

2.8 Cache models of an Entity Bean

This section describes the cache models of Entity Beans.

An Entity Bean supports three types of cache models.

The following table describes the organization of this section:

Table 2-22: Organization of this section (Cache models of an Entity Bean)

Category	Title	Reference location
Description	Types of cache models of an Entity Bean	*2.8.1*
Implementation	Defining in cosminexus.xml	*2.8.2*
Setup	Settings in the execution environment	*2.8.3*

Note:
There is no specific description of *Operation* and *Notes* for this functionality.

2.8.1 Types of cache models of an Entity Bean

An Entity Bean supports the following three types of caching of CMP field and status transition of Entity Bean:

- Full caching (commit option A)
- Caching (commit option B)
- No caching (commit option C)

Specify the commit options as attributes (properties) of the Entity Beans included in a J2EE application. For details on the J2EE application settings, see *2.8.2 Defining in cosminexus.xml*.

(1) Full caching (commit option A)

Full caching is the cache model for the Entity Beans of a reference node. When a transaction starts, the data is not read from the database to the Entity Bean instance. As a result, the transaction starts with the Entity Beans in the same state as that during the previous transaction commit.

For example, if the Entity Beans are updated by another J2EE server from the time of the previous transaction commit until the beginning of the transaction, the consistency of the Entity Bean status cannot be maintained.

(2) Caching (commit option B)

Caching is the cache model of the Entity Beans of an update node. When a transaction starts, the data is read from the database to the Entity Bean instance. As a result, the transaction is started with the Entity Beans in the same state as the latest state of the database.

(3) No caching (commit option C)

No caching is the cache model of the Entity Beans of an update node. During transaction commit, the Entity Beans are passivated. When the transaction starts, the Entity Beans are activated once and the data is read from the database to the Entity Bean instance. As a result, the transaction is started with the Entity Beans in the same state as the latest state of the database. This cache model is, therefore, applied while using a large number of Entity Beans in a business application.

2.8.2 Defining in cosminexus.xml

The definition for the commit option of the cache model of an Entity Bean is specified in the `<ejb-jar>` tag of `cosminexus.xml`.

Tag to be specified

`<entity>-<caching-model>` tag

Setting contents

Specify the cache method of the CMP field.

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

2.8.3 Settings in the execution environment

You can also define the commit option of the cache model of an Entity Bean in the execution environment. Specify the settings in the J2EE application imported into a J2EE server. Execute the setup only for setting up or changing the property of a J2EE application that does not include `cosminexus.xml`.

Set up the J2EE applications in the execution environment using the server management commands and the property files. Use the Entity Bean property file to define the commit option of the Entity Bean cache model.

The tags specified in the Entity Bean property file correspond to either the DD or `cosminexus.xml`. For details on the `cosminexus.xml` settings, see *2.8.2 Defining in cosminexus.xml*.

2.9 Managing the Enterprise Bean pool

This section describes how to manage an Enterprise Bean pool.

In the EJB container, you create a pool for each type of Enterprise Beans and manage these pools.

The following table describes the organization of this section:

Table 2-23: Organization of this section (Managing an Enterprise Bean pool)

Category	Title	Reference location
Description	Pooling of Stateless Session Beans	*2.9.1*
	Pooling of Entity Beans	*2.9.2*
	Pooling of Message-driven Beans	*2.9.3*
Implementation	Defining in cosminexus.xml	*2.9.4*
Setup	Settings in the execution environment	*2.9.5*

Note:

There is no specific description of *Operation* and *Notes* for this functionality.

2.9.1 Pooling of Stateless Session Beans

The pooling of Stateless Session Beans is a functionality to pool the Stateless Session Beans based on the access volume from the client. In the EJB container, you create a pool for each Stateless Session Bean and manage these pools. By specifying the maximum value#, and the minimum value, the pooling operation can be customized.

When a J2EE application is started, Stateless Session Beans equivalent to the minimum value are generated and pooled. If the pooled Stateless Session Beans in the method-ready state are accessed from the client, they are executed immediately. The number of pooled Stateless Session Beans will be between the maximum and minimum values, depending upon the access volume from the client.

If the number of client requests for these Stateless Session Beans exceeds the maximum number, the execution of Stateless Session Beans is put on hold until the instance becomes usable.

#

The maximum value of pooling in Stateless Session Beans becomes the maximum number of sessions that can be established concurrently by the client.

2.9.2 Pooling of Entity Beans

The pooling of Entity Beans is a functionality to pool the Entity Beans based on the access volume from the client. In the EJB container, you create a pool for each Entity Bean and manage these pools. By specifying the maximum and minimum values, the pooling operation can be customized.

When a J2EE application is started, Entity Beans equivalent to the minimum value are generated and pooled. The number of pooled Entity Beans will be between the maximum and minimum values, depending upon the access volume from the client.

The pooled Entity Beans are in two states, namely the ready state and the pool state.

Ready-state Entity Beans

The data is read from the database into the instance and has the identity as an Entity Bean. The Entity Beans in the ready state are already in the state in which they can be executed when accessed from the client.

Pool-state Entity Beans

The data is not read from the database into the instance and does not have the identity as an Entity Bean. The Entity Beans in the pool state are activated once, change to ready state, and then to the executable state.

When the Entity Beans in the ready state become large in number, several of them are passivated and change to the pool state. At this point, however, from among the Entity Beans in the ready state, those engaged in a transaction are not passivated.

2.9.3 Pooling of Message-driven Beans

The pooling of Message-driven Beans is the functionality used for pooling the Message-driven Beans based on the number of messages. You can specify the maximum value to customize the pooling operation.

Based on the requests from JMS, the instances equal to the value specified in the maximum number of instances within the pool are created during the deployment.

The instances are destroyed according to the requests from JMS.

2.9.4 Defining in cosminexus.xml

The definition for the pool management of an Enterprise Bean is specified within the <ejb-jar> tag of cosminexus.xml. The specified tag is different for each type of Enterprise Bean to be set up.

The following table describes the definition of pool management of an Enterprise Bean in cosminexus.xml:

Table 2-24: Definition of pool management of an Enterprise Bean in cosminexus.xml

Items	Tag to be specified	Setting contents
Pooling of Stateless Session Beans	<session>-<stateless>-<pooled-instance> tag	Specify the maximum value and the minimum value of the number of instances to be pooled.
Pooling of Entity Beans	<entity>-<pooled-instance> tag	
Pooling of Message-driven Beans	<message>-<pooled-instance> tag	

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

2.9.5 Settings in the execution environment

You can also define the pool management of an Enterprise Bean in the execution environment. Specify the settings in the J2EE applications imported into a J2EE server. Execute the settings only to specify or change the property of the J2EE application that does not include cosminexus.xml.

Set up the J2EE applications in the execution environment using the server management commands and the property files. Use the following property files to define the reference mapping:

Table 2-25: Property files used for defining the transaction management method in an Enterprise Bean

Setting target	Attribute files
Session Bean	Session Bean attribute file
Entity Bean	Entity Bean attribute file
Message-driven Bean	Message-driven Bean attribute file

The tags specified in the property files correspond to either the DD or cosminexus.xml. For details on the cosminexus.xml settings, see *2.9.4 Defining in cosminexus.xml*.

2.10 Controlling the access to the Enterprise Beans

This section describes how to control access to the Enterprise Beans.

In the EJB container, the access to an Enterprise Bean from the client can be controlled with the security management functionality of the J2EE service.

The following table describes the organization of this section:

Table 2-26: Organization of this section (Controlling access to an Enterprise Bean)

Category	Title	Reference location
Description	Preventing access control to an Enterprise Bean	*2.10.1*
Setup	Settings in the execution environment	*2.10.2*

Note:
There is no specific description of *Implementation*, *Operation* and *Notes* for this functionality.

For details on the security management functionality other than controlling the access to an Enterprise Bean, see *9. Security Management* in the *uCosminexus Application Server Security Management Guide*.

2.10.1 Preventing access control to an Enterprise Bean

In the EJB container, the access to an Enterprise Bean from the client can be controlled with the security management functionality of the J2EE service. As per the default operations of a J2EE server, even if an application does not use the functionality of access control, the basic processing for access control is performed.

For this, if you use the option for preventing the access control to an Enterprise Bean (Option for preventing access control to an Enterprise Bean), you can prevent the checking of the execution permission at the invocation source when invoking business methods. If this check is prevented, the processing for access control is not implemented at all in the EJB container, and therefore, the invocation processing of the Enterprise Bean business methods will become lighter. Hitachi, therefore, recommends that you use the prevention option when the functionality for access control is not used.

However, note that if an Enterprise Bean using the access control functionality of a J2EE server is invoked from another J2EE server that uses the option for preventing access control, the operation results in an error.

Customize the properties of a J2EE server to specify the settings for preventing access control to an Enterprise Bean.

2.10.2 Settings in the execution environment

When using access control to an Enterprise Bean, you must set up a J2EE server.

The definition for disabling the control of access to an Enterprise Bean in the Easy Setup definition file is specified in the `<configuration>` tag of the logical J2EE server (`j2ee-server`). Specify the definition in the system property for the JavaVM of a J2EE server in the Easy Setup definition file.

The parameter name and the settings are as follows:

Parameter to be specified
ejbserver.container.security.disabled

Setting contents
Specify whether to disable the functionality for controlling access to an Enterprise Bean.

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*.

2.11 Setting up a timeout in the EJB container

This section describes how to set up a timeout in the EJB container.

The following types of timeout values can be set in the EJB container:

- Timeout of a Stateful Session Bean
- Timeout of the EJB objects in the Entity Beans
- Timeout of RMI-IIOP communication
- Timeout in awaiting instance acquisition

The following table describes the organization of this section:

Table 2-27: Organization of this section (Setting up a timeout in the EJB container)

Category	Title	Reference location
Description	Types of timeouts	*2.11.1*
	Timeout of a Stateful Session Bean	*2.11.2*
	Timeout of the EJB objects in the Entity Beans	*2.11.3*
	Timeout in awaiting instance acquisition	*2.11.4*
	Timeout of RMI-IIOP communication	*2.11.5*
Implementation	Defining in cosminexus.xml	*2.11.6*
	Implementing a timeout for RMI-IIOP communications	*2.11.7*
Setup	Settings in the execution environment	*2.11.8*
Notes	Precautions during setup of a communication timeout	*2.11.9*

Note:
There is no specific description of *Operation* for this functionality.

2.11.1 Types of timeouts

The following table describes the types of timeouts that can be set in the EJB container, the overview of timeout functionality, and the references:

Table 2-28: Types of timeouts

Types of timeouts	Overview of timeout
Timeout of a Stateful Session Bean	Specify a timeout period from the point of time a Stateful Session Bean was last accessed. Delete the Stateful Session Beans for which the specified time period has elapsed.
Timeout of the EJB objects in the Entity Beans	Specify the timeout period in the EJB objects connected to a passivated Entity Bean. Delete the Stateful Session Beans for which the specified time period has elapsed.
Timeout of RMI-IIOP communication	Specify the timeout period for the communication between the EJB client and the CORBA Naming Service and between the EJB client and the Enterprise Beans.
Timeout in awaiting instance acquisition	In the Stateless Session Beans and the Entity Beans, specify the timeout period for awaiting instance acquisition during the receipt of a request.

2.11.2 Timeout of a Stateful Session Bean

Timeout of Stateful Session Beans is a functionality that monitors the time that has elapsed since the Stateful Session Bean was last accessed and uses a timer to delete the Stateful Session Beans not accessed from the client even after the elapse of the specified time. You can specify the timeout period in the EJB container. The Stateful Session Beans engaged in a transaction, however, are not deleted.

If a Stateful Session Bean that has been deleted due to a timeout is invoked, the following exceptions are thrown based on the type of the interface:

- In the case of a remote component interface
  Exception `java.rmi.NoSuchObjectException` is thrown.
- In the case of a local component interface
  Exception `java.ejb.NoSuchObjectLocalException` is thrown.
- In the case of a business interface
  Exception `javax.ejb.NoSuchEJBException` is thrown. If, however, the business interface inherits `java.rmi.Remote`, exception `java.rmi.NoSuchObjectException` is thrown.

The timeout settings for the Stateful Session Bean are specified as attributes (properties) of the Session Beans included in a J2EE application. For details on the settings, see *2.11.6 Defining in cosminexus.xml*.

2.11.3 Timeout of the EJB objects in the Entity Beans

Timeout of the EJB objects is a functionality for deleting the EJB objects, from among those connected to a passivated Entity Bean, for which the specified time period has elapsed. You can specify the timeout period in the EJB container. Note that if an Entity Bean that deletes the EJB objects due to a timeout is invoked, an exception (`java.rmi.NoSuchObjectException`) occurs.

The EJB local objects of the local interface are also similarly deleted due to a timeout. If an Entity Bean contains both EJB objects and EJB local objects, when the passivated Entity Bean is not accessed within the specified time from either of the interfaces, the EJB objects and the EJB local objects are deleted. If the deleted EJB local objects are invoked, an exception (`javax.ejb.NoSuchObjectLocalException`) occurs.

The timeout settings for the EJB objects of an Entity Bean are specified as attributes (properties) of the Entity Beans included in a J2EE application. For details on the settings, see *2.11.6 Defining in cosminexus.xml*.

2.11.4 Timeout in awaiting instance acquisition

The EJB container allocates an instance when it receives a request. During the allocation, if the maximum value is specified in the instance pool (the "method-ready" pool of the Stateless Session Bean and the "pool" pool of the Entity Bean), the acquisition of an instance is awaited if another request is being processed in all instances. A timeout can be specified for this waiting time.

If you specify a timeout value, an exception is returned to the client when the instance cannot be acquired within the specified time.

The timeout settings for awaiting instance acquisition are specified as attributes (properties) of the Session Beans or the Entity Beans included in a J2EE application. For details on the settings, see *2.11.6 Defining in cosminexus.xml*.

2.11.5 Timeout of RMI-IIOP communication

A timeout value can be set for communication between the EJB client and the CORBA Naming Service, and between the EJB client and the Enterprise Beans. Furthermore, if you deploy CTM in between the EJB client and a J2EE server, you can set up a communication timeout between the EJB client and the CTM and between the CTM and a J2EE server.

To set up a timeout for an RMI-IIOP communication, use the request timeout functionality of Cosminexus TPBroker that is used as the RMI-IIOP communication platform by the EJB container.

Depending on the range of the timeout to be specified, you specify a timeout either using the property of the definition file or using the APIs that Application Server provides.

(1) RMI-IIOP communications for which a timeout can be specified

The following figure shows the RMI-IIOP communications (for which a timeout can be specified) working with/ without CTM, and also explains the range of timeout:

Figure 2-14: Communications for which a timeout can be specified (when CTM linkage exists)

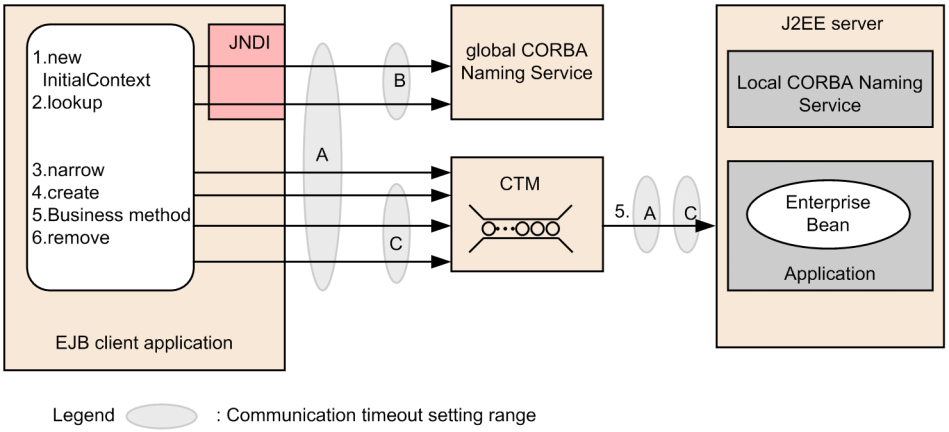
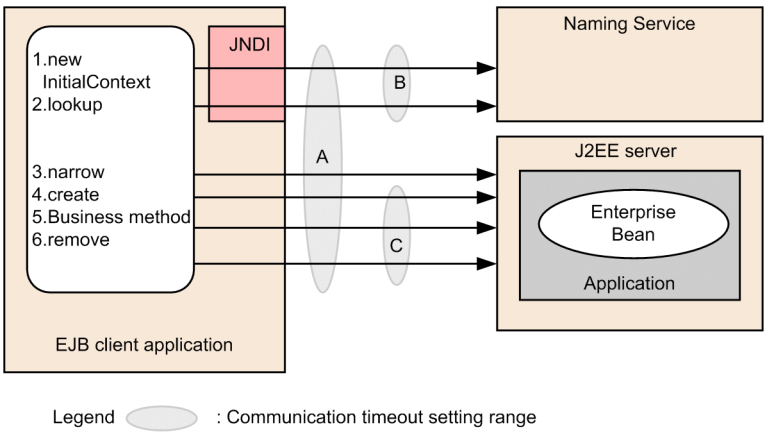


Figure 2-15: Communications for which a timeout can be specified (when CTM linkage does not exist)



The timeout for RMI-IIOP communications can be set up at three locations that are indicated by A, B, and C in the figure.

- In the case of A

  The timeout is enabled for all the RMI-IIOP communications from steps 1 to 6 in *Figure 2-14* and *Figure 2-15*. The timeout is set up in the definition file. For details on how to set up the timeout in the definition file, see *2.11.8 (1) Setting up a timeout for the RMI-IIOP communication (setting up a J2EE server and EJB client applications)*.

- In the case of B

  The timeout is enabled for the communication with the CORBA Naming Service indicated in steps 1 and 2 in *Figure 2-14* and *Figure 2-15*. The timeout is set up in the definition file. For details on how to set up a timeout in the definition file, see *2.11.8 (1) Setting up a timeout for the RMI-IIOP communication (setting up a J2EE server and EJB client applications)*.

- In the case of C

  The timeout is enabled for the API communication indicated in steps 4 to 6 in *Figure 2-14* and *Figure 2-15*.

  In an EJB client application, specify the timeout period of API communication until create - business method - remove is set by using the API (methods of `RequestTimeoutConfigFactory` class and `RequestTimeoutConfig` class), during development of the application. For details on how to set up using the API, see *2.11.7 Implementing a timeout for RMI-IIOP communications*.

(2) Specification range and timing of communication timeout

Use this functionality to specify the communication timeout for ORB. In other words, the communication timeout is set for all the RMI-IIOP communications below ORB.

The time of setting a timeout value is during the execution of the first `new javax.naming.InitialContext()` after invocation of the client.

Even if you do not use the CORBA Naming Service, when you use this functionality, execute `new javax.naming.InitialContext()` at the beginning of client processing.

(3) Processing of the client when a communication timeout occurs

If a response is not returned for the request from the client within the value specified in the properties, the request is cancelled as a timeout. When a communication timeout occurs due to this functionality, an `java.rmi.RemoteException` (`org.omg.CORBA.TIMEOUT`) exception is thrown. In a client using this functionality, the occurrence of this exception must be considered during invocation of business methods of the Enterprise Beans.

2.11.6 Defining in cosminexus.xml

Of the timeout settings in the EJB container, the definition of the timeout of a Stateful Session Bean, timeout of EJB objects of an Entity Bean, or the timeout in awaiting instance acquisition is specified in the `<ejb-jar>` tag of `cosminexus.xml`. The tag to be specified will differ for each type of the target Enterprise Bean.

The following table describes the definition of the timeout of the EJB container in `cosminexus.xml`:

Table 2-29: Definition of the timeout of EJB container in cosminexus.xml

Items	Tag to be specified	Setting contents
Timeout of a Stateful Session Bean	`<session>-<stateful>-<removal-timeout>` tag	Specify the time period for maintaining the inactive status until the session is deleted.
Timeout of the EJB objects in the Entity Beans	`<entity>-<entity-timeout>` tag	Specify the time period for existence of the EJB object.
Timeout in awaiting instance acquisition	In the case of a Session Bean `<session>-<stateless>-<instance-timeout>` tag In the case of an Entity Bean `<entity>-<instance-timeout>` tag	Specify the timeout period for acquiring instances.

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

2.11.7 Implementing a timeout for RMI-IIOP communications

You can set up a timeout for the RMI-IIOP communications with an API.

The timeout indicated by C in *Figure 2-14* and *Figure 2-15* can be set up by using APIs. For specifying a timeout by using APIs, use the APIs of the `com.hitachi.software.ejb.ejbclient` package. For details on the functionality and syntax of APIs, see *4. APIs Used in EJB Client Applications* in the *uCosminexus Application Server API Reference Guide*.

The timeout value for an RMI-IIOP communication is the value specified in the property when the first `InitialContext` is generated after starting up the client process. Even when you do not use a naming service, you must generate `InitialContext` for setting up a timeout in the RMI-IIOP communication.

The communication timeout for a naming service is the value that is specified in the property during an API invocation of JNDI, such as `InitialContext` creation and lookup.

2.11.8 Settings in the execution environment

Of the timeouts in an EJB container, the timeout for an RMI-IIOP communication can be set up in a J2EE server that is the client process or in the EJB client applications.

You can specify the timeout of a Stateful Session Bean, the EJB objects of an Entity Bean, or the timeout in awaiting instance acquisition in a J2EE application. Reference while you set up or change the property of the J2EE application that does not include `cosminexus.xml`.

(1) Setting up a timeout for the RMI-IIOP communication (setting up a J2EE server and EJB client applications)

You can set up a timeout for A or B in *Figure 2-14* and *Figure 2-15*.

For setting the timeout as a property, the setup method will differ depending on the Enterprise Bean from which the method is invoked (forms of EJB clients). Specify the settings as properties of a J2EE server or the EJB client application of the caller (EJB client).

(a) When the EJB client is in the form of an Enterprise Bean, JSP, or servlet

Set up the timeout in a J2EE server in which the client-side Enterprise Bean, JSP, or servlet is running. Specify the settings of a J2EE server in the Easy Setup definition file.

Specify the following parameter in the `<configuration>` tag of the logical J2EE server (`j2ee-server`) of the Easy Setup definition file.

Parameter to be specified

ejbserver.rmi.request.timeout

Setting contents

Specify the timeout period for the communication between the client and the server in the RMI-IIOP communication.

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*.

(b) When the EJB client is in the form of an EJB client application

For the EJB client application, specify the timeout value as an enabled property during the execution of the EJB client application.

Specify the following key in `usrconf.properties` (user property file for Java applications):

Key to be specified

`ejbserver.rmi.request.timeout` key

Setting contents

Specify the timeout period for the communication between the client and the server in the RMI-IIOP communication.

For details on `usrconf.properties` and keys, see *14.3 usrconf.properties (User property file for Java applications)* in the *uCosminexus Application Server Definition Reference Guide*.

(2) Definition of the timeout of a Stateful Session Bean, timeout of EJB objects of an Entity Bean, or timeout in awaiting instance acquisition

The definition of the timeout of a Stateful Session Bean, timeout of EJB objects of an Entity Bean, or the timeout in awaiting instance acquisition can also be set up in the execution environment. Specify the settings in the J2EE application imported into a J2EE server. Execute the setup only to specify or change the property of the J2EE application that does not include `cosminexus.xml`.

Set up the J2EE application in the execution environment using the server management commands and the property files. Use the following property files to define the reference mapping:

Table 2-30: Property files used to define the timeout of a Stateful Session Bean, timeout of the EJB objects of an Entity Bean, or the timeout in awaiting instance acquisition

Setting target	Attribute files
Session Bean	Session Bean attribute file
Entity Bean	Entity Bean attribute file

The tags specified in the property files correspond to the DD or `cosminexus.xml`. For details on the `cosminexus.xml` settings, see *2.11.6 Defining in cosminexus.xml*.

2.11.9 Precautions during setup of a communication timeout

The common precautions for setting a communication timeout are explained below:

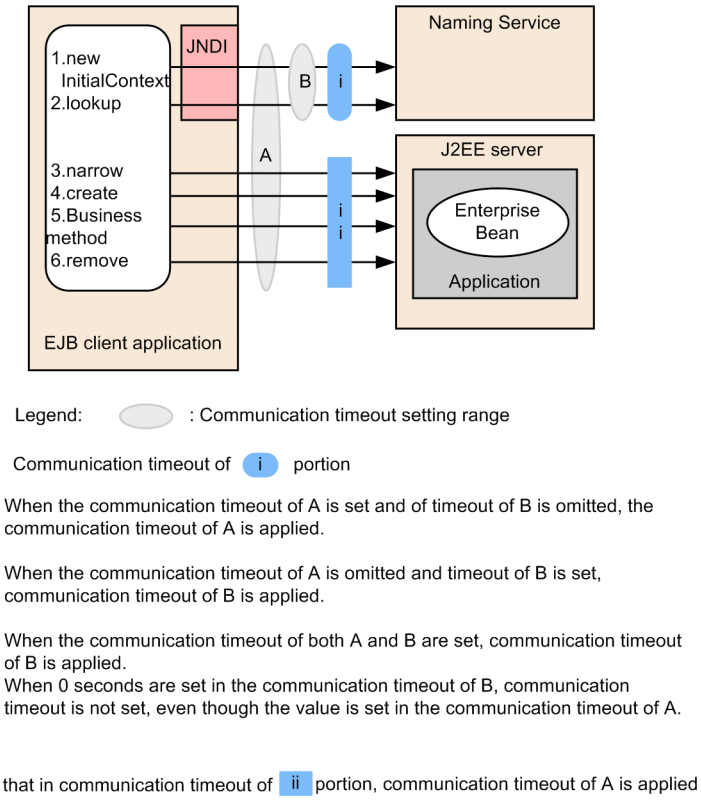
- **Precautions for when the definition of communication timeout overlaps**

  If a communication timeout is set up for both A and B in *Figure 2-14* and *Figure 2-15*, and a communication timeout is set up for A and C, you perform the operation as follows:

  - **Operations when a communication timeout is specified for both A and B**

    When a communication timeout is specified for both A and B, the operations will be as shown in the figure below:

    Figure 2-16: Operations when a communication timeout is specified for both A and B



  - **Operations when a communication timeout is set for both A and C**

    When a communication timeout is specified for both A and C, the communication timeout setting for C is enabled.

- **Precautions during client implementation**

  Along with the RMI-IIOP communication timeout and the naming service communication timeout, if a response is not returned within the specified time for a request sent from the client, the corresponding request will be cancelled as the timed out request. In such a case, the exceptions `java.rmi.RemoteException` (such as

`org.omg.CORBA.TIMEOUT`) and `javax.naming.NamingException` will be thrown. For implementing a client that uses a communication timeout, give consideration to the fact that these exceptions might occur during invocation of the methods in Enterprise Beans or in JNDI APIs.

- **Precautions concerning operation at the server side after the occurrence of a timeout**

  When a request from the client reaches the server (naming service and Enterprise Bean) and a timeout occurs during the processing at the server side, an exception is returned to the client. However, the processing continues normally at the server side even after the occurrence of a timeout, and therefore, the instances of the Enterprise Bean are not destroyed and the resources such as the resource connection are not released.

2.12 Timer Service functionality

This section explains the functionality of the Timer Service.

The Timer Service is functionality by which the EJB container invokes an Enterprise Bean at a specified time, elapsed time, or interval.

You can use the Timer Service in version EJB 2.1 or later. For details on the functionality that you can use in each version, see the EJB specifications.

The following table describes the organization of this section:

Table 2-31: Organization of this section (Functionality of the Timer Service)

Category	Title	Reference location
Description	Overview of the Timer Service	*2.12.1*
	Operation during the generation of an EJB timer and execution of a callback	*2.12.2*
	Automatic generation of an EJB timer	*2.12.3*
	Deleting the EJB timer	*2.12.4*
	Functionality for operating the Timer Service	*2.12.5*
	Operations of the EJB timer and callback	*2.12.6*
Implementation	Implementing an application using the Timer Service	*2.12.7*
	Precautions during the implementation of the Timer Service	*2.12.8*
Setup	Settings in the execution environment	*2.12.9*
Notes	Precautions during the use of the Timer Service	*2.12.10*

Note:
There is no specific description of *Operation* for this functionality.

2.12.1 Overview of the Timer Service

The *Timer Service* is functionality for invoking Enterprise Beans at a specified time, elapsed time, or interval. This functionality is provided in the EJB container. If you use the Timer Service, the processing wherein the time is specified, such as batch processing with the time when the machine load is low specified and daily processing at a fixed interval, can be executed easily.

This section explains the contents of the timeout value that can be set in the Timer Service, the operations of the EJB timer used to set timeout, and the operations of the Timer Service.

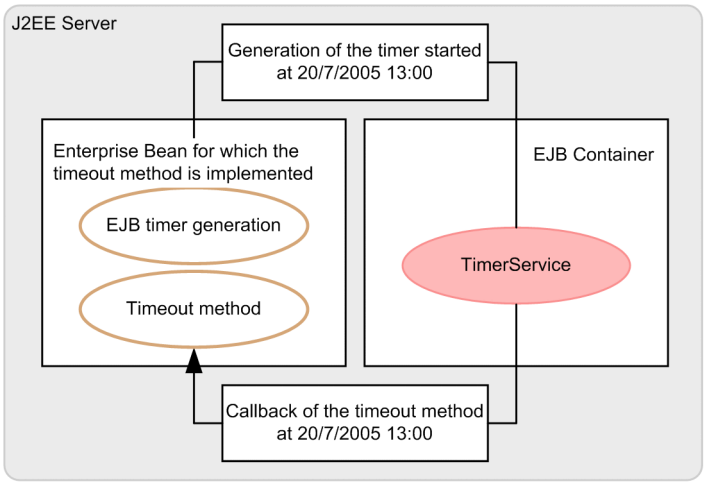
This subsection explains the timeout value that can be specified in the Timer Service, the support range of the Timer Service, as well as transaction management by the Timer Service.

(1) Timer Service and the EJB timer

Use the API defined in Java EE to operate the Timer Service from the Enterprise Bean. Generate an *EJB timer* to execute a processing for which a time is specified. Specify the time for timeout in the EJB timer. The generated EJB timer is managed in the EJB container and once the time for timeout is reached, the Enterprise Bean method is called back by the EJB container. The method that is called back at this point is called the *timeout method*.

The following figure shows an overview of the Timer Service processing:

Figure 2-17: Overview of the Timer Service operation



(2) Types of EJB timers

The following two types can be set as the types of EJB timers:

- single-event timer

  The single-event timer is the EJB timer for executing the timeout method only once.

  The timeout is set either by specifying the time of executing the timeout method or by specifying the time period from invocation of the method for generating the EJB timer until execution of the timeout method.

  You can generate this timer using the following methods of the `javax.ejb.TimerService` interface:

  - `createTimer`(long duration, Serializable info) method
  - `createTimer`(Date expiration, Serializable info) method
  - `createSingleActionTimer` method

- interval timer

  The interval timer is the EJB timer for repeated execution of the timeout method at fixed intervals.

  The timeout is set either by specifying the time of executing the first timeout method or by specifying the time period from invocation of the method for generating the EJB timer until execution of the first timeout method. Apart from this setting, specify the timeout interval for executing the timeout method from second time onwards. This is the interval from one timeout to the next.

  You can generate the timer using the following methods of the javax.ejb.TimerService interface:

  - `createTimer`(long `initialDuration`, long `intervalDuration`, Serializable info) method
  - `createTimer`(Date `initialExpiration`, long `intervalDuration`, Serializable info) method
  - `createIntervalTimer` method

- calendar-based timer

  The calendar-based timer is the EJB timer to execute the timeout method on the date and time specified in the calendar format. You can specify single or multiple values. Also you can specify a wild card or range. For details on the method of specifying the calendar-based timer, see *2.12.7(3) Method for specifying a schedule in calendar format*.

  You can generate the `javax.ejb.TimerService` interface with the following method:

  - createCalendarTimer method

  You can also create the method with the `@Schedule` annotation. For details, see *2.12.3 Automatically generating an EJB timer*.

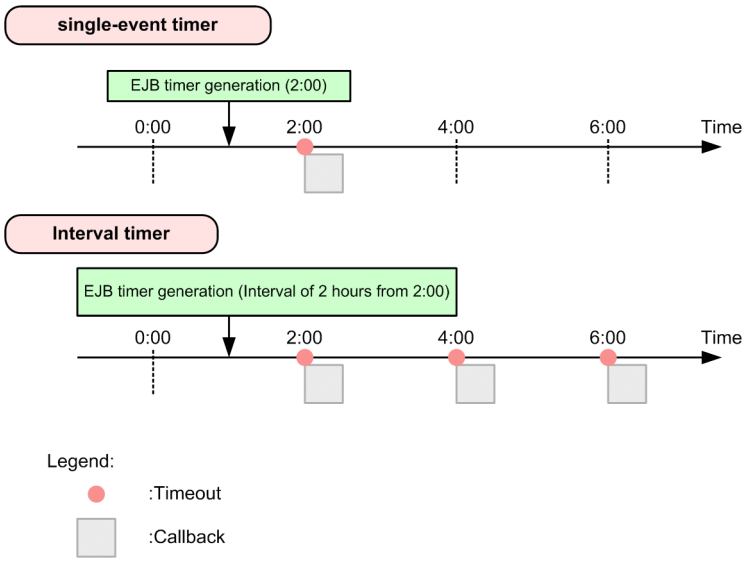
The following table describes the types of timers that can be set as an EJB timer as well as example settings:

Table 2-32: Types of timers that can be set as EJB timer and example settings

Timer type	Example settings	Description
single-event	2006/4/15 12:00	Execute the timeout method only once at 2006/4/15 12:00.
	After 24 hours	Execute the timeout method only once, after 24 hours from the generation of EJB timer.
Interval	2006/4/1 24 hours interval from 12:00	Execute the first timeout method at 01.04.06 12:00. After this, repeat the timeout method at an interval of 24 hours.
	10 hours interval after 24 hours	Execute the first timeout method after 24 hours from the generation of EJB timer. After this, repeat the timeout method at an interval of 10 hours.
calendar-based	First day of every month 12:00	Execute the timeout method on the first day of every month at 12:00

The following figure shows the operations of the single-event timer and interval timer from among the EJB timers. This figure shows the operations of the single-event timer that calls back the timeout method only once at 2:00 and the interval timer that calls back the timeout method at an interval of two hours, starting from 2:00.

Figure 2-18: Operations of the single-event timer and interval timer



(3) Support range of the Timer Service

The following table describes the support status of the Timer Service functionality defined in the Java EE specifications:

Table 2-33: Support status of the Timer Service functionality defined in the Java EE specifications

Timer Service functionality defined in the Java EE specifications	Support status
Transaction	Y
EJB timer persistence	--[#]
Acquiring the `TimerService` objects (DI, JNDI lookup, EJBContext)	Y
Specifying the timeout method (annotation, `TimedObject` implementation)	Y
Specifying the timeout methods (specifying by DD)	--

Legend:

Y: Can be used

--: Cannot be used

#

Specification of the persistent attribute of the `@Schedule` annotation is also not effective.

If a J2EE server is restarted due to a failure, the EJB timer that was in use before restart is not inherited. For details on automatic generation of the EJB timer during invocation of a J2EE server, see *2.12.3 Automatically generating an EJB timer*.

The following table describes the support status of Timer Service functionality for various types of Enterprise Beans:

Table 2-34: Support status of Timer Service functionality for various types of Enterprise Beans

Timer Service functionality	Message-driven Bean	Session Bean			Entity Bean
		Stateful Session Bean	Stateless Session Bean	Singleton Session Bean	
Acquiring the `TimerService` objects	N	--	Y	Y	N
Operating the objects related to the Timer Service (`TimerService`, `Timer`, `TimerHandle`)	N	Y	Y	Y	N
Automatic generation of the timer with annotation	N	--	Y	Y	N

Legend:

Y: Can be used

N: Cannot be used

--: Cannot be used (J2EE specifications)

(4) Transaction management by the Timer Service

The Timer Service supports transactions. Specifically, the generation of an EJB timer, deletion of the EJB timer, and the timeout method comply with the transactions. The transaction management of the timeout method is explained below:

For details on generating the EJB timer, see *2.12.2 Operation during the generation of an EJB timer and execution of a callback*, and for details on deleting the EJB timer, see *2.12.4 Deleting the EJB timer*.

(a) Transaction attributes that can be set in the timeout method

In the timeout method, you can select either BMT or CMT for transaction management.

If CMT is selected, the transaction attributes that can be specified in the timeout method are as follows:

- `Required` attribute
- `RequiresNew` attribute
- `NotSupported` attribute

If an attribute other than those described above is specified, the J2EE applications fail to start.

(b) Transaction management for callback of the timeout method

In the timeout method, when the `Required` attribute or the `RequiresNew` attribute is specified in CMT, callback will be retried if the transaction rolls back during callback of the timeout method. For details on retrying callback, see *2.12.5(2) Retrying callback of the timeout method*.

2.12.2 Operation during the generation of an EJB timer and execution of a callback

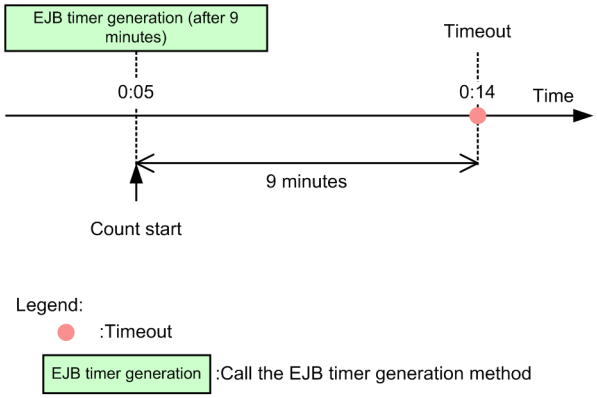
Generate an EJB timer to execute processing at the time specified by the Timer Service. To stop the execution of the processing by the EJB timer, delete the EJB timer. The time for generating and deleting the EJB timer depends upon the type of the EJB timer and upon whether processing is being executed under transaction management. The time for generating and deleting the EJB timer, and the operations of timeout method callback during generation and deletion of the EJB timer are explained below:

During the generation of the EJB timer, a single EJB timer is generated by the timer generation method (`createTimer` method of the `javax.ejb.TimerService` object). If an EJB timer is generated, a timeout occurs at the specified time and the timeout method is called back.

(1) Counting of timeout time

During the generation of the EJB timer, in the case of an EJB timer that specifies the time from the invocation of the EJB timer until the execution of the first timeout method, the counting of time starts from the point the method for generating the EJB timer is invoked. The following figure shows the generation of the EJB timer and the start of the time count:

Figure 2-19: Generating EJB timer and starting the time count



(2) Timing of generating the EJB timer and executing callback of the timeout method

The timing of generating an EJB timer and callback of the timeout method depends upon whether the EJB timer is generated under transaction management.

When an EJB timer is generated under transaction management

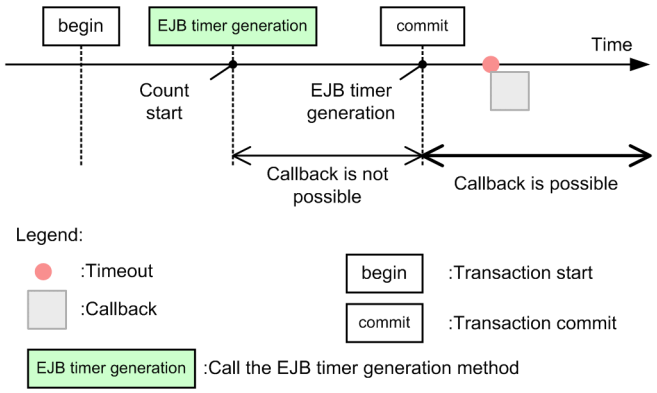
The EJB timer is generated when a transaction is committed.

The timing of executing callback of the timeout method is different in the case of an EJB timer wherein the time for timeout is reached after the transaction is committed, and in the case of an EJB timer wherein the time for timeout is reached before the transaction is committed. Note that if a transaction rolls back, the generation of the EJB timer is cancelled.

- **In the case of an EJB timer, wherein the time for timeout is reached after the transaction is committed**

  The timeout method is called back as per the specified time. The following figure shows the generation of the EJB timer and the execution of callback:

Figure 2-20: Generating the EJB timer and executing callback (in the case of an EJB timer, wherein the timer for timeout is reached after the transaction is committed)

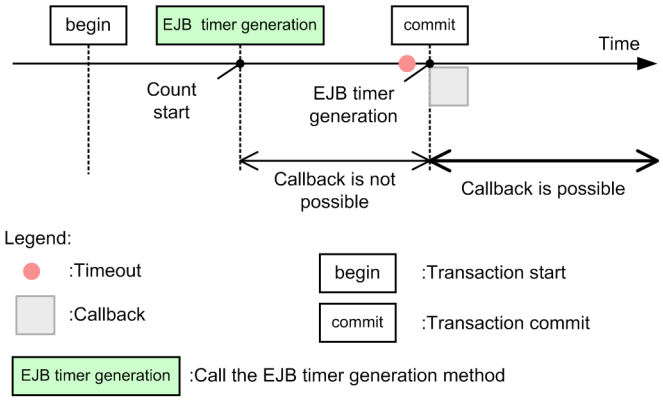


In this figure, the time count is started from the time the method for generating the EJB timer is invoked until the time for a timeout is reached and the timeout method is called back at the specified time. The EJB timer is generated during transaction commit.

- **In the case of an EJB timer, wherein the time for a timeout is reached before the transaction is committed**

  The timeout method is called back immediately after the transaction is committed. The following figure shows the generation of the EJB timer and the execution of callback:

  Figure 2-21: Generating the EJB timer and executing callback (in the case of an EJB timer, wherein the timer for timeout is reached before the transaction is committed)



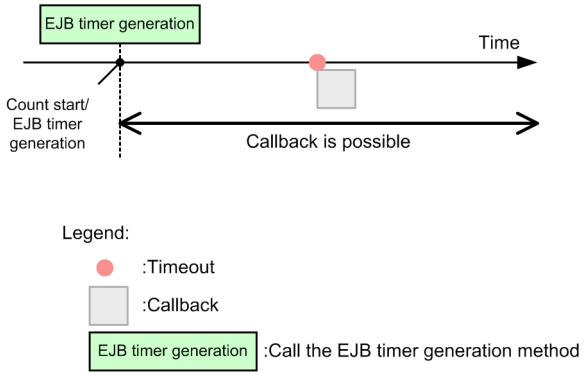
  In this figure, the time count is started from the time the method for generating the EJB timer is invoked until the time for the timeout is reached before the transaction commits. The EJB timer is not generated until the transaction is committed, therefore, the timeout method is not called back even when the time for the timeout is reached. Callback is executed immediately after the transaction is committed.

When an EJB timer is not generated under transaction management

The EJB timer is generated immediately after the method for generating the EJB timer is invoked.

The following figure shows generation of an EJB timer that is not under transaction management:

Figure 2-22: Generating the EJB timer and executing callback (when an EJB timer is not generated under transaction management)



In this figure, the EJB timer is generated immediately after the method for generating the EJB timer is invoked. After this, the timeout method is called back at the specified time.

2.12.3 Automatically generating an EJB timer

This section explains the method of automatically generating an EJB timer. The methods of automatically generating an EJB timer are as follows:

- Method of specifying the `@Schedule` annotation

  This method specifies the `@Schedule` annotation in an Enterprise Bean that generates the EJB timer. You can use this method in EJB 3.1 or later .
- Servlet method

  This method implements the process of automatically generating the EJB timer in the servlet.
- Management event method

  This method uses a Management event to automatically generate an EJB timer. Specify to automatically generate an EJB timer as a Management event that occurs when the message is output when a J2EE server is started.

  You need not modify the J2EE application if the EJB client that invokes the Enterprise Bean generating the EJB timer is created.

When you use the method of specifying the `@Schedule` annotation and the `Servlet` method, you can automatically generate an EJB timer on the startup of an application. You can also automatically generate the EJB timer on the startup of the J2EE server if you enable the automatic start of an application. When you use the Management event method, you can automatically generate the EJB timer on startup of the J2EE server.

(1) Automatically generating an EJB timer on J2EE server startup (Method of specifying the `@Schedule annotation`)

You use this method in the case of a Stateless Session Bean and a Singleton Session Bean of EJB 3.1. When an application starts, the EJB container automatically generates a timer based on the `@Schedule` annotation definition specified in the Session Bean class.

You can generate a timer by specifying the `@Schedule` annotation in the method of a Bean class that executes processes with the timer, or its parent class. When starting multiple timers, specify the `@Schedule` annotation.

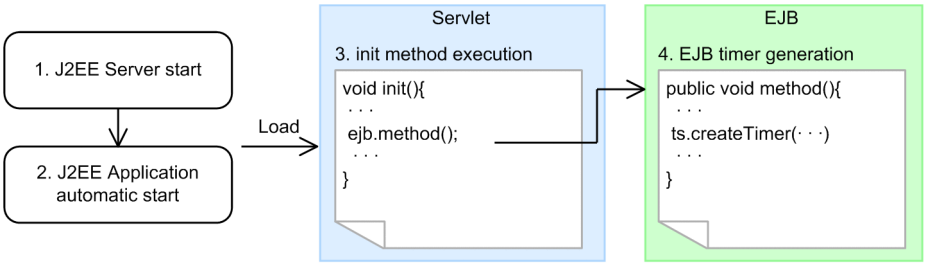
(2) Automatically generating the EJB timer when starting a J2EE server (servlet method)

This section explains the flow of automatic generation of the EJB timer and its setup when the process of automatically generating an EJB time is implemented in the servlet.

(a) Flow of automatic generation of the EJB timer

The flow in which the servlet method automatically generates an EJB timer is shown below:

Figure 2-23: Flow of automatic generation of the EJB timer (servlet method)



1. Re-start the J2EE server.
2. The J2EE application starts automatically.
   The J2EE application that was running when the J2EE server is stopped, starts automatically (the application must be enabled to start automatically).
3. The `init` method is executed.
   When the J2EE application starts, the servlet is loaded, and the `init` method is executed. The `init` method creates the Enterprise Bean and invokes the business method.
4. The EJB timer is generated.
   In the business method of the Enterprise Bean, the EJB timer is generated from the Timer Service object.

(b) Setup for automatically generating the EJB timer

Specify the following settings to automatically generate an EJB timer in the servlet:

1. Create the business method that generates the EJB timer in the Enterprise Bean.
2. Create the servlet.
   Specify to invoke the business method created in step 1 by the `init` method.
3. Load the servlet that is created in step 2 when the J2EE application starts.
   Specify a value of 0 or more as the value of the `<load-on-startup>` tag of the servlet created in step 2 in the DD (`web.xml`).
4. When a J2EE server is started, enable the application to start automatically.

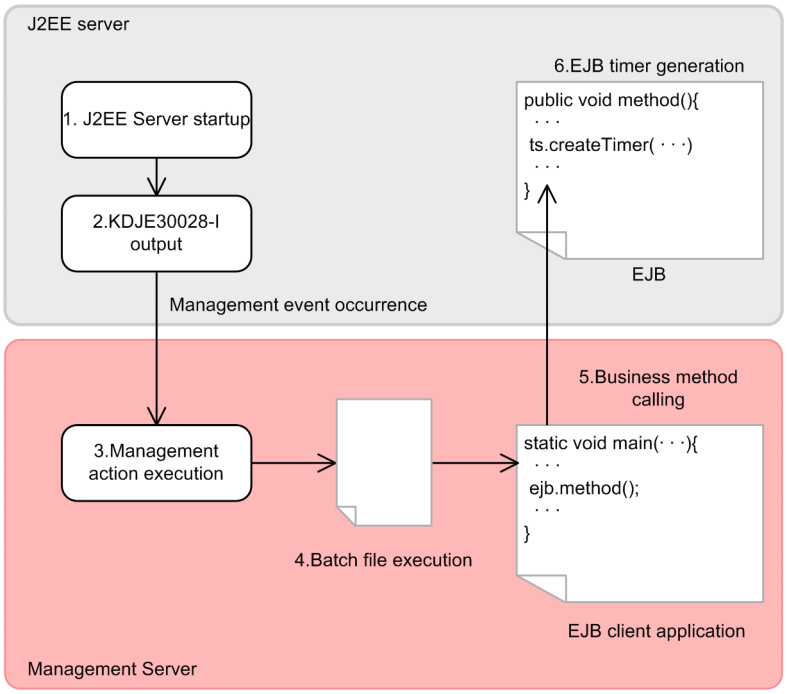
(3) Automatically generating the EJB timer when starting a J2EE server (Management event method)

This section explains the flow of automatic generation of the EJB timer and its setup when the process of automatically generating an EJB timer is set as a Management event.

(a) Flow of automatic generation of the EJB timer

The flow in which the Management event method automatically generates the EJB timer is as follows:

Figure 2-24: Flow of automatic generation of the EJB timer (Management event method)



1. Re-start the J2EE server.
2. A message is displayed.
   After a J2EE server has started, the message KDJE30028-I is displayed. The Management event occurs when this message is output.
3. The Management action is executed.
   The Management event is received and the Management action is executed. The batch file is executed as the Management action.
4. Batch file is executed.
   The EJB application is executed in the batch file.
5. The business method is invoked in the EJB client application.
   In the EJB client application, the Enterprise Bean is generated and the business method is invoked.
6. The EJB timer is generated.
   In the business method of the Enterprise Bean, the EJB timer is generated from the Timer Service object.

(b) Setup for automatically generating the EJB timer

Specify the following settings to automatically generate an EJB timer by using a Management event.

1. Create the business method that generates the EJB timer in the Enterprise Bean.
2. Generate the EJB client application.
   Specify to invoke the business method that is created in step 1.
3. Create the environment and the batch file for executing the EJB client application created in step 2.
4. Specify to automatically execute the Management event.
   Enable the Management event. For the Management event, specify the message KDJE30028-I that is displayed when a J2EE server starts, and specify the batch file created in step 3 as the management action for this management event.
   For details on the Management events, see *9. Notification of Management Events and Automatic Execution of Processing with Management Actions* in the *uCosminexus Application Server Operation, Monitoring, and Linkage Guide*.

2.12.4 Deleting the EJB timer

To delete the EJB timer, cancel the EJB timer.

Note that in the case of a single-event timer, the EJB timer is deleted when the callback of the timeout method finishes. In the case of an interval timer, the EJB timer is not deleted until the EJB timer is cancelled.

During the cancellation of the EJB timer, a single EJB timer is deleted by the EJB timer cancellation method (`cancel` method of the `javax.ejb.Timer` object). If the EJB timer is deleted, the subsequent callbacks are not executed. The timing for deleting the EJB timer by EJB timer cancellation depends upon whether the cancellation of the EJB timer is executed under transaction management.

When the EJB timer is cancelled under transaction management

The EJB timer is deleted when the transaction is committed. As a result, the time for a timeout may be reached during the period when the method for cancellation of the EJB timer is invoked until the transaction is committed.

When the transaction rolls back, cancellation of the EJB timer is cancelled.

When the EJB timer is not cancelled under transaction management

The EJB timer is deleted immediately after the method for cancellation of the EJB timer is invoked.

Reference note

When stopping the J2EE applications, all EJB timers of the J2EE applications to be stopped are deleted. When stopping a J2EE server or during its abnormal termination, all EJB timers on a J2EE server are deleted.

2.12.5 Functionality for operating the Timer Service

The functionality used for operating the Timer Service is explained below. The operation functionality consist of the following two types:

- Functionality for controlling the number of callback threads of the timeout method
- Functionality for retrying callback of the timeout method

Customize the properties of a J2EE server to specify the functionality. For details on the properties to be set up, see *2.12.9 Settings in the execution environment*.

(1) Functionality for controlling the number of callback threads of the timeout method

Specify the number of threads to be executed concurrently in the entire J2EE server for processing callback of the timeout method. This is called the maximum number of callback threads.

When multiple EJB timers timeout concurrently, depending upon the maximum number of callback threads of the timeout method, the operation will be as follows:

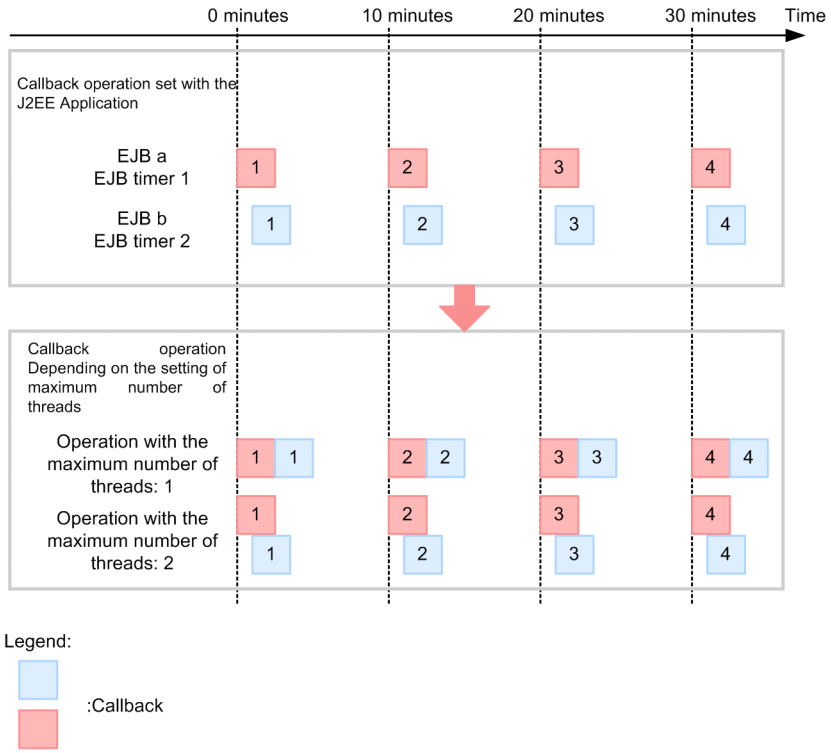
- When the maximum number of callback threads is 1 (default)

  Callback is executed sequentially. As a result, callback may start even later than the time set in the EJB timer.
- When the maximum number of callback threads is 2 or more

  Only the specified number of callback processes are executed concurrently. If you increase the maximum thread count, the resource consumption will increase by an equal amount. Therefore, specify an appropriate value for the number of threads to be executed concurrently.

The following figure shows the relationship between the settings for the maximum number of callback threads and callback processing:

Figure 2-25: Relationship between the settings for the maximum number of callback threads and callback processing



In this figure, the settings are specified so that EJB timer 2 will be timed out one minute after the timeout of the EJB timer 1 occurs. When the maximum number of callback threads is set to 1, the callback processing of EJB timer 2 will start after the callback processing of the EJB timer 1 finishes. When the maximum number of callback threads is specified as 2, two callback processes can be executed concurrently, therefore, the callback processing of EJB timer 2 is started one minute after the EJB timer 1, as per the settings.

Important note

Even if there is a surplus in the number of callback threads, but a shortage of instances of Enterprise Beans, the callback processing is executed after awaiting release of the instances. Therefore, specify the Enterprise Bean instance pool after considering the number of instances to be called back.

(2) Retrying callback of the timeout method

In the case of a failure in calling back the timeout method, retry callback.

Callback may fail in the following cases:

- When an unsearched exception (`java.lang.RuntimeException`, `java.lang.Error` and their subclasses) is thrown during callback due to a timeout
- When the timeout method is either the `Required` attribute or the `RequiresNew` attribute of CMT and the transaction rolls back

Set the following settings for retrying callback:

Number of retry

Specify the number of retries. When you specify 0, retry is not performed. If the count of executing retries reaches the specified retry number, retry is not performed at a timeout.

Interval of executing retry

Specify the time period from failure of a callback until the retry callback.

2.12.6 Operations of the EJB timer and callback

This subsection explains the operations of the EJB timer and callback in the following cases:

- When a past time is specified in the EJB timer
- When multiple EJB timers have been specified in one Enterprise Bean class
- When the previous callback has not finished during timeout
- When cancel EJB timer is invoked from multiple threads
- When cancel EJB timer is invoked during execution of callback
- When an unconcluded transaction with the EJB timer cancelled, is present during callback

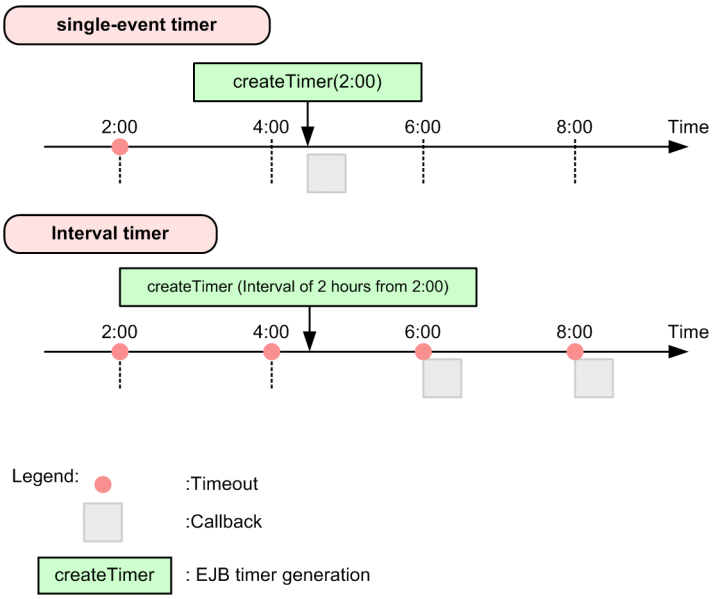
(1) When a past time is specified in the EJB timer

The operations differ according to the types of timers.

(a) In the case of a single-event timer or interval timer

When a past time is specified in the EJB timer, call back the timeout method once immediately after the generation of the EJB timer, in the case of a single-event timer. In the case of an interval timer, call back the timeout method during the timeout that occurs after the generation of the EJB timer. The following figure shows the operations when a past time is specified in the EJB timer:

Figure 2-26: Operation when a past time is specified in the EJB timer



In the case of a single-event timer

When a single-event timer that is set to timeout at 2:00, passes the timeout timing and the timeout occurs at 4:15, a callback to the timeout method is executed once, immediately after the timeout occurs.

In the case of an interval timer

When an interval timer that is set to timeout at an interval of 2 hours starting from 2:00, passes the timeout timing and the timeout occurs at 4:15, the callback to the timeout method is executed at the subsequent timeout timings (the first timeout will be executed at 6:00).

(b) In the case of a calendar-based timer

The operations when you specify a past date and time in a calendar format or specify an invalid value as the date and time (such as 2/31) are as follows:

- When you specify a past date and time in the timer generated with an API

The application starts, but the timer is not generated and the message KDJE43206-W is output.

- When you specify a past date and time in the timer generated with an annotation
  The application starts, but the timer is not generated and the message KDJE43220-W is output.
- When you specify an invalid value (such as 2/31) as the date and time in the timer
  The application starts, but the timer is not generated.
- When you invoke the `getNextTimeout` method with a Timer object immediately before the time limit elapses

The message KDJE43211-W is output when you invoke the `getNextTimeout` method.

(2) When multiple EJB timers have been specified in one Enterprise Bean class

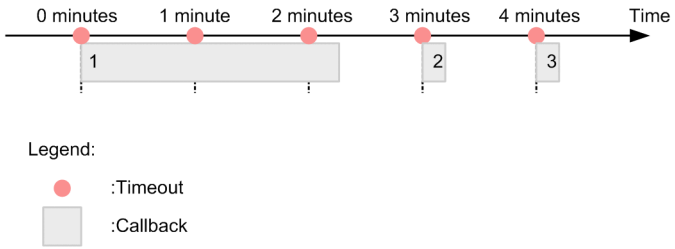
When there are multiple EJB timers for one Enterprise Bean class, the callback processing is executed concurrently if callback of these multiple EJB timers overlap. The callback threads and the Enterprise Bean instances for which processing can be performed concurrently are, however, restricted in certain cases. If such callback threads and instances are not present, wait until the release of the threads and the instances.

(3) When the previous callback has not finished during timeout

Only one callback can be executed at a time in a single EJB timer. Depending upon the processing contents, the processing of the timeout method may sometimes take longer. As a result, in the case of an interval timer, the time of the next timeout elapses during execution of the callback processing and the time of multiple timeouts may elapse until one callback finishes. In such a case, the callback processing that could not be performed as per the timeout time is not executed, but callback processing is executed for the timeout that occurs after the time of completion of the previous callback.

The following figure shows the operations when the previous callback has not finished during current callback:

Figure 2-27: Operations when the previous callback has not finished during current callback



In this figure, the setting is such that timeout occurs at an interval of one minute and the callback processing is executed. When the time for the second and the third timeout elapses during processing of the first callback, the second callback processing will be performed at the next timeout after completion of the first callback processing (that is the scheduled time for the fourth timeout). The callback processing is not performed twice for the timeout that has elapsed the timeout time.

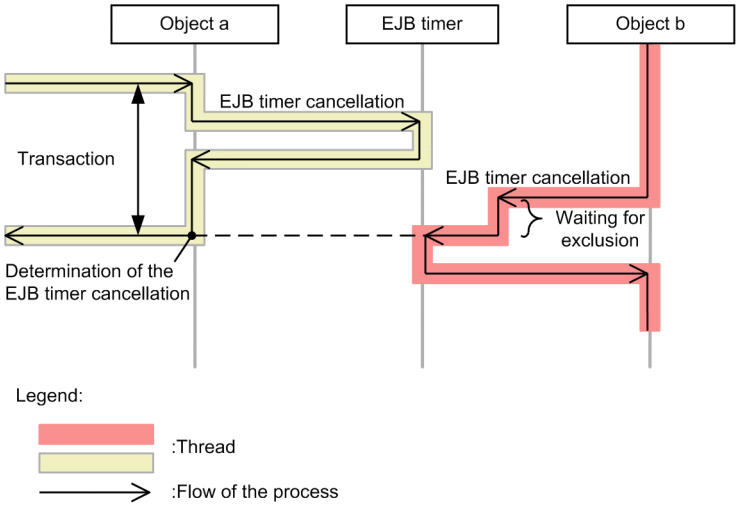
(4) When EJB timer cancellation is invoked from multiple threads

When the EJB timer cancellation method of the same EJB timer is invoked concurrently from multiple threads, cancellation is executed sequentially.

If, however, EJB timer cancellation is executed under the transaction management, it is not possible to verify whether the EJB timer is deleted until the transaction is concluded. As a result, if some other thread cancels the same EJB timer, it results in exclusive waiting.

The following figure shows the operations when cancellation of the same EJB timer is invoked concurrently from multiple threads:

Figure 2-28: Canceling the EJB timer invoked from multiple threads

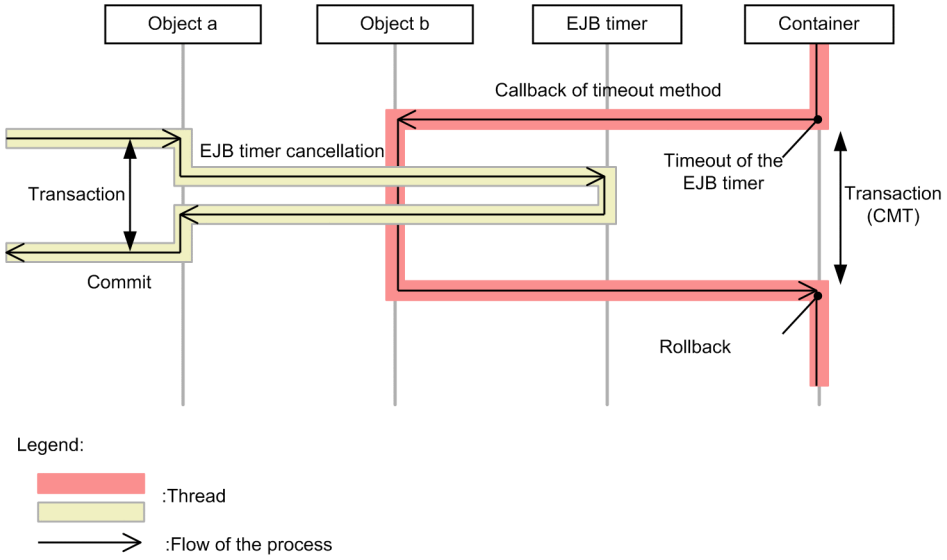


(5) When cancel EJB timer is invoked during execution of callback

When EJB timer cancellation method is invoked during the execution of callback of the timeout method, the operation is as follows:

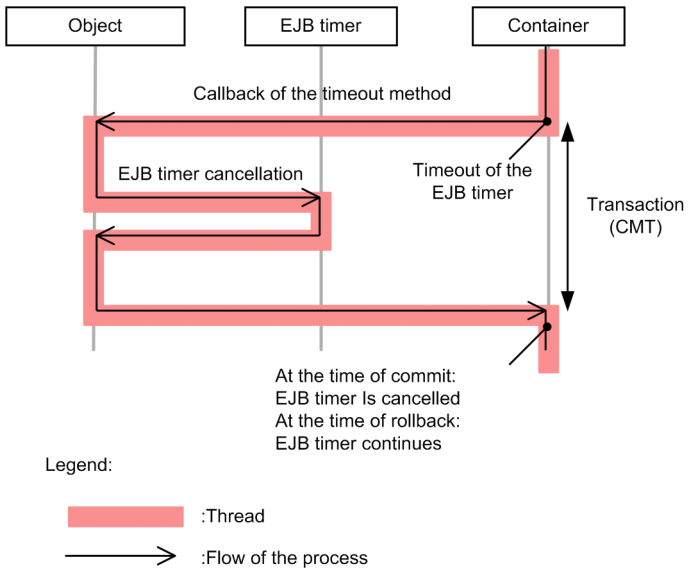
- The invocation of EJB timer cancellation method terminates normally.
- The callback processing of the timeout method is inherited and is executed, but the following operation is performed after completion of callback:
  - When the timeout method uses a CMT transaction, the transaction rolls back.
  - When the timeout method does not use a CMT transaction, it does not affect the result of a transaction executed during callback.
- Even in the following cases, in which callback is normally retried, retry will not be performed:
  - When the CMT transaction rolls back
  - When an unsearched exception is thrown
- A message is output in the message log when callback finishes.
  - When the timeout method uses a CMT transaction, the message KDJE43161-W is output.
  - When the timeout method does not use a CMT transaction, the message KDJE43160-W is output.
- The operation is different in the case, wherein EJB timer cancellation is executed from another transaction during callback of the timeout method using a CMT transaction, and in the case, wherein EJB timer cancellation is executed in the timeout method using a CMT transaction.
  - The following figure shows the operation when EJB timer cancellation is executed from another transaction during callback of the timeout method using a CMT transaction:

Figure 2-29: Cancellation during execution of callback



- The following figure shows the operation when EJB timer cancellation is executed in the timeout method using a CMT transaction:

Figure 2-30: Cancellation of EJB timer in the timeout method

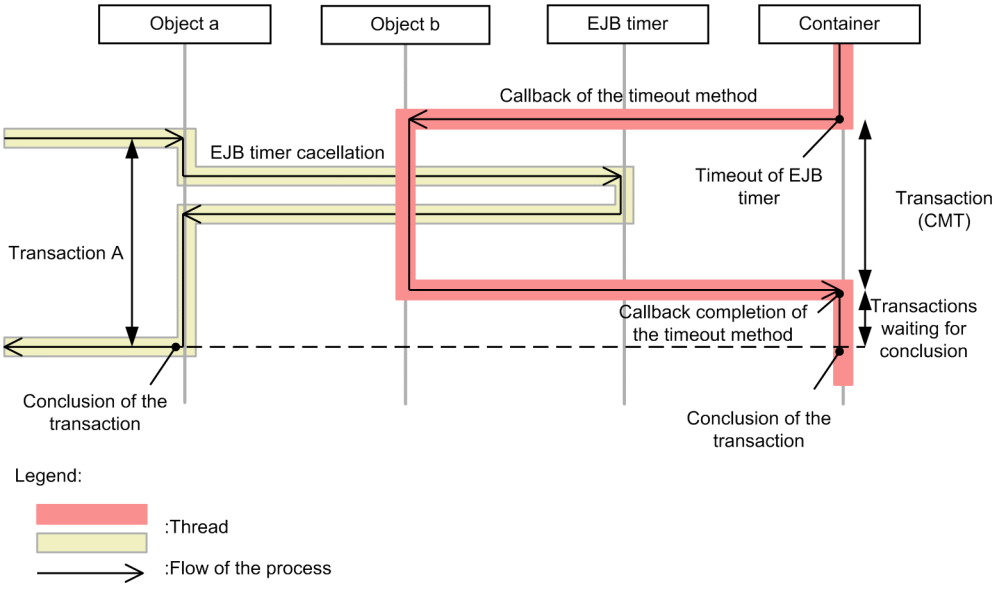


In this figure, the EJB timer actually is cancelled after completion of the timeout method. As a result, the EJB timer is cancelled once the transaction is committed. The EJB timer continues to exist when the transaction rolls back.

(6) When an unconcluded transaction with the EJB timer cancelled, is present during callback

For an unconcluded transaction A in which the EJB timer is cancelled, the callback processing of the timeout method waits exclusively from the time of completion of callback until the transaction A is concluded. Therefore, when the timeout method is under the management of a CMT transaction, the conclusion of the transaction will be in the waiting state until the transaction A is concluded. The following figure shows the flow of processing in such a case:

Figure 2-31: Callback operations when an unconcluded transaction with the EJB timer cancelled



2.12.7 Implementing an application using the Timer Service

This subsection describes the details of implementation with an API and implementation with an annotation as implementation contents of an application using the Timer Service. This subsection also describes the method of specifying a schedule in the calendar format.

(1) Implementation with API

Implement the following contents:

- **Specifying the timeout method**
  Specify the timeout method to be called back with either of the following methods:
  - Specify the timeout annotation in the method used as the timeout method.
  - Implement the `TimedObject` interface in the Enterprise Bean. In such a case, the `ejbTimeout` method defined in the `TimedObject` interface will become the timeout method.
- **Acquiring the javax.ejb.TimerService objects**
  Acquire the `TimerService` object using DI, the `getTimerService` method of the EJBContext interface, or the `lookup` method of JNDI.
- **Generating the EJB timer**
  Invoke any of the following methods of the `TimerService` object and implement the code for generating the timer.
  - createTimer()
  - `createSingleActionTimer()`
  - `createIntervalTimer()`
  - `createCalenderTimer()`
- **Canceling the EJB timer (Timer)**
  When you must execute the cancellation process, implement the code for canceling the timer. Acquire the timer from `javax.ejb.TimerService` and `javax.ejb.TimerHandle`.

The examples of implementing this processing and the precautions during the implementation are as follows:

(a) Example of implementation when DI is used (specify the timeout method with the Timeout annotation)

The following is an example of implementation when an annotation. In this example, the timeout method (`myTimeout`) is specified with the Timeout annotation (@Timeout).

```
@Stateless public class TimerSessionBean{
  @Resource TimerService timerService;

  public void createMyTimer(long intervalDuration){
    Timer timer = timerService.createTimer
                          (intervalDuration, "MyTimer");
  }

  @Timeout public void myTimeout(Timer timer) {
    System.out.println("TimerSessionBean: myTimeout ");
  }

  public void cancelTimers(){
    Collection<Timer> timers = timerService.getTimers();
    for(Timer timer: timers) {
      timer.cancel();
    }
  }
}
```

(b) Example of implementation when EJBContext is used (implement the TimedObject interface)

The following is an example of acquiring the `TimerService` object by using SessionContext that is a subclass of EJBContext. In this example, the `TimedObject` interface is implemented to execute the processing.

```
public class TimerSessionBean implements SessionBean, TimedObject{
  private SessionContext context;

  public void createMyTimer(long intervalDuration) {
    System.out.println("TimerSessionBean: start createTimer ");
    TimerService timerService = context.getTimerService();
    Timer timer = timerService.createTimer
                          (intervalDuration, "MyTimer");
  }

  public void ejbTimeout(Timer timer) {
    System.out.println("TimerSessionBean: ejbTimeout ");
  }

  public void setSessionContext(SessionContext sc) {
    context = sc;
  }
}
```

(c) Example of implementation when lookup is used (Implement the TimedObject interface)

The following is an example of acquiring the `TimerService` object using the JNDI. In this example, the `TimedObject` interface is implemented to execute the processing.

```
public class TimerSessionBean implements SessionBean, TimedObject{
  private SessionContext context;

  public void createMyTimer(long intervalDuration) {
    System.out.println("TimerSessionBean: start createTimer ");
    InitialContext context = new InitialContext();
    TimerService timerService =
      (TimerService)context.lookup("java:comp/TimerService");
    Timer timer = timerService.createTimer
                          (intervalDuration, "MyTimer");
  }

  public void ejbTimeout(Timer timer){
    System.out.println("TimerSessionBean: ejbTimeout ");
  }
}
```

(2) Implementation with an annotation

You can automatically generate a timer using the @Schedule annotation.

Examples are as follows:

(a) Example of @Schedule annotation specification

An example of the @Schedule annotation is as follows. In this example, one timer is generated.

```
// In this example, generateMonthlyAccountStatements method is set
// to be executed at 1:00 on first day of every month with @Schedule
// annotation.
@Schedule(hour="1", dayOfMonth="1",info="AccountStatementTimer")
public void generateMonthlyAccountStatements() { ... }
```

Specify any character string in the info attribute of the @Schedule annotation. You can acquire the specified character string with the `getInfo` method of the related Timer object.

(b) Example of @Schedules annotation specification

An example of the @Schedules annotation specification is as follows. In this example, multiple timers are generated.

```
// In this example, sendLunchNotification method is set to be executed
// at 12:00 from Monday to Thursday and at 11:00 on Friday with
// @Schedules annotation.
@Schedules (
     {  @Schedule(hour="12", dayOfWeek="Mon-Thu"),
        @Schedule(hour="11", dayOfWeek="Fri")
      }
)
public void sendLunchNotification() { ... }
```

You can invoke multiple timers from one callback method with the @Schedules annotation.

(3) Method for specifying a schedule in calendar format

Use values in the coding of a calendar format as follows:

The values that you can specify are based on the standard specifications.

- **Specifying a value**
  Specify a value in seconds or months, as shown below:
  Example
  second = "10"
  month = "Sep"
- **Wild card**
  Specify the wild card as shown below:
  Example
  second = "*"
  dayOfWeek = "*"
- **Specifying multiple values**
  You can specify multiple values by separating the values with ",".
  Example
  second = "10,20,30"
  dayOfWeek = "Mon,Wed,Fri"
  minute = "0-10,30,40"
- **Specifying the range**
  You can specify a value for a fixed period using "-".
  Example
  second="1-10"
  dayOfWeek = "Sat-Mon"
  The following example shows the range from the 27th day of a month to the 3rd day of the next month. First "-" indicates a time interval between the start time and the end time.

Example
dayOfMonth = "27-3"
This is an example of specifying days. However, you can also specify seconds, minutes, or hours in a similar way.

- **Incremental specification**
  Specify the incremental value with "/". The process is executed every time the value specified as the incremental value increases.
  In the following example, the process is executed every 5 minutes.
  Example:
  minute = "*/5"
  This example is same as in the case of specifying "0,5,10,15,20,25,30,35,40,45,50,55".
  In the following example, the process is executed every 10 seconds after 30 seconds of each minute.
  Example:
  second = "30/10"
  This example is same as in the case of specifying "second = "30,40,50, ...".

! Important note

If an argument of the `createCalendarTimer` method is invalid, the `IllegalArgumentException` exception is thrown and the message KDJE43209-E is output.

2.12.8 Precautions when using the Timer Service

This subsection describes the precautions when using the Timer Service.

(1) Specifying the info argument of the createTimer method

The `getInfo` method of the `Timer` object assumes the object specified in the `info` argument of the `createTimer` method of the `TimerService` object or the `info` argument of the constructor method of the `TimerConfig` object as the return value. Therefore, the return value of the `getInfo` method might be different from the object during the execution of a method such as the `createTimer` method, and therefore an error might occur.

To prevent this, Hitachi recommends that you set the object specified in the `info` argument of the `createTimer` method be invariable, such as a type String or Integer, or do not change the status of the object specified in the `info` argument. If changed, the return value of the `getInfo` method will be the object specified after the change.

(2) Specifying the timeout method in the DD and attribute files

When specifying the timeout method in the <method> tag of a DD and the property file, perform either of the followings:

- Do not add the definition of the <method-intf> tag immediately below the <method> tag.
- Keep the element of the <method-intf> tag as blank.

(3) Operation when attempting to acquire a Timer Service object from a Bean that does not support Timer Service objects

The following table describes the methods for acquiring a Timer Service object when the Timer Service is not supported. When attempting to acquire a `TimerService` object from a Bean or servlet that does not support Timer Service, perform the following operation according to the methods:

Table 2-35: Operation when attempting to acquire a Timer Service object from a Bean that does not support Timer Service objects

Method for acquiring the TimerService objects	Operation
`EJBContext#getTimerService`	The `IllegalStateException` exception is thrown.

Method for acquiring the TimerService objects	Operation
JNDI Lookup	The `NamingException` exception is thrown.
DI	An attempt to deploy has failed.

Irrespective of whether or not the timeout method is implemented in a Bean that supports the Timer Service, you can acquire the TimerService objects.

(4) Operation specifications of the APIs provided by the TimerService

Of the operations performed during the invocation of an API provided by the TimerService, some operation specifications are not specified clearly in the EJB specifications. The operation specifications of `javax.ejb.TimerService` and `javax.ejb.Timer` in Application Server are as follows:

- **javax.ejb.TimerService**

  The following table describes the operation when the timeout method is implemented in the Bean that invokes the methods of `javax.ejb.TimerService`, and when the timeout method is not implemented:

  Table 2-36: Operation when the javax.ejb.TimerService API is used from the Bean

Implementation of the timeout method	Types of javax.ejb.TimerService methods	
	`createTimer`, `createCalendarTimer`, `createIntervalTimer`, `createSingleActionTimer`	`getTimers`
When the timeout method is implemented#	The EJB timer generation processing is executed.	The EJB timer correction is returned.
When the timeout method is not implemented	The `IllegalStateException` exception is thrown.	A blank correction is returned.

  #

  When the timeout method is implemented, the operation is performed according to the EJB specifications.

- **javax.ejb.Timer API**

  The following table describes the operation during the invocation of the methods of `javax.ejb.Timer` for the case when the timeout method is executed and when the timeout method is not executed:

  Table 2-37: Operation when the javax.ejb.Timer API is used

Execution of the timeout method	Types of javax.ejb.Timer methods				
	`cancel`	`getHandle`, `getInfo`, `isCalendarTimer`, `getSchedule`	`getNextTimeout`	`getTimeRemaining`	`isPersistent`
When the timeout method is not executed#	See *2.12.6 Operations of the EJB timer and callback*.	Operation is performed according to the specifications.	The timing of occurrence of the next timeout is returned.	The time period until the occurrence of the next timeout is returned.	Always returns false because persistence of the EJB timer is not supported.
When the timeout method is executed			The time registered as the expected start time of a timeout being executed is returned.	`0` is returned.	

#
When the timeout method is not executed, the operation is performed according to the EJB specifications.

Reference note

A sample program of the Timer Service is provided in Application Server. For an overview and details on how to execute the sample programs, see *Appendix M Sample Programs Provided by Application Server* in the *uCosminexus Application Server System Setup and Operation Guide*.

(5) Precautions during execution of the timeout method

- Execute the timeout method in any of the following formats. If you do not execute the timeout method in either of these formats, the application fails to start.
  - `void` <method name>`()`
  - `void` <method name>(Timer timer)
- An application exception must not be thrown in the timeout method. If an application exception is thrown, the application fails to start.
- Do not declare `final` or `static` in the timeout method.

2.12.9 Settings in the execution environment

When using the TimerService, you must set up a J2EE server.

Set up a J2EE server with the Easy Setup definition file. Specify the 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 TimerService in the Easy Setup definition file:

Table 2-38: Definition of the TimerService in the Easy Setup definition file

Items	Parameter to be specified	Setting contents
Maximum retry count	`ejbserver.ejb.timerservice.retryCount`	Specify the maximum count for retrying callback of the timeout method of the Timer Service.
Retry interval	`ejbserver.ejb.timerservice.retryInterval`	Specify a value in seconds for the retry interval of calling back the timeout method of the Timer Service.
Maximum number of threads to call back the timeout method	`ejbserver.ejb.timerservice.maxCallbackThreads`	Specify the maximum number of threads to call back the timeout method of the Timer Service in the entire J2EE server.

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.*

2.12.10 Precautions when using the Timer Service

The precautions when using the Timer Service are explained below:

- When the Timer Service is used, there may be a difference in the time specified in the EJB timer, and the time when the timeout method actually is called back. In such a case, the reasons could be as follows:
  - Execution of a garbage collection

    If a garbage collection is executed in JavaVM at the time specified in the EJB timer, the processing of garbage collection is given priority. The timeout method is called back after the garbage collection finishes, therefore, there may be some variations from the specified time.
  - Platform or hardware environment

Use the time period of JavaVM in the Timer Service. The time period of JavaVM depends upon the platform or hardware environment. The callback is also executed as per the time period of JavaVM, therefore, there may be some variations from the specified time.

- On a machine where a J2EE server is running, if you modify the system time using a software, such as NTP client software, the following operations are performed for the timeout time of the already registered EJB timer:
  - In the case of a single-event timer

    Timeout occurs assuming that the system time before modification is inherited.
  - In the case of an interval timer

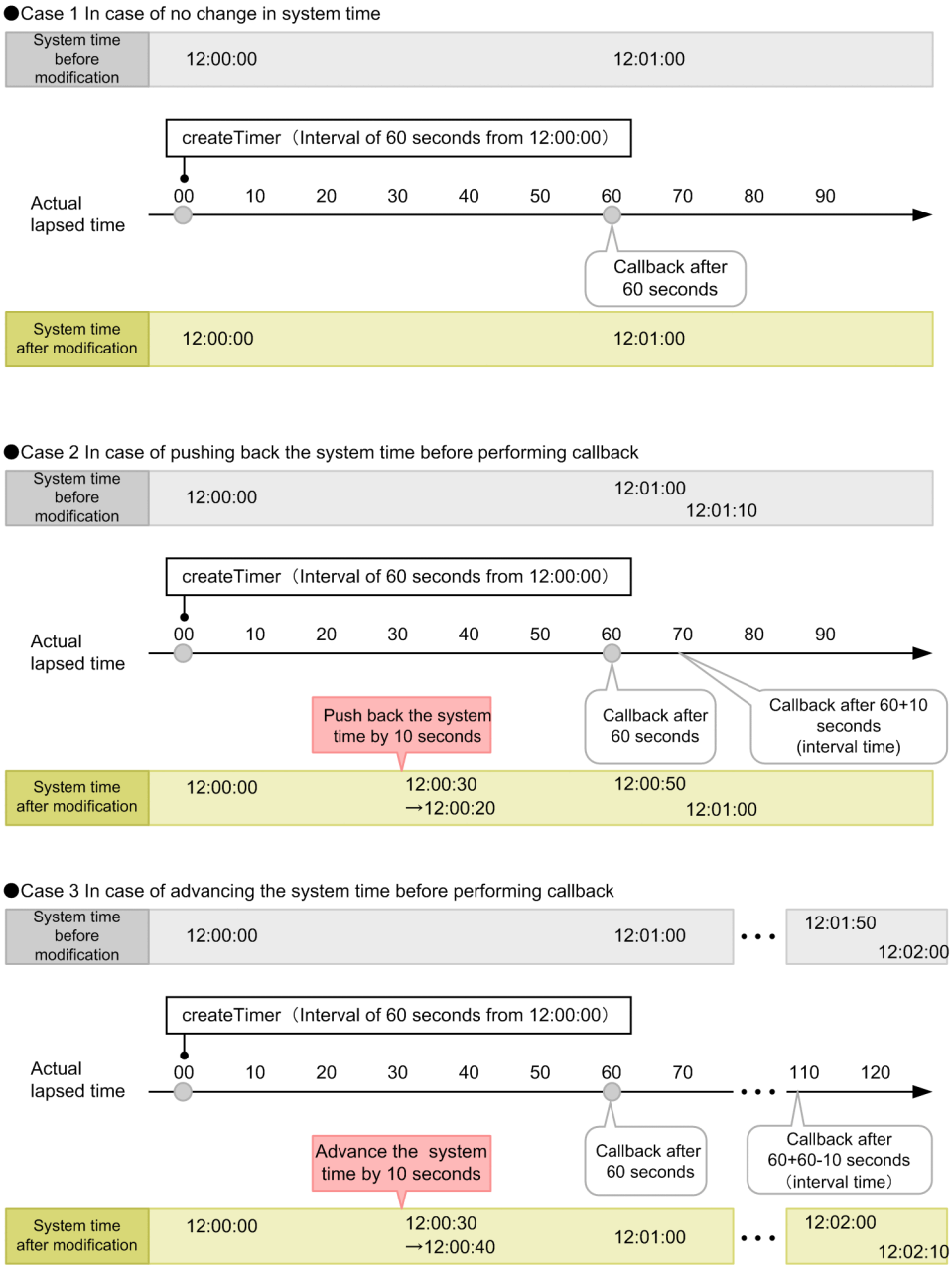
    The first timeout occurs assuming that the pre-modification system time is inherited after modification. From the second time onwards, a timeout occurs as per the system time after modification.
  - In the case of a calendar-based timer or when using the method of specifying the @Schedule annotation

    If you set only one callback at a specified date and time, a timeout occurs assuming that the system time before the modification is inherited.

    If you set a periodic callback, the first timeout occurs assuming that the pre-modification system time is inherited after modification. From the second time onwards, timeouts occur as per the system time after modification.

  The following figure shows examples of the callback timing when the system time is not changed, when the system time is returned, and when the system time is advanced.

Figure 2-32: Example of call back timing



2.13 Invoking the remote interface of EJB

This section describes how to invoke the remote interface of an EJB.

The following table describes the organization of this section:

Table 2-39: Organization of this section (Invoking the remote interface of an EJB)

Category	Title	Reference location
Description	Optimizing local invocation in the EJB remote interface	*2.13.1*
	Referencing and passing the values of the EJB remote interface	*2.13.2*
	Operation during the occurrence of a communication failure in the EJB remote interface	*2.13.3*
Setup	Defining in cosminexus.xml	*2.13.4*
	Settings in the execution environment	*2.13.5*
Notes	Precautions concerning invocation of the EJB remote interface	*2.13.6*

Note:

There is no specific description of *Implementation* and *Operation* for this functionality.

2.13.1 Optimizing local invocation in the EJB remote interface

This subsection explains the optimization of local invocation in the remote interface of EJB.

The methods defined in the remote interface of EJB are invoked by RMI-IIOP, however, local invocation optimization can be applied to this invocation.

Note that the methods defined in the local interface of EJB are not invoked by using RMI-IIOP, but a normal Java invocation method is used, therefore, this functionality is not applicable.

When optimizing local invocation in the remote interface, you can select the range for optimizing the local invocation. Customize the properties of a J2EE server to specify the range for optimization. For details on customizing the operation settings of a J2EE server, see *2.13.5 Settings in the execution environment*.

The following table describes the correspondence between the range and the operations of the functionality for optimizing local invocation and the values specified in the keys of a J2EE server properties (`usrconf.properties`):

Table 2-40: Range and operations of functionality for optimizing local invocation

Items	Value of `ejbserver.rmi.localinvocation.scope` key#		
	All	app	None
Range of local invocation optimization	Within the same J2EE server	Within the same application	There is no range
Thread configuration	Caller and Callee are always in the same thread	Caller and Callee are in the same thread only in the same application	Caller and Callee are always in different threads
Class loader configuration	EJB is loaded by the container class loader (J2EE server unit)	EJB is loaded by the application class loader (application unit)	
Local transaction	Can be used in a J2EE server	Can be used in the same application	Can be used in the same J2EE component

#

This is a key specified in `usrconf.properties`.

2.13.2 Referencing and passing the values of the EJB remote interface

Normally, if an EJB method containing a remote interface is invoked, the arguments and the return value are copied and passed by value, however, the arguments and the return value can also be passed by reference. When the values are passed by reference, the aim is to reduce the load as compared to copying the values and passing them.

When passing the values by reference, you need to pay attention to the changes in the arguments and the return value and to the deployment of the client and application wherein values are passed by reference, since the arguments and the return value are referenced directly.

When the objects with the `java.io.Serializable` interface implemented are defined in the arguments or the return value of a method, you can expect a reduction in the load by pass by reference of EJB remote interface values. You can expect better results when the number and size of the objects is larger.

The two methods of setup are explained below. If setup is performed using either of the following methods, the pass by reference of values is enabled:

- **Setting for each EJB**

  Define enabling and disabling of the functionality for each EJB, as an attribute of the Session Bean or the Entity Bean.
- **Setting for each J2EE server**

  Collectively define the enabling and disabling of the functionality for each J2EE server as properties of a J2EE server.

2.13.3 Operation during the occurrence of a communication failure in the EJB remote interface

You can choose either of the following operations of the client, when communication failure occurs during invocation of the EJB method defined as remote interface from the EJB client:

- Re-establish the connection and resend the request
- Neither re-establish the connection nor resend the request

The settings for using this functionality are specified as properties of a J2EE server or the EJB client application.

Note that in the case of an EJB client application, you can also specify in API (`setProperty` method of `java.lang.System` class). For defining in the `java.lang.System.setProperty` method, specify the settings after starting the process of the EJB client application and before invoking the methods of the Enterprise Bean for the first time.

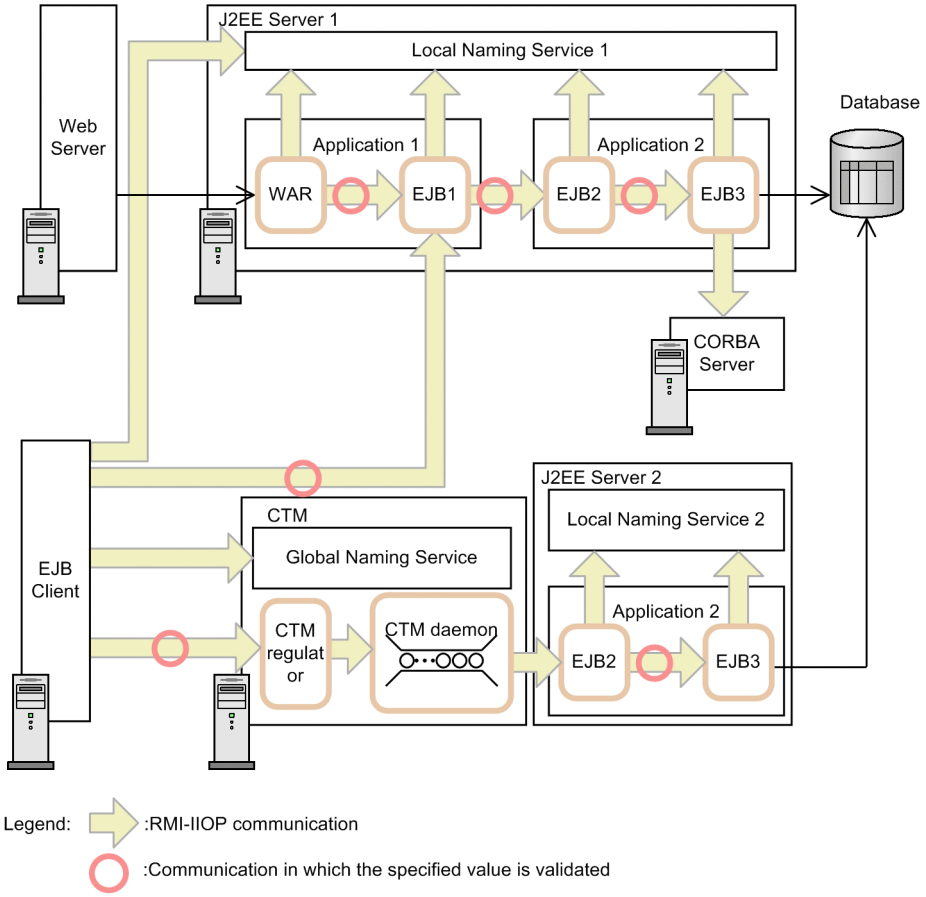
(1) Communication for which the settings are enabled

The operations during communication failure are enabled when communication failure occurs during the invocation of the EJB method defined as the remote interface. Invocation of the EJB method indicates the following:

- Invocation of EJB from a Web application
- Invocation of EJB from the EJB client
- Invocation of EJB from another EJB

The following figure shows the communication for which the settings are enabled:

Figure 2-33: Communication for which the settings are enabled



Note that the settings are disabled in the following cases:

- Invocation of the EJB method defined as a local interface
- Invocation of the EJB method defined as a remote interface within the range in which optimization of local invocation is enabled
- Invocation of the Naming Service

(2) Recommended settings

Depending upon the system type, Hitachi recommends the following settings:

In the case of a search and reference node system

Hitachi recommends that you specify the settings for re-establishing the connection and resending the requests. Consequently, the results can be acquired without facing failure in requests.

In the case of an update node system

Hitachi recommends that you specify the settings for not executing either re-establishment of the connection or resending of requests. In an update node system, if you specify the settings for re-establishing the connection and resending the requests, there is a risk of sending duplicate requests.

2.13.4 Defining in cosminexus.xml

Of the invocation functionality of an EJB remote interface, the settings that specify the Enterprise Bean in which the pass by reference functionality of the EJB remote interface is to be enabled are defined in `cosminexus.xml`.

The definition is specified in the `<ejb-jar>` tag of `cosminexus.xml`. The tag to be specified is different for each type of the target Enterprise Bean.

The definition of a timeout of an EJB container as specified in `cosminexus.xml` is as follows:

Tag to be specified

In the case of a Session Bean

`<session>-<pass-by-reference>` tag

In the case of an Entity Bean

`<entity>-<pass-by-reference>` tag

Contents of the file

Specify whether to enable the pass by reference functionality for the values of EJB remote interface in each Enterprise Bean.

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

2.13.5 Settings in the execution environment

Of the invocation functionality of an EJB remote interface, the settings for the following functionality must be specified in a J2EE server:

- Range of the local invocation optimization functionality of the EJB remote interface
- Settings for specifying whether to enable the pass by reference functionality of the EJB remote interface
- Operation of the EJB client during the occurrence of a communication failure in the EJB remote interface#

#

When the EJB client is in the form of an EJB client application, specify the settings as properties of the EJB client application.

Furthermore, the settings that specify the Enterprise Bean in which the pass by reference functionality of the EJB remote interface is enabled can be defined in the J2EE application. Reference the settings when setting up or changing the properties of a J2EE application that does not include `cosminexus.xml`.

(1) Setting J2EE servers

Set up a J2EE server with the Easy Setup definition file. Specify the settings in the `<configuration>` tag of the logical J2EE server (`j2ee-server`) of the Easy Setup definition file.

The following table describes the definition of the invocation functionality of an EJB remote interface in the Easy Setup definition file:

Table 2-41: Definition of the invocation functionality of an EJB remote interface in the Easy Setup definition file

Items	Parameter to be specified	Setting contents
Range of the local invocation optimization functionality	`ejbserver.rmi.localinvocation.scope`	Specify the range of optimizing a local invocation in the EJB remote interface.
Pass by reference functionality of the remote interface	`ejbserver.rmi.passbyreference`#	Specify whether to enable the pass by reference functionality of the remote interface.
Operation of the EJB client during the occurrence of a communication failure in the remote interface	`ejbserver.container.rebindpolicy`	If a J2EE server at the specification location is the client of another J2EE server, specify the operation for retrying a connection at the EJB client side, and the operation for resending requests.

#

In a J2EE application, you can specify whether to enable the pass by reference functionality in each Enterprise Bean. The pass by reference functionality will be enabled if `Enable` is specified in either a J2EE server or the Enterprise Bean.

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) Setting EJB client applications

When the EJB client is in the form of an EJB client application, the operation of the EJB client during the occurrence of a communication failure in the EJB remote interface will be specified in the property of the EJB client application.

Key to be specified

`ejbserver.container.rebindpolicy` key

Setting contents

Specify the operation for retrying a connection at the EJB client side, and the operation for resending requests.

(3) Setting J2EE applications

You can define the settings for specifying whether to enable the pass by reference functionality of the EJB remote interface in each Enterprise Bean, in the execution environment. Specify the settings in the J2EE application imported into a J2EE server, and execute the setup only when setting up or changing the properties of a J2EE application that does not include `cosminexus.xml`.

A J2EE application is set up in the execution environment with server management commands and property files. Use the following property files to define the reference mapping:

Table 2-42: Property files used to define the management method of an Enterprise Bean transaction

Setting target	Attribute files
Session Bean	Session Bean attribute file
Entity Bean	Entity Bean attribute file

The tags specified in the property files correspond to the DD or `cosminexus.xml`. For details on the `cosminexus.xml` settings, see *2.13.4 Defining in cosminexus.xml*.

2.13.6 Precautions concerning invocation of the EJB remote interface

This subsection describes the precautions concerning invocation of the EJB remote interface.

(1) Precautions when applying optimization of local invocation

- When applying optimization of local invocation of the EJB remote interface in the processing within the same application (when specifying `ejbserver.rmi.localinvocation.scope=app` in `usrconf.properties`), specify the same host in the following provider URL:
  - Host of the provider URL used in a J2EE server
  - Provider URL host used from the J2EE application

  If different hosts are specified, the local invocation will not be optimized.

  Furthermore, even when the same host is specified but the string specified in the host name is different, the local invocation will not be optimized. For example, if there is a difference in upper case and lower case characters or a different IP address and host name is specified, the local invocation will not be optimized.
- The provider URLs used in a J2EE server are determined in the following priority order:

  In case of `ejbserver.naming.startupMode=inprocess`:

  1. Value of the `vbroker.se.iiop_tp.host` property
  2. Value of `InetAddress.getLocalHost().getHostName()`

  In case of `ejbserver.naming.startupMode=automatic`:

  1. Value of `InetAddress.getLocalHost().getHostName()`

In case of `ejbserver.naming.startupMode=manual`:

1. Value of the `ejbserver.naming.host` property
2. Value of `InetAddress.getLocalHost().getHostName()`

- The provider URLs used from a J2EE application are determined in the following priority order:
  1. Part of the argument passed to lookup when the naming service switching functionality is used
  2. `java.naming.provider.url` property specified when `InitialContext` is generated
  3. Provider URL that a J2EE server uses

(2) Precautions when setting up the operation during the occurrence of a communication failure in the EJB remote interface

When `NO_RECONNECT` (no reconnection or resending) is selected with the `ejbserver.container.rebindpolicy` key of system properties and if a connection is disconnected due to the communication failure, you will not be able to reuse the corresponding object reference because a reconnection is prevented. Therefore, when a method of the Enterprise Bean is to be invoked for the next time in the EJB client, re-execute the `lookup` method in an EJB home object and the `create` method in an EJB object, and then invoke the method. If the connection is disconnected during the method invocation of an Enterprise Bean, the method will throw either of the following exceptions:

`java.rmi.RemoteException`

Instance when the detail field is `org.omg.CORBA.REBIND`

`java.rmi.MarshalException`

Instance when the detail field is `org.omg.CORBA.COMM_FAILURE`

An example of coding of the client that catches these exceptions is as follows:

```
try {
     //JNDI.lookup()
     //EJBHome.create()
     //EJBObject.invoke()
    } catch (java.rmi.MarshalException e) {
      if (e.detail instanceof org.omg.CORBA.COMM_FAILURE) {
        //Processing corresponding to communication failure
      }
    } catch (java.rmi.RemoteException e) {
      if (e.detail instanceof org.omg.CORBA.REBIND) {
        //Processing corresponding to communication failure
      }
    }
```

(3) Precautions when executing multiple threads

When you use multiple threads (from one client) to invoke an EJB method defined as a remote interface, there is only one connection to the server and multiple threads share that connection. If a timeout occurs and the connection closes with one EJB invocation, a communication failure occurs in the invocation of other EJBs. If you have selected "reconnect the connection and resend the request" as a client operation, EJB invocation will be executed again.

To prevent the closure of the connection when a timeout occurs, specify the following contents in `usrconf.properties` (user property file for the J2EE server, user property file for a batch server, or user property file for a Java application) at the EJB invocation source (client side). For details, see the *TPBroker Operation Guide*.

Contents:

```
vbroker.ce.iiop.ccm.htc.readerPerConnection=true
vbroker.ce.iiop.ccm.htc.threadStarter=true
```

When you specify the above-mentioned contents, a thread to be activated is added. For details on the estimation of the number of threads, see *5.2.1 Estimating the resources used by J2EE server* in the *uCosminexus Application Server System Design Guide.*

(4) Precautions when the caching functionality has been enabled in the naming management functionality

When the caching functionality has been enabled in the naming management functionality, the disabled object reference on the cache will be acquired if the `lookup` method is executed after the connection is disconnected. In such a case, when you execute the `javax.rmi.PortableRemoteObject.narrow` method and the `create` method using the acquired object reference, a CORBA exception (such as `org.omg.CORBA.OBJECT_NOT_EXIST`) will occur.

Clear the disabled cache after the connection is disconnected. For details on the procedure, see *2.8.2 Clearing the cache used in naming* in the *uCosminexus Application Server Common Container Functionality Guide*.

2.14 Fixing the communication port and IP address of the EJB container (TPBroker options)

This section describes how to fix the communication port and IP address (TPBroker options) of an EJB container.

By defining the options of Cosminexus TPBroker via J2EE server, you can perform the operations by fixing the communication ports and the IP address of the EJB container. Hitachi strongly recommends that you specify the settings for fixing the ports when the security of the system is to be improved by reducing the used ports to the minimum extent. For details on the Cosminexus TPBroker option, see *TPBroker Users Guide*.

The following table describes the organization of this section:

Table 2-43: Organization of this section (Fixing the communication port and IP address of an EJB container) (TPBroker option)

Category	Title	Reference location
Description	Fixing the communication port	*2.14.1*
	Fixing the IP address	*2.14.2*
Setup	Settings in the execution environment	*2.14.3*

Note:

There is no specific description of *Implementation*, *Operation*, and *Notes* for this functionality.

2.14.1 Fixing the communication port

By default, a random value is allocated by Cosminexus TPBroker to the communication ports of the EJB container.

On the contrary, you can fix the communication ports in each J2EE server by specifying any value as the Cosminexus TPBroker option. Specify the port numbers so that there is no duplication of the port numbers with other programs.

2.14.2 Fixing the IP address

By default, the IP address of the EJB container is acquired by Cosminexus TPBroker from the execution environment machine system, and is then allocated.

On the contrary, you can fix the IP address in each J2EE server by specifying any value as the Cosminexus TPBroker option.

2.14.3 Settings in the execution environment

When fixing the communication port and IP address of an EJB container, you must set up a J2EE server.

Set up a J2EE server with the Easy Setup definition file. Specify the settings in the `<configuration>` tag of the logical J2EE server (`j2ee-server`) of the Easy Setup definition file.

The following table describes the definition in the Easy Setup definition file for fixing the communication port and IP address of the EJB container:

Table 2-44: Definition for fixing the communication port and IP address of the EJB container in the Easy Setup definition file

Items	Parameter to be specified	Setting contents
Communication port of the EJB container	`vbroker.se.iiop_tp.scm.iiop_tp.listener.port`	Specify the communication port of the EJB container.
Whether or not to fix the IP address or the host name	`vbroker.se.iiop_tp.host`	Specify whether or not to fix the IP address or host name used by the EJB container.

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.*

2.15 Using the interceptor

This section describes how to use the interceptor.

The interceptor can be specified in Application Server with a DD, property file, or annotation (excluding the default interceptor).

The following table describes the organization of this section:

Table 2-45: Organization of this section (Using the interceptor)

Category	Title	Reference location
Description	Overview of the usage of the interceptor	*2.15.1*
	Preventing the invocation of upper level interceptor	*2.15.3*
	Execution order of an interceptor	*2.15.4*
Implementation	Defining in an annotation or a DD	*2.15.2*
Setup	Configuring in the execution environment	*2.15.5*
Notes	Notes on inceptors	*2.15.6*

Note:
There is no specific description of *Operation* for this functionality.

2.15.1 Overview of the usage of the interceptor

The following interceptors can be used in Application Server:

- **Default interceptor**
  This interceptor is applicable to all components included in EJB-JAR. The default interceptor becomes the upper level interceptor of the class level interceptor and the method level interceptor. Set this interceptor by using a DD or a property file.
- **Class level interceptor**
  This interceptor is applicable to the specified classes. The class level interceptor becomes the upper level interceptor of the method level interceptor. Set this interceptor by using an annotation, a DD, or a property file.
- **Method level interceptor**
  This interceptor is applicable to the specified business methods. Set this interceptor by using an annotation, a DD, or a property file.

2.15.2 Defining in an annotation or a DD

You set up an interceptor as an EJB-JAR property either by specifying in an annotation or by using a DD or a property file.

(1) Defining the default interceptor

The default interceptor is applicable to all the components included in EJB-JAR.

In a DD, you can specify the information of the default interceptor by coding the `<interceptor-binding>` tag below the `<ejb-jar>` tag.

The following table describes the components to be specified below the `<interceptor-binding>` tag:

Table 2-46: Components (default interceptor) to be specified below the <interceptor-binding> tag when using a DD

Tag names	Optional/ Required	Contents of the file
<description>	Optional	Specify any information.
<ejb-name>	Required	Specify a wild card (*).
<interceptor-class>	Required	Specify the class name of the interceptor class in the element.

The value of the components specified in any other tag is not applied.

The following is an example of coding of the DD when the default interceptor is used:

```
<ejb-jar>
    ...
  <assembly-descriptor>
   <interceptor-binding>
     <description xml:lang="en">Default Interceptor</description>
     <ejb-name>*</ejb-name>
     <interceptor-class>test.ejb30.MyDefaultIC</interceptor-class>
     <interceptor-class>test.ejb30.MyDefaultIC2</interceptor-class>
   </interceptor-binding>
  </assembly-descriptor>
    ...
</ejb-jar>
```

In this example of coding, `test.ejb30.MyDefaultIC` and `test.ejb30.MyDefaultIC2` are specified as the default interceptor classes.

- Rules for specifying the interceptor class

The specification method of the class name in the *interceptor-class* tag conforms to the EJB 3.0 specification. Specify as per the following rules:

- You can specify one interceptor class name in one *interceptor-class* tag.
- You can code multiple *interceptor-class* tags. If you code multiple *interceptor-class* tags, the interceptor is invoked in the order in which you have coded.
- You can specify the interceptor method in an interceptor class, specified in the *interceptor-class* tag, with the following annotations:
- `@AroundInvoke`
- `@PostConstruct`
- `@PreDestroy`

For details on annotations, see *2. Dependency Injection and Annotation corresponding to the annotation servers* in the *uCosminexus Application Server API Reference Guide*.

(2) Defining the class level interceptor

The class level interceptor is applicable to the specified classes. This point describes how to set a class level interceptor by using a DD and by using the EJB-JAR property file in the execution environment.

In a DD, you can specify the information of the class level interceptor by describing the <interceptor-binding> tag below the <ejb-jar> tag.

The following table describes the elements below the <interceptor-binding> tag.

Table 2-47: Elements (class level interceptor) to be specified below the <interceptor-binding> tag when using a DD tag

Tag name	Mandatory/ Optional	Contents to be specified
<ejb-name>	Mandatory	Specify EJB name.
<interceptor-class>	Mandatory	Specify the class name of the interceptor class in the element.

Note that elements are not specified in the <method> tag.

(3) Defining the method level interceptor

The method level interceptor is applicable to the specified business methods. This point describes how to set a method level interceptor by using a DD and by using the EJB-JAR property file in the execution environment.

In a DD, you can specify the information of the method level interceptor by describing the <interceptor-binding> tag below the <ejb-jar> tag.

The following table describes the elements below the <interceptor-binding> tag.

Table 2-48: Elements (method level interceptor) to be specified below the <interceptor-binding> tag when using a DD tag

Tag name	Mandatory/ Optional	Contents to be specified
<ejb-name>	Mandatory	Specify the EJB name.
<interceptor-class>	Mandatory	Specify the class name of the interceptor class in the element.
<method-name>	Mandatory	Specify the method name in the element.
<method-params>	Optional	Specify the argument list for the method in the element.

The following table describes the range of the business methods that define the method level interceptor when you specify a business method in the <method-name> tag and you omit the value specified for the <method-params> tag or you specify the argument list in the <method-params> tag. Note that when * (wild card) is specified in the <method-name> tag, the definition of the method level interceptor is not used.

Table 2-49: Range of business methods that define the method level interface

Value specified in the <method-params> tag	Range of business methods that define the method level interceptor
The value is omitted.	An interceptor is defined for all business methods that have exactly matching method names.
An argument list is specified.	An interceptor is defined for the business methods that have exactly matching method names and argument list. If you specify an argument list, you can overwrite the information specified in an annotation in the DD.

■ Applicable rules for method level interceptor

The definition of the method level interceptor that is used when executing a business method is determined in the following order:

1. If a definition completely matches with the business method to be executed, the method name specified in the <method-name> tag, and the argument list specified in the <method-params> tag, then the definition of that method level interceptor is used.
2. If the business method to be executed and the method specified in the <method-name> tag match and the argument list is not specified in the <method-params> tag of a definition, then the definition of that method level interceptor is used.

3. If a method level interceptor in which the business method to be executed and the method specified in the <method-name> tag are matching, is not defined, then the method level interceptor definition is not used.

Note that when the business method to be executed corresponds to both definitions, namely the definition in which the method specified in the <method-name> tag and the argument list specified in the <method-params> tag match completely, and the definition in which the method specified in the <method-name> tag is matching and the argument list is not specified in the <method-params> tag, then the definition that completely matches with the method name and argument list is used.

Also, the method level interceptor specified using annotations is handled as a definition in which the method names and the argument list match completely. If you want to overwrite the method level interceptor specified using annotations in DD, you must describe the interceptor in DD by using a definition in which the method names and the argument list match completely.

2.15.3 Controlling the invocation of upper level interceptor

You can control the execution of an upper level interceptor in a class level interceptor and a method level interceptor by using an annotation or a DD.

The following table describes the range in which the execution of the interceptor can be controlled.

Table 2-50: Range in which the execution of the upper level interceptor can be controlled

Type of interceptor	Interceptor to be controlled		
	Default interceptor	Class level interceptor	Method level interceptor
Default interceptor	N	N	--
Class level interceptor	Y#	N	--
Method level interceptor	Y	Y	--

Legend:

Y: The invocation of the upper level interceptor can be controlled.

N: Does not correspond to the upper level, and will be ignored even if you create a definition to control invocation.

--: Invocation of the upper level interceptor cannot be defined.

#

When a definition for controlling the invocation of the default interceptor in the class level interceptor is specified, the invocation of the default interceptor is controlled irrespective of the definition in the method level interceptor.

2.15.4 Execution order of the interceptors

The execution order of an interceptor is determined according to the following rules by default:

- Execution order of EJB 3.0 specifications

  The following execution order is provided in the EJB 3.0 specifications:

  1. Default interceptor
  2. Class level interceptor
  3. Method level interceptor
  4. Interceptor method specified in the Bean class

- Rules to be followed when there are multiple interceptors of the same level

  If there are multiple interceptors of the same level, the interceptors are executed in the sequence defined in the annotations or DD.

- If the interceptor class has a parent class and the interceptor method is defined in the parent class

  Execution will take place from the parent class.

- Rules according to the definition for controlling the invocation of an upper level interceptor

When a definition to control the invocation of an upper level interceptor is specified, the interceptor of the specified level is not invoked.

- The execution order in the <interceptor-order> tag

  If you execute the following business methods, an interceptor of a level higher than the interceptor defining the <interceptor-order> tag of DD or the property file is not invoked:

  - The business method of the Enterprise Bean class to which the class level interceptor defining the execution order in the <interceptor-order> tag is applied
  - The business method to which the method level interceptor defining the execution order in the <interceptor-order> tag is applied

  Among the interceptors that use the <interceptor-order> tag, the definition of the lowest level interceptor and the interceptor of a level lower than that interceptor is executed.

Note that the execution order of the default interceptor described here can be changed using the following method:

- You can exclude the execution of the default interceptor class by using the @ExcludeDefaultInterceptors annotation or the <exclude-default-interceptors> tag of DD.
- You can exclude the execution of the class level interceptor class by using the @ExcludeClassInterceptors annotation or the <exclude-class-interceptors> tag of DD.
- You can replace the order of interceptor classes of all levels by describing the <interceptor-order> tag of DD.

Points (1) to (4) describe these rules and the execution order combining the rules for overwriting annotations based on DD. For details on overwriting an annotation based on DD, see *12.6.2 Overwriting annotations based on DD* in the *uCosminexus Application Server Common Container Functionality Guide*.

(1) When the definition for controlling the invocation of the default interceptor and class level interceptor is specified

The following table describes the execution order of the interceptors when the definition for controlling the invocation of the default interceptor and class level interceptor is specified. In this table, the execution order for combinations shown by Y/N in the column *Use of <interceptor-order>* for each item number is shown by numbers in front of the DD or annotation in the column *Execution order of the interceptor*.

Table 2-51: Execution order of the interceptors (when the definition for controlling the invocation of the default interceptor and class level interceptor is specified)

No.	Use of <interceptor-order>			Execution order of the interceptor			
	Default	Class level	Method level	Default	Class level	Method level	Interceptor method
1	Y	Y	Y	--	--	1. DD	2. Annotation
2	Y	Y	N	--	--	1. Annotation 2. DD	3. Annotation
3	Y	N	Y	--	--	1. DD	2. Annotation
4	Y	N	N	--	--	1. Annotation 2. DD	3. Annotation
5	N	Y	Y	--	--	1. DD	2. Annotation
6	N	Y	N	--	--	1. Annotation 2. DD	3. Annotation
7	N	N	Y	--	--	1. DD	2. Annotation
8	N	N	N	--	--	1. Annotation 2. DD	3. Annotation

Legend:

Default: Default interceptor

Class level: Class level interceptor

Method level: Method level interceptor

Interceptor method: Interceptor method specified in the Bean class

Y: Execution order is specified by using the <interceptor-order> tag.

N: Execution order is not specified by using the <interceptor-order> tag.

--: Not executed.

DD: The interceptor specified in the DD is executed.

Annotation: The interceptor specified in the annotation is executed.

(2) When the definition for controlling the invocation of the default interceptor is specified

The following table describes the execution order of the interceptors when the definition for controlling the invocation of the default interceptor is specified. In this table, the execution order for combinations shown by Y/N in the column *Use of <interceptor-order>* for each item number is shown by numbers in front of the DD or annotation in the column *Execution order of the interceptor*.

Table 2-52: Execution order of the interceptors (when the definition for controlling the invocation of the default interceptor is specified)

No.	Use of <interceptor-order>			Execution order of the interceptor			
	Default	Class level	Method level	Default	Class level	Method level	Interceptor method
1	Y	Y	Y	--	--	1. DD	2. Annotation
2	Y	Y	N	--	1. DD	2. Annotation 3. DD	4. Annotation
3	Y	N	Y	--	--	1. DD	2. Annotation
4	Y	N	N	--	1.Annotation 2. DD	3. Annotation 4. DD	5. Annotation
5	N	Y	Y	--	--	1. DD	2. Annotation
6	N	Y	N	--	1. DD	2. Annotation 3. DD	4. Annotation
7	N	N	Y	--	--	1.DD	2. Annotation
8	N	N	N	--	1. Annotation 2. DD	3. Annotation 4. DD	5. Annotation

Legend:

Default: Default interceptor

Class level: Class level interceptor

Method level: Method level interceptor

Interceptor method: Interceptor method specified in the Bean class

Y: Execution order is specified by using the <interceptor-order> tag.

N: Execution order is not specified by using the <interceptor-order> tag.

--: Not executed.

DD: The interceptor specified in the DD is executed.

Annotation: The interceptor specified in the annotation is executed.

(3) When the definition for controlling the invocation of the class level interceptor is specified

The following table describes the execution order of the interceptors when the definition for controlling the invocation of the class level interceptor is specified. In this table, the execution order for combinations shown by Y/N in the

column *Use of <interceptor-order>* for each item number is shown by numbers in front of the DD or annotation in the column *Execution order of the interceptor*.

Table 2-53: Execution order of the interceptors (when the definition for controlling the invocation of the class level interceptor is specified)

No.	Use of <interceptor-order>			Execution order of the interceptor			
	Default	Class level	Method level	Default	Class level	Method level	Interceptor method
1	Y	Y	Y	--	--	1. DD	2. Annotation
2	Y	Y	N	1. DD	--	2. Annotation 3. DD	4. Annotation
3	Y	N	Y	--	--	1. DD	2. Annotation
4	Y	N	N	1. DD	--	2. Annotation 3. DD	4. Annotation
5	N	Y	Y	--	--	1. DD	2. Annotation
6	N	Y	N	1. DD	--	2. Annotation 3. DD	4. Annotation
7	N	N	Y	--	--	1. DD	2. Annotation
8	N	N	N	1. DD	--	2. Annotation 3. DD	4. Annotation

Legend:

Default: Default interceptor

Class level: Class level interceptor

Method level: Method level interceptor

Interceptor method: Interceptor method specified in the Bean class

Y: Execution order is specified by using the `<interceptor-order>` tag.

N: Execution order is not specified by using the `<interceptor-order>` tag.

--: Not executed.

DD: The interceptor specified in the DD is executed.

Annotation: The interceptor specified in the annotation is executed.

(4) When the definition for controlling the invocation of the upper level interceptor is not specified

The following table describes the execution order of the interceptors when the definition for controlling the invocation of the upper level interceptor is not specified. In this table, the execution order for combinations shown by Y/N in the column *Use of <interceptor-order>* for each item number is shown by numbers in front of the DD or annotation in the column *Execution order of the interceptor*.

Table 2-54: Execution order of the interceptors (when the definition for controlling the invocation of the upper level interceptor is not specified)

No.	Use of <interceptor-order>			Execution order of the interceptor			
	Default	Class level	Method level	Default	Class level	Method level	Interceptor method
1	Y	Y	Y	--	--	1. DD	2. Annotation
2	Y	Y	N	--	1. DD	2. Annotation 3. DD	4. Annotation
3	Y	N	Y	--	--	1. DD	2. Annotation

No.	Use of <interceptor-order>			Execution order of the interceptor			
	Default	Class level	Method level	Default	Class level	Method level	Interceptor method
4	Y	N	N	1. DD	2. Annotation 3. DD	4.Annotation 5. DD	6. Annotation
5	N	Y	Y	--	--	1.DD	2. Annotation
6	N	Y	N	--	1. DD	2. Annotation 3. DD	4. Annotation
7	N	N	Y	--	--	1.DD	2. Annotation
8	N	N	N	1. DD	2. Annotation 3. DD	4. Annotation 5. DD	6. Annotation

Legend:

Default: Default interceptor

Class level: Class level interceptor

Method level: Method level interceptor

Interceptor method: Interceptor method specified in the Bean class

Y: Execution order is specified by using the `<interceptor-order>` tag.

N: Execution order is not specified by using the `<interceptor-order>` tag.

--: Not executed.

DD: The interceptor specified in the DD is executed.

Annotation: The interceptor specified in the annotation is executed.

2.15.5 Configuring the execution environment

Interceptors can also be set in the execution environment. When setting an inceptor in the execution environment, set the interceptor to the J2EE application that is imported to a J2EE server.

J2EE application can be set in the execution environment with the server management command and property file. EJB- JAR property file is used to define interceptors. Note that you cannot perform settings by using the server management command (`cjsetresprop -type ejb` command) for EJB-JAR files that are not included in the J2EE application.

The tag specified in the EJB-JAR property file is compatible with DD. For details on the definitions in DD (`ejb-jar.xml`), see *2.15.2 Defining in an annotation or a DD*.

2.15.6 Notes on inceptors

- When there are multiple `<interceptor-binding>` tags that have specified "*" (wild card) in the `<ejb-name>` tag, the contents described in the uppermost `<interceptor-binding>` tag become valid. The subsequent contents are not set.
- In a property file, if there are multiple `<interceptor-binding>` tags in which the value of the `<ejb-name>` tag, the `<named-method>` tag, and all the elements under the `<named-method>` tag match, the contents described in the upper most `<interceptor-binding>` tag become valid. The subsequent contents are not set.
- In a DD, if there are multiple `<interceptor-binding>` tags in which the value of the `<ejb-name>` tag, the `<method>` tag, and all the elements under the `<method>` tag match, the contents described in the upper most `<interceptor-binding>` tag become valid. The subsequent contents are not set.

2.16 Omitting local business interfaces (Using No-Interface view)

With EJB 3.1, when executing a local invocation in a Session Bean, you can create an EJB without creating a local business interface. In such cases, you can invoke all the business methods, which are published as the No-Interface view from the client. Omission of the local business interface leads to easier development and maintenance activities of the EJB.

The following table describes the organization of this section.

Table 2-55: Organization of this section (omitting a local business interface (using No-Interface view))

Category	Title	Reference location
Description	Overview of No-Interface view	*2.16.1*
Implementation	Definition for using No-Interface view	*2.16.2*
	Methods that cannot be used	*2.16.3*
Notes	Precautions during development	*2.16.4*

Note: There is no specific description of *Setup and Operation* for this functionality.

2.16.1 Overview of No-Interface view

A session Bean is accessed from the following three clients:

- Remote client

  A remote client invokes a business method by using the remote business interface (or the remote home interface). This client invokes a method provided in the remote client view of a Session Bean.
- Local client

  A local client invokes a business method by using the local interface. This client invokes a method provided in the local client view of a Session Bean.
- Web service client

  A Web service client invokes a business method by using the Web service. This client invokes a method provided in the WebService client view of a Session Bean.

The No-Interface view is one of the local client views. It is available in EJB 3.1 or later.

If you use the No-Interface view, definition of a local business interface is not required. If none of the remote client view, local client view or Web service client view is implemented in an application, the No-Interface view is created by the EJB container. Even if any of the views is implemented in an application, you can use the No-Interface view by explicitly specifying the `@LocalBean` annotation. You need not implement any special interface for using the No-Interface view in a Session Bean.

By using the No-Interface view, you can use all the public methods in a Session Bean from the client. You can acquire the reference from a client to the No-Interface view with the DI or JNDI look-up.

No-Interface view is created when an application starts and it is available until the application stops.

! Important note

When you invoke a method published in the No-Interface view by the client, you cannot invoke a method that specifies an access modifier other than a public access modifier. If you invoke a method that specifies an access modifier other than a public access modifier, javax.ejb.EJBException occurs.

2.16.2 Definition for using No-Interface view

An EJB container creates the No-Interface view when it corresponds to one of the following views:

- When the following views are not implemented in a Session Bean:
  - Local client view
  - Remote client view
  - Web service client view

  The `java.io.Serializable` interface, `java.io.Externalizable` interface, and interface in the `javax.ejb` package are excluded from the judgment of whether the above mentioned views are implemented.
- When the `@LocalBean` annotation is defined in a Session Bean

2.16.3 Methods that cannot be used

When using the No-Interface view, do not use the following methods as business methods. If you use these methods, a compilation error occurs and the application fails to start.

- `public void init(Hashtable)`
- `public void init(Object)`
- `public void init(String, Hashtable)`
- `public void initializeBIInstance()`
- `public Throwable convertRemoteException(Throwable)`
- `public Throwable unwrapServerException(Throwable)`

Also, do not use the following methods defined in the java.lang.Object class:

- `equals(Object)`
- `hashCode()`
- `toString()`
- `clone()`
- `finalize()`

2.16.4 Precautions during development

The precautions to be taken when developing a Session Bean that uses the No-Interface view are as follows:

- Do not include `java.rmi.RemoteException` in the method of a Session Bean where the local business interface is omitted.

  However, if you invoke a method in which an interface in the `javax.ejb` package is implemented, an error does not occur even if `java.rmi.RemoteException` is included in the `throws` clause.
- A method of the Session Bean in which an interface in the `javax.ejb` package is implemented does not serve as a business method. When using the No-Interface view, do not invoke such methods from the client.
- You cannot declare `final` in any method in the Bean class and its parent class. You cannot invoke a method with a final declaration in the No-Interface view.
- If you declare `static` or `final` in the methods of a Session Bean, such methods are not treated as business methods. When using the No-Interface view, do not invoke such methods from the client.
- If you use the `@AroundInvoke` annotation in a normal business interface, execution of the business methods results in a deployment error. However, when using the No-Interface view, if you use the `@AroundInvoke` annotation in a public method, an error does not occur. Such methods are treated as business methods.
- With the standard specifications, you can use the `@Remove` annotation in business methods. However, if you specify the `@Remove` annotation in a business method in which the `@PreDestroy` annotation is also specified, `javax.ejb.NoSuchObjectLocalException` is thrown, on invoking the method from the client.

2.17 Asynchronous invocation of Session Bean

In EJB3.1, you can invoke the business method of a Session Bean asynchronously. Consequently, you can execute multiple processes in parallel.

The following table describes the organization of this section.

Table 2-56: Organization of this section (Asynchronous invocation of Session Bean)

Category	Title	Reference location
Description	Applicability of asynchronous invocation of Session Bean	*2.17.1*
	Handling transactions in asynchronous invocation	*2.17.2*
	Handling the application information in an asynchronous invocation	*2.17.3*
Implementation	Defining the annotation used for asynchronous invocation	*2.17.4*
	Specifying return values for an asynchronous method	*2.17.5*
	Operation for execution status and execution result of an asynchronous method based on Future<V> object	*2.17.6*
	Definitions in cosminexus.xml	*2.17.7*
Notes	Notes on annotation when implementing an asynchronous method	*2.17.8*
	Notes on operation of an asynchronous method	*2.17.9*

Note: There is no specific description of *Setup and Operation* for this functionality.

2.17.1 Applicability of asynchronous invocation of Session Bean

If you execute an asynchronous invocation for the Session Bean, the EJB container immediately returns the control to the client which is a source of invocation without waiting for the processing of the Session Bean. A method that receives the asynchronous invocation executes processing in a thread different from the source of invocation. On the client machine, throughput of the application improves because application moves to the next processing without waiting for the result.

You can execute an asynchronous invocation for a Session Bean containing the following EJB 3.1 compliant interfaces or views:

- Remote business interface
- Local business interface
- No-Interface view

You can perform an asynchronous invocation by using the `@Asynchronous` annotation.

You can execute all the public methods in a Stateless Session Bean or Singleton Session Bean as asynchronous methods by using the `@Asynchronous` annotation.

Note that you cannot implement asynchronous methods with the method of a Stateful Session Bean because the session details and transaction status at the time of executing the asynchronous invocation are not retained.

The transmission of security information when an asynchronous invocation is executed is the same as that in the case of a synchronous invocation.

2.17.2 Handling transactions in asynchronous invocation

You can specify only the following transaction attributes in the asynchronous invocation of a Session Bean:

- `Required` attribute
- `RequiresNew` attribute

- `NotSupported` attribute

However, the transaction context of the client, which is the source of invocation, is not transmitted in the asynchronous method. The application developer needs to handle it as if it does not have the transmitted transaction context. For example, even if you define the `Required` attribute in the asynchronous method, it is processed in the same way as when the `RequiresNew` attribute is specified in an EJB container.

The following table describes the transactions that correspond to the specification of the transaction attribute in an asynchronous method specified on the client machine.

Table 2-57: Transactions that correspond to the specification of transaction attributes in asynchronous methods specified on the client machine

Transaction attributes specified in methods	Transaction on the client machine	Transaction associated with asynchronous method
NOT_SUPPORTED	None	None
	T1	None
REQUIRED	None	T2
	T1	T2
REQUIRES_NEW	None	T2
	T1	T2
SUPPORTS	None	None
	T1	None
NEVER	None	None
	T1	None
MANDATORY	None	Error
	T1	Error

Legend:

None: No transaction is specified or no transaction

T1: Transaction set by the invocation source client

T2: New transaction which is started by an EJB container

2.17.3 Handling root application information in asynchronous invocation

An EJB container performs the output of the root application information at the invocation source to an option, with the event ID (0x84C0) of the PRF trace, which is output just before the callback of an asynchronous method. You can compare the requests at the invocation source and the invocation destination by using this root application information.

2.17.4 Defining the annotation used for asynchronous invocation

When executing an asynchronous invocation, specify the `@Asynchronous` annotation for a business method or a class of the Session Bean. If you specify the `@Asynchronous` annotation in a class, it is considered that the `@Asynchronous` annotation is specified for all the business methods in the class.

The request from the client machine sent by specifying the `@Asynchronous` annotation is executed with a new thread as a Daemon thread.

2.17.5 Specifying return values for an asynchronous method

You can select either of the following values as a return value in an asynchronous method.

- `void`
- `java.util.concurrent.Future<V>` (V is a type of return value)
  Hereafter, it is described as Future<V>.

Note that if you select `void`, you cannot declare an application exception. If you select `Future<V>`, you can declare an application exception. If invoked from a remote interface, the `Future<V>` object used to access the processing result is retained in an EJB container.

In an asynchronous method, you can pass the processing result object of the method to the invocation source by using `javax.ejb.AsyncResult<V>`, which is an implementation class of `Future<V>`.

The examples when `Future<V>` and void are specified as return values in coding that uses the `@Asynchronous` annotation are as follows:

```
// Enterprise Bean Business method

//Example when Future<V> is specified as return value
@Asynchronous
public Future<Integer> performCalculation(...) {
    // ... do calculation
    Integer result = null;
    if (ejb_context.wasCancelCalled()) {
        return new AsyncResult<Integer>(-1);
    }
    ...
    return new AsyncResult<Integer>(result);
}
// Example when void is specified as return value
@Asynchronous
public void performAddition (...) {
    Integer result = null;
    // ... do addition
}
```

2.17.6 Operation for execution status and execution result of an asynchronous method based on Future<V> object

If you specify `Future<V>` as the return value of the asynchronous method, you can execute the following processes using the method of `Future<V>`. You cannot execute these processes when the return value is `void`.

- Cancelling the asynchronous invocation processing
- Acquiring the execution result of the asynchronous invocation processing
- Confirming the execution status of the asynchronous invocation processing
- Acquiring causes of an exception occurrence in the asynchronous invocation processing

(1) Cancelling asynchronous invocation processing

You can cancel the processing by using the cancel method of the `Future<V>` object.

If the cancellation is successful, true is returned as the return value of the method. If the cancellation fails, false is returned as the return value.

If the asynchronous invocation fails, whether to interrupt the processing or execute it as it is, is decided depending on the specification in the `mayInterruptIfRunning` parameter of the cancel method.

(2) Acquiring the execution result of the asynchronous invocation processing

You use the `get` method of the `Future<V>` object to acquire the execution result.

The processing is complete when the processing is successfully executed and the processing result is acquired as the return value or when `ExecutionException` occurs. Hereafter, you can acquire the same result if you acquire the execution result using the same `Future<V>` object.

Note that if a timeout occurs during the method invocation of the `Future<V>` object in Application Server, `EJBException` is thrown for the client by the EJB container.

(3) Confirming the execution status of asynchronous invocation processing

You can confirm whether the execution of the asynchronous processing is complete or cancelled. You can check the status using the following methods of the `Future<V>` object:

- `isDone` method

  When the processing is successfully completed, an exception is thrown, or when the processing is cancelled, true is returned.

- `isCancelled` method

  When the processing is successfully cancelled, true is returned, or when the processing cancellation fails, false is returned.

In the asynchronous method, you can confirm whether the cancel method was invoked from the client machine, using the following methods:

- `wasCancelCalled` method

  When the execution cancellation of the asynchronous invocation processing is invoked by the cancel method, and true (stop the running process) is specified in the `mayInterruptIfRunning` parameter of the `cancel` method, true is returned. If false (complete the running process) is specified in the `mayInterruptIfRunning` parameter of the cancel method, false is returned.

(4) Acquiring causes of the exception occurrence in asynchronous invocation processing

If a system exception occurs in the method of the business interface, `java.rmi.RemoteException` is returned instead of `javax.ejb.EJBException` to the invocation source client machine. If a system exception is received during the asynchronous invocation processing, you can determine that the asynchronous method was not executed on the client machine. In such cases, you can invoke the asynchronous method again.

If an exception other than a system exception, such as an application exception occurs, the client can determine that the exception has occurred during the asynchronous processing after executing the asynchronous processing by the EJB container. In such cases, you can acquire the cause of the exception occurrence by using the exception object.

An example of coding to obtain the cause of the exception that occurs at the time of asynchronous invocation, by invoking the `getCause` method of an exception object, is as follows:

```
// Bean Client
Future future = asynSessionBean.performCalculation();
while (!future.isDone()) {
    Thread.currentThread().sleep(10000);
    future.cancel(true);
    break;
}
if (future.isCancelled() == false) {
    Integer answer = null;
    try{
        answer  = (Integer)future.get();
    }  catch(ExecutionException e){
        System.out.println("caught exception: " +e.getCause());
    }
    System.out.println("Answer=" + answer.toString());
}
asynSessionBean.performAddition();
```

2.17.7 Definitions in cosminexus.xml

You can set the following items in `cosminexus.xml`:

- Number of threads that can simultaneously execute the asynchronous invocation processing and the retention period of the threads
- Timeout of the processing result retention period of the asynchronous method

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

(1) Number of threads that can simultaneously execute the asynchronous invocation processing and retention period of the threads

Define attributes such as the number of threads that can be simultaneously executed and the retention period of threads in `cosminexus.xml`. You can set the following parameters:

- `max-thread-pool-size`
  Specify the maximum number of threads that can be generated in a pool.
- `min-thread-pool-size`
  Specify the maximum number of unused threads that can be stored in a pool.
- `thread-pool-keep-alive`
  If the number of unused threads exceeds the number specified in `min-thread-pool-size`, set the time required for retaining all the threads, until the threads end.

(2) Timeout of the processing result retention period of the asynchronous method

Set a timeout in the retention period of the `Future<V>` object used for acquiring the processing result of the asynchronous method invocation that uses a remote interface. If you do not set the timeout, the processing result is retained until the application stops. This can result in an increase in the memory usage volume and occurrence of the OutOfMemory error in cases such as when there are many processing results. To prevent this, set the maximum value of the period for which the results are to be retained. Set the following parameters:

- result-timeout-value
  Specify the period (Unit: minute) for which the processing result is to be retained. Processing results that exceed the specified period are deleted from the EJB container and are no longer available for referencing.

If you invoke the method in the `Future<V>` object that shows the processing result after deletion, an EJBException that has the character string "KDJE43202-I" in the message is thrown. You can confirm whether "KDJE43202-I" exists in the exception message and determine whether the result is deleted due to a timeout occurrence.

However, in the case of a local client, because the `Future<V>` object is retained locally, the method ends successfully without throwing any exception.

Notes on the Java heap estimation and Java heap tuning when an asynchronous invocation is to be executed, are as follows:

(a) Java heap estimation when executing the asynchronous invocation

If an asynchronous processing is invoked using the remote interface, the `Future<V>` object that shows processing results is retained in JavaVM on which the EJB container operates. Therefore, if there are many requests, you must be careful so that the `OutOfMemory` error does not occur in JavaVM.

When executing the asynchronous invocation, estimate the Java heap size required for each asynchronous method using the following estimation formula.

Java heap size (unit: KB) required for one asynchronous method

$=(1+A) \times (B+C+1) \times D$

Legend:

A: Object size (KB) depending on the user application

B: Average value (minute) of the asynchronous method execution time

C: Value of the result-timeout-value specified in cosminexus.xml

D: Average execution count of asynchronous methods invoked in one minute

Add the values calculated for each asynchronous method and then calculate the required Java heap size as a whole.

Tip

- The java heap size required for invoking the asynchronous method in a remote interface is calculated by the above estimation formula. When invoking with a local interface, the result is not maintained in the EJB container and hence estimation is not required.
- "Object size depending on the user application" is the size of the processing result object. Usually, you need not be aware of the size of the processing result object of the user application. However, if the size is large, change the Java heap size as and when required after measuring the size to be used in the test program.

(b) Notes on tuning the timeout period

Consider the following points regarding the tuning of the timeout period:

- For the timeout period, specify a value longer than the interval in which the `isDone` method of the `Future` object is executed in an application. Also specify a value longer than the period from the invocation of the `get` method of the `Future` object until the invocation of the asynchronous method.
- If you want to reduce the resource usage volume, decrease the value of `result-timeout-value`. However, in such cases, you cannot access the deleted processing results once the time is elapsed.
- Calculate a size that does not consume too much of the Java heap memory after considering the average value of the execution time of the asynchronous method, execution time of the asynchronous method, and the average value of the execution count in one minute. Then set the definition of `result-timeout-value`.

2.17.8 Notes on annotation when implementing an asynchronous method

- If you specify the asynchronous method in a class unit, (if you specify the `@Asynchronous` annotation for a class), you must specify `void` or `Future <V>` as the return value of all the methods in the class. If you specify any other type, the application fails to start.
- If you specify an asynchronous method in a method unit, (if you specify the `@Asynchronous` annotation for method), you must specify `void` or `Future <V>` as the return value of the method for which the `@Asynchronous` annotation is specified. If you specify any other type, the application fails to start.

2.17.9 Notes on operation of an asynchronous method

- You cannot execute the method cancellation functionality in a Session Bean that executes the asynchronous invocation, if you try to cancel the method, the KDJE52703-W message is output; however, the method is not cancelled.

  To cancel the asynchronous method, use a method of the `Future<V>` object.
- When the application stops, cleanup processing is executed. Cleanup processing does not end until all the running processes end. When all the processes are complete, cleanup processing ends and the application stops.
- If you invoke the asynchronous processing of a Session Bean by using a remote interface, an object which causes full garbage collection during the server operation, is generated. Therefore, if you want to prevent the occurrence of a full garbage collection, use the remote interface and do not invoke the asynchronous processing.

2.18 Specifications in Session Synchronization annotation

You can define the processing (Session Synchronization) that is executed before and after the start and end of a transaction by using annotations in addition to implementing the `javax.ejb.SessionSynchronization` interface.

This section describes how to set the processing executed before and after the start and end of a transaction with annotations, when a transaction is managed in CMT.

The following table describes the organization of this section:

Table 2-58: Organization of this section (Setting the processing to be executed before and after start and end of transaction)

Category	Title	Reference location
Description	Method of setting Session Synchronization with annotation	*2.18.1*
Implementation	Rules for implementation	*2.18.2*
Notes	Notes on implementation	*2.18.3*

Note: There is no specific description of *Setup and Operation* for this functionality.

2.18.1 Method of setting Session Synchronization with annotation

When a transaction is managed in CMT, transactions are managed by the container. In such cases, you can develop an application without implementing start and end of the transaction. In the management by a container, the transaction starts immediately before the start of the method and it is committed and ended immediately after the end of the method.

If you want to execute a specific processing before and after the start and end of a transaction by the container, use the Session Synchronization functionality. You can use the Session Synchronization functionality in either of the following methods:

- Method for implementing the `javax.ejb.SessionSynchronization` interface
- Method of specifying the annotation

You can use the method that specifies the annotation in EJB3.1 or later. If you specify annotations, you can set the timing of executing the process before and after the start and end of a transaction without implementing the `javax.ejb.SessionSynchronization` interface. This simplifies the development of the application.

This subsection describes the specifications when annotations are to be used. You can use the following annotations:

`@AfterBegin`

`@AfterBegin` is an annotation which reports the start of a new transaction. A container invokes the processing with the specification of this annotation after the start of the transaction and immediately before invoking the business method.

`@BeforeCompletion`

`@BeforeCompletion` is an annotation which reports the completion of the business method. The processing with the specification of this annotation is invoked after completing the business method and immediately before committing the transaction. If you want to roll back the processing implemented by the business method, you must invoke the `setRollbackOnly` method at this time.

`@AfterCompletion`

`@AfterCompletion` is an annotation which reports the completion of a transaction. The processing with the specification of this annotation is invoked immediately after completion of the transaction.

In this annotation, you can specify `true` or `false` as a parameter. If you specify true, processing is executed when the transaction is committed. If you specify false, processing is executed when the transaction is rolled back.

You can specify these annotations in the class of a Stateful Session Bean (or its parent class). With respective Session Beans, you can specify one of these annotations for each type.

2.18.2 Rules for implementation

Specify the method that specifies annotations by conforming to the following rules. If you do not specify the method properly, an error occurs when the application starts.

- You can use this functionality only in a Stateful Session Bean that manages transactions with CMT. You cannot use this functionality in the Stateless Session Bean and Singleton Session Bean. In BMT, you can manage the timing of the transaction processing with a Session Bean and hence you need not use this functionality.
- Do not declare `final` and `static` in a method that specifies annotation.
- Set the return value of a method that specifies annotations to the void type.
- Do not specify parameters in `@AfterBegin` and `@BeforeCompletion`.
- Specify only one boolean type value in the `@AfterCompletion` parameter.

2.18.3 Notes on implementation

Precautions during the implementation are as follows:

- A container invokes the `afterBegin` method, `beforeCompletion` method, or `afterCompletion` method only when the `javax.ejb.SessionSynchronization` interface is implemented in the target Session Bean (or its parent class) or when the annotation is specified in the Session Bean.

  If both, the implementation of the `javax.ejb.SessionSynchronization` interface and specification of annotations are implemented when developing a Session Bean, implementation of the interface is given priority.
- If you specify the same annotation for multiple times in a Session Bean class, we cannot guarantee which method from among the `afterBegin` method, `beforeCompletion` method or `afterCompletion` method specified for multiple times will be invoked by the container.
- If you specify parameters not conforming to the annotation rules, the application fails to start and the KDJE42039-E message is output.
- You can invoke the methods for which the processing time is set using the `@AfterBegin` annotation, `@BeforeCompletion` annotation, or `@AfterCompletion` annotation directly from the client machine, in the same way as for other business methods. However, generally, the `callback` method is not to be published on the client machine as a business method. Therefore, we recommend declaring a value other than `public` in the access modifier of such methods.

2.19 Using Singleton Session Beans

This section describes the details of the Singleton Session Bean that can be used on Application Server.

The following table describes the organization of this section.

Table 2-59: Organization of this section (Using Singleton Session Bean)

Category	Title	Reference location
Description	Exclusive control of Singleton Session Beans	*2.19.1*
	Error handling in Singleton Session Beans	*2.19.2*
Notes	Precautions when using Singleton Session Beans	*2.19.3*

Note: There is no specific description of *Implementation, Setup, Operation* and *Notes* for this functionality.

2.19.1 Exclusive control of Singleton Session Beans

The two methods for exclusive control of a Singleton Session Bean are as follows:

- **Container-Managed Concurrency**
  This method manages the processing status of the instance of a method level in an EJB container.
- **Bean-Managed Concurrency**
  This method manages the processing status of all the Bean instances in an Enterprise Bean.

When developing a Singleton Session Bean, you must determine the method you use to manage the exclusive control. You cannot use both the methods concurrently.

(1) Container- Managed Concurrency

The EJB container executes exclusive control. Respective business methods and timeout methods are controlled by the Read lock or Write lock. In the case of a method for which the Read lock is set, you can concurrently execute the processing of multiple methods. In the case of a method for which the Write lock is set, any other method is not invoked until the processing of one method is complete.

Specify the type of lock for a method of the Session Bean class or overridden class with an annotation. If you do not explicitly specify the type of lock, operation is performed in the same way as when the Write lock is specified.

When executing exclusive control, you can set the timeout period for pending processes. You can specify the timeout period using the `@AccessTimeout` annotation. If a timeout occurs, `javax.ejb.ConcurrentAccessTimeoutException` is thrown for the invocation source client by an EJB container.

(2) Bean-Managed Concurrency

If you specify the Bean-Managed Concurrency, exclusive control is not executed for a Singleton Session Bean by the EJB container. Implementation related to exclusive control is required during the application development. Implement using `Synchronized` and `volatile` in the Java language depending on the purpose of usage.

2.19.2 Error handling in Singleton Session Beans

The following errors might occur during the initialization of a Singleton Session Bean:

- DI failure
- Occurrence of a system exception in the constructor method
- Occurrence of an exception in the lifecycle callback method (`PostConstruct` or `PreDestroy`)

If initialization of a Singleton Session Bean fails, `javax.ejb.NoSuchEJBException` occurs for invoking the business interface method of a Singleton Session Bean.

If initialization is successful, the instance of the Singleton Session Bean is retained until the application stops. The instance of the Singleton Session Bean is not destroyed even if a system exception is thrown from the business method or the callback method of a Singleton Session Bean.

Important note

With Application Server, you cannot invoke the asynchronous method of a Singleton Session Bean from the `PostConstruct` method of a Singleton Session Bean.

2.19.3 Precautions when using Singleton Session Beans

- Processing of threads waiting for acquisition of Write lock is not interrupted by stopping the application.
- If a circular dependency is specified in the `@DependsOn` annotation, an error occurs in the deploy processing when the application starts.
- With Application Server, you cannot specify the `@DependsOn` annotation for a Singleton Session Bean for different EJB-JAR files.
- In the lifecycle callback method (`PostConstruct` or `PreDestroy`) of a Singleton Session Bean, specification of the transaction attribute of the `@TransactionAttribute` annotation is invalid.

3 EJB Client

This chapter explains the functionality that can be used in the EJB client. The EJB client is a client program that invokes Enterprise Beans.

3.1 Organization of this chapter

An *EJB client* is a client program that invokes the Enterprise Beans executed by the EJB container on a J2EE server.
The types of EJB clients are as follows:

- EJB client application
- Web applications such as servlets or JSPs
- Other Enterprise Beans

An *EJB client application* is a client application that invokes the Enterprise Beans running on a J2EE server.
The following table describes the functionality of an EJB client and the reference location for each functionality:

Table 3-1: Functionality of an EJB client and the reference location for each functionality

Functionality	Reference location
Functionality that can be used in an EJB client	*3.2*
Starting EJB Client Applications	*3.3*
Invoking an Enterprise Bean	*3.4*
Implementing a transaction in EJB client application	*3.5*
Implementing security in EJB client application	*3.6*
Obtaining RMI-IIOP stubs and interfaces	*3.7*
System log output of EJB client application	*3.8*

This chapter mainly describes the functionality of the EJB client applications.

3.2 Functionality that can be used in an EJB client

The following table describes the functionality that can be used in the EJB client. For details on the respective functionality, see the description given in the references. Note that *uCosminexus Application Server* is omitted from the manual names mentioned in the *Reference* column.

Table 3-2: Functionality that can be used in the EJB client

Category		Overview of the functionality	Reference Manual	Reference location
JNDI	Basic functionality	This functionality enables to search and obtain the EJB home object reference and the reference of business interface.	*Common Container Functionality Guide*	*Chapter 2*
	Extended functionality	In a system consisting of multiple Naming Services and J2EE servers, this functionality enables to execute lookup from the EJB client by round robin. This makes load balancing possible.		
		This functionality enables to maintain the objects looked up from the Naming Service (cached) on the memory. By using the cache, the performance-related cost for accessing the Naming Service can be reduced.		
EJB		The Enterprise Beans running in the EJB container cannot be invoked.	This manual	*2.2*, *3.4*#, *3.7*#
		When a communication failure occurs during EJB invocation, the send operation can be selected.	This manual	*2.13.3*
Transaction		A transaction can be started and concluded in the EJB client.	*Common Container Functionality Guide*	*Chapter 3*
Security		User authentication can be performed by using the user and the password defined in the J2EE server. After logging on from the EJB client, the business methods of the EJB with the security role specified can be invoked.	This manual	*3.6*#
			Security Management Guide	*Chapter 6*
Others		A communication timeout can be set for communication between the EJB client and the Naming Service, and between the EJB client and the J2EE server.	This manual	*2.11*
		The performance analysis trace of the EJB client can be output.	*Maintenance and Migration Guide*	*4.6*

#
The implementation method of the application is described here.

The following table describes the extended functionality that can be used in the EJB client application. For details on the respective functionality, see the description given in the reference. Note that data source (JDBC) and connector (Connector) cannot be used in an EJB client application.

Table 3-3: Expansion functionality that can be used in an EJB client application

Category	Overview of the functionality	for the functionality
Transaction	In an EJB client application, you can obtain UserTransaction, and start or conclude the transaction. UserTransaction is obtained by either of the following methods: 1. Using the `UserTransactionFactory` class	*3.5*

Category	Overview of the functionality	for the functionality
Transaction	2. Using lookup Note that the method given in 1. is recommended in Cosminexus.	*3.5*
Others	The EJB client log can be output.	*3.8*#
	EJB client application can be started using the command (`cjclstartap` command).	*3.3.1*

#

This manual describes the system log of an EJB client application. For details on the user log output by the EJB client applications, see 9. *Output of the Application User Log* in the *uCosminexus Application Server Expansion Guide*.

The functionality of an EJB client application is described in the following sections.

3.3 Starting EJB Client Applications

This section describes how to start an EJB client application.

Start the EJB client application using a command.

The following table describes the organization of this section:

Table 3-4: Organization of this section (Starting EJB client applications)

Category	Title	Reference location
Description	Command used for starting an EJB client application	*3.3.1*
	Using the `cjclstartap` command	*3.3.2*
	Using the `vbj` command	*3.3.3*
Setup	Specifying the environment variables required for executing an EJB client application	*3.3.4*
	Specifying the property of an EJB client application	*3.3.5*

Note:
There is no specific description of *Implementation*, *Operation*, and *Notes* for this functionality.

This section describes the commands used for starting the EJB client application and the flow of starting the EJB client application for each command used.

3.3.1 Commands used for starting an EJB client application

The following table describes the commands used for starting an EJB client application:

Table 3-5: Commands used for starting an EJB client application

Command name	Description
`cjclstartap`	The `cjclstartap` command enables you to code beforehand the options and property required for executing an EJB client application in the option definition files and the property files. Note that if the options and property are omitted for each file, the command is executed by using the default value. Hitachi recommends that in Cosminexus you operate the EJB client applications in the `cjclstartap` command that strengthens the troubleshooting functionality. Furthermore, you can also use the `cjclstartap` command to start Java applications. The location to save the command is as follows: • In Windows *Cosminexus-installation-directory*`\CC\client\bin` • In UNIX `/opt/Cosminexus/CC/client/bin`
`vbj`	In the case of the `vbj` command, the options and property required to execute the EJB client applications must be specified as the arguments of the command. You cannot omit the options and property. The `vbj` command is used for the compatibility with the old versions. Use this command when you want to execute the EJB client applications using the `vbj` command according to the past methods. The location to save the command is as follows: • In Windows *Cosminexus-installation-directory*`\TPB\bin\vbj` • In UNIX `/opt/Cosminexus/TPB/bin/vbj`

Important note

When you use uCosminexus Client to create the EJB client environment, replace *Cosminexus-installation-directory*\CC in the storage directory with *Cosminexus-installation-directory*\CCL.

The flow of starting an EJB client application with each of the above commands is described in the following subsections.

3.3.2 Using the cjclstartap command

The following is the flow of starting an EJB client application using the `cjclstartap` command:

1. Specify the environment variables required for executing the EJB client application.

   Among all the environment variables required for executing the EJB client application, specify only the environment variables that are required when you execute the EJB client application using the `cjclstartap` command. For details on the required environment variables, see *3.3.4 Specifying the environment variables required for executing an EJB client application*.

2. In the option definition file (`usrconf.cfg`) of the EJB client application, specify the Java options and the class path of the JAR files.

   Storage location of `usrconf.cfg`

   A sample of `usrconf.cfg` is saved in the following location. Copy this sample file to any location of your choice, and then use.

   In Windows

   Cosminexus-installation-directory\CC\client\templates\usrconf.cfg

   In UNIX

   /opt/Cosminexus/CC/client/templates/usrconf.cfg

   Specifying the JavaVM startup options

   Specify the JavaVM startup options in the `add.jvm.arg` key of `usrconf.cfg`. For details on the options that can be specified, see *16. JavaVM Startup Options* in the *uCosminexus Application Server Definition Reference Guide*.

   Specifying the class path of the JAR files

   Specify the class path of the JAR file in the `add.class.path` key of `usrconf.cfg`. For details on the JAR files that you must specify in the class path, see *3.7.4 Specifying JAR files in the class path of the EJB client application*.

3. Specify the properties in the property file (`usrconf.properties`) of the EJB client application.

   Storage location of `usrconf.properties`

   A sample of `usrconf.properties` is saved in the following location. Copy this sample file to any location of your choice, and then use the file.

   In Windows

   Cosminexus-installation-directory\CC\client\templates\usrconf.properties

   In UNIX

   /opt/Cosminexus/CC/client/templates/usrconf.properties

   Specifying properties

   For details on the contents that can be specified in the properties, see *3.3.5 Specifying the property of an EJB client application*. If necessary, also see *3.5.3 Precautions during the implementation of a transaction in the EJB client application*, *3.8 System log output of an EJB client application*, and *9.10 User log output settings for EJB client applications (when using the cjclstartap command)* in the *uCosminexus Application Server Expansion Guide*.

   For details on the properties that can be specified, see *14.3 usrconf.properties (User property file for Java applications)* in the *uCosminexus Application Server Definition Reference Guide*.

4. When `usrconf.cfg` and `usrconf.properties` are saved in a location other than the current directory in which the `cjclstartap` command is executed, specify the absolute path of the storage location of `usrconf.cfg` and `usrconf.properties` with the environment variable `CJCLUSRCONFDIR`.

Save the created files `usrconf.cfg` and `usrconf.properties` in the same directory and specify the absolute path of that directory with the environment variable `CJCLUSRCONFDIR`.

This operation is not required when `usrconf.cfg` and `usrconf.properties` are saved in the current directory in which the `cjclstartap` command is executed. Proceed to step 5.

5. Use the `cjclstartap` command to start the EJB client application.

   For details on the `cjclstartap` commands, see *cjclstartap (Starting Java applications)* in the *uCosminexus Application Server Command Reference Guide*.

   ! Important note

   When you use uCosminexus Client to create the EJB client environment, replace *Cosminexus-installation-directory*`\CC` in the storage directory with *Cosminexus-installation-directory*`\CCL`.

When `false` is set up in `ejb.client.directory.shareable`, and the `cjclstartap` command is executed, the work file that will be used by the command is created. If the work file is corrupt, the operation of the `cjclstartap` command or `cjcldumpap` command will not be guaranteed. The output destination of the work file and the file name are as follows:

In Windows

Current-directory`\.cjclstartap.lock`

ejb-clien-log-directory`\.ejbclientlog.lock`

Current-directory`\cjclstartap.pid`

In UNIX

Current-directory`/.cjclstartap.lock`

ejb.client.log-directory`/.ejbclientlog.lock`

Current-directory`/cjclstartap.pid`

Current-directory`/.COSMINEXUS_CC_EJBCLIENT_`*ProcessID*

Reference note

You can also use the `cjclstartap` command to start Java applications. For details on how to start Java applications by using the `cjclstartap` command, see *cjclstartap (Starting Java applications)* in the *uCosminexus Application Server Command Reference Guide*.

3.3.3 Using the vbj command

This subsection describes the flow of starting an EJB client application using the `vbj` command. If you use a batch file or shell script file to execute an EJB client application, you can describe the following contents in the batch file or shell script file:

1. Specify the environment variables required for executing an EJB client application.

   Among all the environment variables required for executing the EJB client application, specify only the environment variables that are required while executing the EJB client application with the `vbj` command. For details on the required environment variables, see *3.3.4 Specifying the environment variables required for executing an EJB client application*.

2. **Specify the JavaVM startup options.**

   Specify the appropriate JavaVM startup options in the `vbj` command. For details on the options that can be specified, see *16. JavaVM Startup Options* in the *uCosminexus Application Server Definition Reference Guide*.

3. **Specify the class path of the JAR files.**

   Specify the appropriate class paths in the `vbj` command. For details on the required class paths, see *3.7.4 Specifying JAR files in the class path of the EJB client application*.

4. **Specify the properties.**

   For details on the contents that can be specified in properties, see *3.3.5 Specifying the property of an EJB client application*. If necessary, also see *3.8 System log output of an EJB client application*, *3.5.3 Precautions during the implementation of a transaction in the EJB client application*, and *9.11 Implementing and setting up the user log output for EJB client applications (when using the vbj command)* in the *uCosminexus Application Server Expansion Guide*.

For details on the properties that can be specified, see *14.3 usrconf.properties (User property file for Java applications)* in the *uCosminexus Application Server Definition Reference Guide*.

5. Use the `vbj` command to start the EJB client application.

! Important note

When you use uCosminexus Client to create the EJB client environment, replace *Cosminexus-installation-directory*`\CC` in the storage directory with *Cosminexus-installation-directory*`\CCL`.

3.3.4 Specifying the environment variables required for executing an EJB client application

This subsection describes the setup of environment variables of an EJB client application. The environment variables required for executing an EJB client application are as follows:

Table 3-6: Environment variables required for executing an EJB client application (in Windows)

Environment variable	Value	Commands	
		`cjclstartap`	`vbj`
`PATH`#1	*Cosminexus-installation-directory*`\jdk\bin`	--	Y
	Cosminexus-installation-directory`\TPB\bin`	Y	--
	Cosminexus-installation-directory`\PRF\bin`	--	Y
`VBROKER_ADM`	*Cosminexus-installation-directory*`\TPB\adm`	Y	Y
`PRFSPOOL`#2	*Cosminexus-installation-directory*`\PRF\spool`	O	O
`TZ`	`JST-9` (In the case of Japan)	Y	Y

Legend:

Y: The environment variables must be specified in the command. Specification is mandatory.

O: Specified during installation. Specification is optional.

--: Specification is not required.

#1

Specify *Cosminexus-installation-directory*`\jdk\bin` at the beginning of the environment variable PATH.

#2

The Cosminexus Performance Tracer log is output under the `PRFSPOOL` environment variable set by the installer. However, if the PRF daemon is not allocated to the machine on which the EJB client application is running, the module trace increases monotonically.

Do not set the `PRFSPOOL` environment variable when the PRF daemon is not allocated.

Use one of the following methods:

- Delete the `PRFSPOOL` environment variable from the system environment variables.
- Disable the `PRFSPOOL` environment variable when you execute the EJB client.

Table 3-7: Environment variables required for executing an EJB client application (in UNIX)

Environment variable	Value	Commands	
		`cjclstartap`	`vbj`
`LIBPATH`, or `LD_LIBRARY_PATH`#1	`/opt/Cosminexus/TPB/lib` `/opt/Cosminexus/PRF/lib`	Y	Y
`PATH`#2	`/opt/Cosminexus/jdk/bin`	--	Y
	`/opt/Cosminexus/TPB/bin`	Y	--
	`/bin`	--	Y

Environment variable	Value	Commands	
		cjclstartap	vbj
PATH#2	/usr/bin	--	Y
VBROKER_ADM	/opt/Cosminexus/TPB/adm	Y	Y
PRFSPOOL#3	/opt/Cosminexus/PRF/spool	O	O
TZ	JST-9 (In the case of Japan)	Y	Y

Legend:

Y: The environment variables must be specified in the command. Specification is mandatory.

O: Specification is optional.

--: Specification is not required.

#1

The name of the environment variable used differs depending on the OS.

LIBPATH: In AIX

LD_LIBRARY_PATH: In HP-UX (IPF), Linux, or Solaris

#2

Specify /opt/Cosminexus/jdk/bin at the beginning of the environment variable PATH.

#3

The Cosminexus Performance Tracer log is output under the PRFSPOOL environment variable. However, if the PRF daemon is not allocated to the machine on which the EJB client application is running, the module trace increases monotonically.

Do not set the PRFSPOOL environment variable when the PRF daemon is not allocated. Specifically disable the PRFSPOOL environment variable when you execute the EJB client.

If AIX is the execution environment, apart from the environment variables described in the above table, you must specify environment variables specific to AIX. For details, see *4.1.11 Items to be checked when setting system environment variables* in the *uCosminexus Application Server System Setup and Operation Guide*.

3.3.5 Specifying the property of an EJB client application

In an EJB client application, you can specify properties corresponding to the functionality to be used. For details on the properties that can be specified in an EJB client application, see *14.3 usrconf.properties (User property file for Java applications)* in the *uCosminexus Application Server Definition Reference Guide*.

3.4 Invoking an Enterprise Bean

This section describes how to invoke an Enterprise Bean from an EJB client application and also describes the operation settings to be specified in the client when a communication failure occurs during the method invocation.

The following table describes the organization of this section:

Table 3-8: Organization of this section (Invoking an Enterprise Bean)

Category	Title	Reference location
Description	Flow of Enterprise Bean invocation from an EJB client application	*3.4.1*
Implementation	Implementation for invoking an Enterprise Bean	*3.4.2*

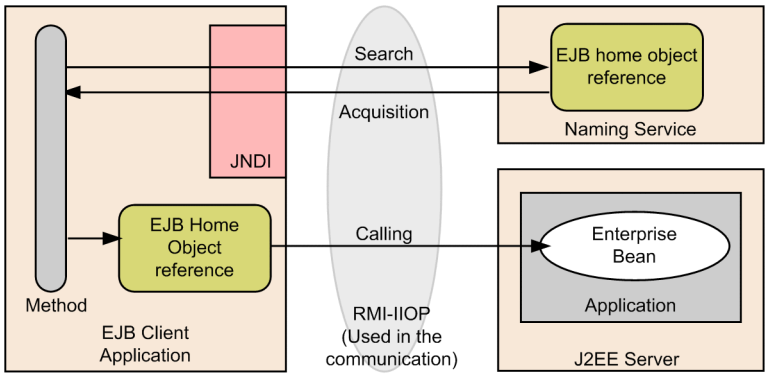
Note:

There is no specific explanation of *Setup*, *Operation*, and *Notes* for this functionality.

3.4.1 Flow of Enterprise Bean invocation from an EJB client application

This subsection describes the flow of invoking an Enterprise Bean based on the example of searching and obtaining the references of an EJB home object.

Figure 3-1: Flow of invoking an Enterprise Bean from the EJB client application by using the home interface



To invoke an Enterprise Bean from the EJB client application using the home interface, obtain the references of the EJB home object using JNDI. To do so, you must implement in such a way so that the JNDI naming context is generated in the EJB client application and the references of the EJB home object can be searched. For details on how to invoke an Enterprise Bean, see *3.4.2 Implementation for invoking an Enterprise Bean*. Furthermore, since an EJB client application uses RMI-IIOP for communication, you can reference the stubs and interfaces of RMI-IIOP. For details on how to obtain the stubs and interfaces of RMI-IIOP, see *3.7 Obtaining RMI-IIOP stubs and interfaces*.

3.4.2 Implementation for invoking an Enterprise Bean

To invoke an Enterprise Bean from the EJB client application, use JNDI. This subsection describes how to invoke an Enterprise Bean when the references of the EJB home object are look up and also when the references of the business interface are look up.

(1) Invoking an Enterprise Bean by searching the references of the EJB home object

The method of invoking an Enterprise Bean using look up the references of the EJB home object is described as follows, based on an example of implementation:

(a) Generating the JNDI naming context

Generate the JNDI naming context to be used for the look up of the references of the EJB home object.

```
javax.naming.Context ctx = new javax.naming.InitialContext();
```

(b) Searching and obtaining the references of the EJB home object

Use the generated JNDI naming context to obtain the references of the EJB home object. To obtain the references of the EJB home object, perform lookup with either the automatically bound name (Portable Global JNDI name or a name starting with HITACHI_EJB) or the name provided by using the user-specified name space functionality. In the following example, lookup is performed using the user-specified name space and the references are obtained. For details on how to perform lookup, see *2.3 Binding to and looking up an object in the JNDI namespace* in the *uCosminexus Application Server Common Container Functionality Guide*.

```
String ejbName = "MySample";
java.lang.Object obj = ctx.lookup(ejbName);
SampleHome sampleHome =
(SampleHome)javax.rmi.PortableRemoteObject.narrow(obj, SampleHome.class);
```

(c) Generating the Enterprise Beans and invoking the methods

Generate the instances of the Enterprise Beans using the `create` method of the EJB home object. By doing this, the methods of the Enterprise Beans required in the application can be invoked.

```
Sample remoteSample = sampleHome.create();
//Generate Enterprise Bean instances
String result = remoteSample.getData("data");
//Invoke the business methods
```

In the Entity Bean, when you use the `find` method that returns the collection type, the objects obtained from the collection must be narrowed in the Enterprise Bean class.

```
Collection c = home.findByXXX(keyValue);
Iterator i=c.iterator();
while (i.hasNext()) {
  Sample remoteSample=(Sample)javax.rmi.PortableRemoteObject.narrow(i.next(),
Sample.class);
                //Invoke a business method in RemoteSample.
}
```

(2) Invoking an Enterprise Bean by searching the references of the business interface

The method of invoking an Enterprise Bean using look up of the references of the business interface is described as follows as per the implementation example:

(a) Generating the InitialContext

To invoke an Enterprise Bean using the business interface, first of all generate the `InitialContext`.

```
// Generate the InitialContext
InitialContext ctx = new InitialContext();
```

(b) Searching and obtaining the references of the business interface

Use the generated `InitialContext` to obtain the references of the business interface. To obtain the references of the business interface, perform lookup with either the automatically bound name or the name provided by using the user-specified name space functionality. For details on how to look up a business interface using an automatically bound name, see *2.5 Lookup by a name starting with HITACHI_EJB* in the *uCosminexus Application Server Common Container Functionality Guide*.

```
// Obtain the reference of the business interface
Sample sample = (Sample)ctx.lookup("HITACHI_EJB/SERVERS/MyServer/EJBBI/SampleApp/
Sample");
```

(c) Calling a method

When the references of the business interface are obtained, the business method can be invoked.

```
// Invoke the business method
String result = sample.getData("data");
```

3.5 Implementing a transaction in an EJB client application

This section describes the implementation of a transaction in an EJB client application.

The following table describes the organization of this section:

Table 3-9: Organization of this section (Implementing a transaction in an EJB client application)

Category	Title	Reference location
Implementation	Procedure for using a transaction in the EJB client	*3.5.1*
	Obtaining UserTransaction using lookup	*3.5.2*
Notes	Precautions during the implementation of a transaction in the EJB client application	*3.5.3*

Note:

There is no specific description of *Operation* for this functionality.

Tip

The following is the description for *Description* and *Setup*:

Description:

See *3. Resource Connection and Transaction Management* in the *uCosminexus Application Server Common Container Functionality Guide*.

Setup:

See *3.20 Points to be considered when starting a transaction in the EJB client applications* in the *uCosminexus Application Server Common Container Functionality Guide*.

! Important note

When you use uCosminexus Client to create the EJB client environment, you cannot use transactions of the EJB client application.

3.5.1 Procedure for using a transaction in the EJB client

To use a transaction in the EJB client application:

1. Add the following JAR files to the class path specified during the startup of the EJB client application:
   - 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`#

   #
   Add `ejbserver.jar` after `HiEJBClientStatic.jar`.
2. Add the system properties required during the startup of the EJB client application.
   For details on how to add system properties, see *3.20 Points to be considered when starting a transaction in the EJB client applications* in the *uCosminexus Application Server Common Container Functionality Guide*.
3. Immediately after the startup of the process of the EJB client application, execute the initialization processing of the service from the user codes implemented in the EJB client application.
   To execute the initialization processing of the service, invoke the `EJBClientInitializer` class (`com.hitachi.software.ejb.ejbclient.EJBClientInitializer`).

Note that if you generate `javax.naming.InitialContext`, and invoke the `getUserTransaction` method of the `UserTransactionFactory` class before invoking the `initialize` method of the `EJBClientInitializer` class, the initialization processing will be executed at that point of time. Therefore, you need not execute the initialization processing in the `EJBClientInitializer` class.

For details on the syntax and functionality of the `EJBClientInitializer` class, see *4.2 EJBClientInitializer class* in the *uCosminexus Application Server API Reference Guide*.

4. Obtain the UserTransaction object.

   The `UserTransaction` object is obtained with either of the following two methods:

   - Using the `UserTransactionFactory` class

     Obtain the `UserTransaction` object using the `getUserTransaction` method of the `com.hitachi.software.ejb.ejbclient.UserTransactionFactory` class. For details on the syntax and functionality of the `UserTransactionFactory` class, see *4.5 UserTransactionFactory class* in the *uCosminexus Application Server API Reference Guide*.

   - Using lookup

     Obtain the `UserTransaction` object using the look up from the naming service. For details on how to obtain the `UserTransaction` object using lookup, see *3.4.8 Process overview and points to be considered when using UserTransaction interface* in the *uCosminexus Application Server Common Container Functionality Guide*.

   Hitachi recommends that in the EJB client you obtain `UserTransaction` using the `UserTransactionFactory` class. However, if you are not able to change the source codes of the EJB client application because of the migration from an application server of another company, use lookup.

5. From the thread that invokes the Enterprise Bean, invoke the `begin` method of the `UserTransaction` interface and start the transaction.
6. Invoke the Enterprise Bean running on the server.
7. From the thread that invokes the Enterprise Bean, invoke the `commit` method or the `rollback` method of the `UserTransaction` interface and conclude the transaction.

3.5.2 Obtaining UserTransaction using lookup

Indicate the search string specified when using look up for `UserTransaction` from the EJB client application.

`HITACHI_EJB/SERVERS/`*`server-name`*`/SERVICES/UserTransaction`

Because the object obtained based on the result of the lookup has `java.lang.Object` type, the object is used by casting in the `javax.transaction.UserTransaction` type.

If a failure occurs in the lookup, the `javax.naming.NamingException` exception will occur.

The specifications for casting in the `UserTransaction` type and for exceptions are the same as that for using the look up for `UserTransaction` from a Web application running on the J2EE server and from an Enterprise Bean

! Important note

The lookup of `UserTransaction` is not supported in the following environments:

- Lookup of `UserTransaction` in the global Naming Service used when linking with CTM
- Lookup of UserTransaction using the user-specified name space management functionality that is used with the round robin search functionality

When using UserTransaction in these environments, use the `UserTransactionFactory` class to obtain `UserTransaction`.

3.5.3 Precautions during the implementation of a transaction in the EJB client application

The precautions to be taken when implementing a transaction in an EJB client are indicated below:

- If an exception occurs during the initialization processing of a service, the system property might not be specified properly. You follow the exception message in such a case.
- If the `mandatory`, `required`, and `supports` attributes are specified in the Container-Managed Transaction (CMT), the invoked Enterprise Bean can be executed within the scope of the transaction started in the EJB client application.
- If an EJB client application shuts down because of a failure during the transaction processing, you need to restart the EJB client application and execute the recovery process of the global transaction. Design the EJB client application in such a way so that after restarting the EJB client application, the `initialize` method of the `EJBClientInitializer` class is invoked and the recovery process of the global transaction starts. Transaction is recovered in the background, therefore, initialize method does not wait until completion of recovery and returns.
- If the transaction is started in an EJB client application, design the EJB client application so that its process always stops after all the transactions are completed. If you stop the processing of the EJB client application without waiting for the transactions to conclude, the transactions in the `Preparing` status might be remained without being concluded. In such a state, you cannot perform the normal termination of Application Server and the resource adapter. Furthermore, the resource lock might not be released.

  For any reason if a transaction that is in the `Preparing` state remains back, you must restart the EJB client application and execute the recovery process of the global transaction.
- In an EJB client application, the root application information and client application information are not included in the performance analysis trace that JTA and OTS output. To trace a request, use a hash code of the thread and XID information. The message output when a transaction timeout occurs includes the hash code of the thread that starts the transaction instead of including the root application information.

3.6 Implementing security in an EJB client application

This section describes the implementation of security in an EJB client application.

In an EJB client application, the user can be authenticated by using the user name and password defined in the J2EE server. If a login is performed by authenticating the user from the EJB client application, you can invoke the methods of an Enterprise Bean in which the `Security` role is specified.

The following table describes the organization of this section:

Table 3-10: Organization of this section (Implementing security in an EJB client application)

Category	Title	Reference location
Implementation	Preconditions for implementing security	*3.6.1*
	Sample program when security is implemented	*3.6.2*

Note:
There is no specific description of *Setup*, *Operation*, and *Notes* for this functionality.

Tip

The following is the explanation for *Description*:

Description:
See *6. Authentication by application settings* in the *uCosminexus Application Server Security Management Guide.*

3.6.1 Preconditions for implementing security

Security is implemented in an EJB client application with the APIs that Cosminexus provides. This subsection describes the preconditions for implementing security and also describes how to implement security. For details on the functionality and syntax of APIs, see *4. APIs Used in the EJB Client Applications* the *uCosminexus Application Server API Reference Guide*.

Before implementing security, make sure that the following preconditions are fulfilled:

- The user must be registered in the J2EE server.
- The `Security` role must be specified for the registered user.

(1) Implementing security

To implement security in an EJB client application:

1. Import the package of the security API.
   To use the security API, import the following package:

```
import com.hitachi.software.ejb.security.base.authentication.*
```

2. Obtain the `LoginInfoManager` object.
   Obtain the `LoginInfoManager` object with the program that invokes the methods of the Enterprise Bean. To obtain the object, use the `getLoginInfoManager` method of the static method prepared for the `LoginInfoManager` object.

```
LoginInfoManager lm = LoginInfoManager.getLoginInfoManager();
```

3. Log in with the user name and password.
   After obtaining the `LoginInfoManager` object, invoke the `login` method.

```
lm.login(username, password);
```

4. Invoke the methods of the Enterprise Bean.

After the successful execution of the `login` method, invoke the methods of the Enterprise Bean.

5. Perform a logout.
   After the invocation of the Enterprise Bean methods is complete, log out from the J2EE server with the `logout` method.

```
lm.logout();
```

! Important note

When implementing security in an EJB client application, you must add `HiEJBClientStatic.jar` in the class path and perform compilation.

3.6.2 Sample program when security is implemented

When the name of the Enterprise Bean is *account*, the sample program for invoking the `getAccountID` method will be as follows:

```
import com.hitachi.software.ejb.security.base.authentication.*;
    ...
  try {
    LoginInfoManager lm = LoginInfoManager.getLoginInfoManager();
    String userName = System.getProperty("username");
    String password = System.getProperty("password");
    if(lm.login(userName , password)) {
      try {
        System.out.println("user:" + userName + "login success");
        Context ctx = new InitialContext();
        java.lang.Object obj = ctx.lookup(appUnitPath + "Account");
        AccountHome aHome =
            (AccountHome)PortableRemoteObject.narrow(obj,AccountHome.class);
        Account account  = aHome.create();
        account.getAccountID();
      } finally {
        lm.logout();
      }
    }
  } catch(NotFoundServerException e) {
    System.out.println("not found server");
  } catch(InvalidUserNameException e) {
    System.out.println("invalid user name");
  } catch(InvalidPasswordException e) {
    System.out.println("invalid password");
  } catch(Exception e) {
    e.printStackTrace();
  }
```

3.7 Obtaining RMI-IIOP stubs and interfaces

This section describes the acquisition of the RMI-IIOP stubs and interfaces.

The following table describes the organization of this section:

Table 3-11: Organization of this section (Obtaining RMI-IIOP stubs and interfaces)

Category	Title	Reference location
Description	Overview of obtaining RMI-IIOP stubs and interfaces	*3.7.1*
	Manual download with server management commands	*3.7.2*
	Dynamic class loading	*3.7.3*
Setup	Specifying JAR files in the class path of the EJB client application	*3.7.4*
Notes	Precautions during the use of uCosminexus Client	*3.7.5*

Note:
There is no specific description of *Implementation*, and *Operation* for this functionality.

3.7.1 Overview of obtaining RMI-IIOP stubs and interfaces

An EJB client application uses the RMI-IIOP functionality of Cosminexus TPBroker to invoke applications.

When searching and obtaining the references of an EJB home object, settings must be specified in such way so that the following stubs and classes can be referenced in the EJB client application:

- Stubs of the EJB objects of an Enterprise Bean
- Stubs of the EJB home object of an Enterprise Bean
- All classes in which the stubs are used

Furthermore, you can reference the following interfaces and all classes from the EJB client application:

- Remote interface
- Home interface
- All classes that reference the interfaces

If the Enterprise Beans are invoked using a business interface, you can reference the business interface and the classes used for invoking the business interface.

These classes (RMI-IIOP stubs and RMI-IIOP interfaces) are downloaded using the server management commands or dynamic class loading.

Tip

Differentiating the use of the server management commands and the dynamic class loading

Instead of using the dynamic class loading, the use of server management commands for downloading the classes required for invocation is better from the viewpoint of performance. Therefore, Hitachi recommends that you use server management commands during the actual operation. On the other hand, Hitachi recommends the use of dynamic class loading for obtaining and updating the stubs during development and testing because the dynamic class loading does not consume much time.

3.7.2 Manual download with server management commands

You can download the RMI-IIOP stubs and RMI-IIOP interfaces with the `cjgetstubsjar` server management command. For details on how to use the `cjgetstubsjar` command, see *10.7 Obtaining the RMI-IIOP stub and interface* in the *uCosminexus Application Server Application Setup Guide*.

3.7.3 Dynamic class loading

Start the EJB client application without specifying the RMI-IIOP stubs in the class path. The RMI-IIOP stubs and RMI-IIOP interfaces will be read automatically when the EJB client application invokes the Enterprise Bean.

To use dynamic class loading of RMI-IIOP stubs:

1. Edit the Easy Setup definition file.
   Specify `true` in the `ejbserver.DynamicStubLoading.Enabled` parameter in the `<configuration>` tag of the logical J2EE server (`j2ee-server`) of the Easy Setup definition file.
2. Restart the corresponding J2EE server.
   Start the J2EE server. If the J2EE applications are already running, stop them once and then restart the J2EE servers.

! Important note

The precautions to be taken for using the dynamic class loading are as follows:

- When using dynamic class loading, you cannot start more than one J2EE application having Enterprise Beans with the same package name and interface name, in a single J2EE server. You cannot start more than one J2EE application even when the name of the parent class that acts as the inheritance source is the same. Here, the parent class that acts as the inheritance source is a class created by the user and not a class provided by J2SE and J2EE.
- If the stubs are not specified in the class path of the client program, you cannot recover the Handle (`javax.ejb.Handle`) of the serialized Enterprise Bean object.
- You cannot use the dynamic class loading functionality when executing the look up of the global CORBA Naming Service of CTM.
- If you start the J2EE server by specifying the -`nosecurity` option in the `cjstartsv` command, you cannot use the dynamic class loading when a J2EE application, which operates on the J2EE server, operates as an EJB client.

3.7.4 Specifying JAR files in the class path of the EJB client application

This subsection describes how to set up JAR files in the class path of the EJB client application. The method of setting the JAR file to the class path differs depending upon the command used to invoke the EJB client application.

- Using the `cjclstartap` command
  When using the `cjclstartap` command, set up the JAR files in the class path with the option definition file (`usrconf.cfg`) of the EJB client application.
  For details on `usrconf.cfg` (option definition file for Java application), see *14.2 usrconf.cfg (option definition file for Java application)* in the *uCosminexus Application Server Definition Reference Guide*.
- Using the `vbj` command
  When using the `vbj` command, set up the JAR files with the batch file or shell script file, or with the command arguments.

The following table describes the JAR files required for executing an EJB client application:

Table 3-12: JAR files required for executing the EJB client application

JAR file name	[Type][#1] JAR file location	Contents included	Commands	
			`cjclstartap`	`vbj`
`hitj2ee.jar`	[Fixed] • In Windows *Cosminexus-installation-directory*`\CC\lib` • In UNIX `/opt/Cosminexus/CC/lib`	Class provided with the product	--	Y

JAR file name	[Type]#1 JAR file location	Contents included	Commands	
			cjclstartap	vbj
HiEJBClientStatic.jar	[Fixed] • In Windows *Cosminexus-installation-directory*\CC\client\lib • In UNIX /opt/Cosminexus/CC/client/lib	Class provided with the product	--	Y
vbjorb.jar vbsec.jar	[Fixed] • In Windows *Cosminexus-installation-directory*\TPB\lib • In UNIX /opt/Cosminexus/TPB/lib	Class provided with the product	--	Y
cprf.jar	[Fixed] • In Windows *Cosminexus-installation-directory*\PRF\lib • In UNIX /opt/Cosminexus/PRF/lib	Class provided with the product	--	Y
hntrlibMj.jar or hntrlibMj64.jar #2 #3	[Fixed] • In Windows *Program-files*\Hitachi\HNTRLib2\classes • In UNIX /opt/hitachi/HNTRLib2/classes	Class provided with the product	--	Y
tpotsinproc.jar	[Transaction used] • In Windows *Cosminexus-installation-directory*\TPB\lib • In UNIX /opt/Cosminexus/TPB/lib	Class provided with the product	--	O
ejbserver.jar	[Transaction used] • In Windows *Cosminexus-installation-directory*\CC\lib • In UNIX /opt/Cosminexus/CC/lib		O	O
stubs.jar	[RMI-IIOP stubs] Either download from the J2EE server, or use dynamic class loading	RMI-IIOP stubs • Stubs of the EJB object • Stubs of the EJB home object • Classes in which the stubs are referenced	O	O
numeric-value.jar	[RMI-IIOP interfaces] Download from the J2EE server	RMI-IIOP interfaces • Remote interface	Y	Y

JAR file name	[Type]#1 JAR file location	Contents included	Commands	
			cjclstartap	vbj
numeric-value.jar	[RMI-IIOP interfaces] Download from the J2EE server	• Home interface • Classes in which interfaces are referenced	Y	Y
User-created JAR file	User-created class	This is a user-created class used in the EJB client application. If the unique Filter class, the Formatter class or the Handler class created by the user is used in the user log functionality of the EJB client application, specify these classes also in the class path.#3	Y	Y

Legend:

Y: Must be specified in the class path.

O: If necessary, specify in the class path.

--: Need not be specified in the class path.

#1

The JAR files include the following types:

- [Fixed]
  The file name and location of the specified JAR file are fixed, irrespective of the number of applications invoked from the EJB client application or the contents.
- [Transaction used]
  If a transaction is used in the EJB client application, specify the corresponding JAR file. For details, see *3.20 Points to be considered when starting a transaction in the EJB client applications* in the *uCosminexus Application Server Common Container Functionality Guide*.
- [RMI-IIOP stubs]
  Specify the JAR file corresponding to each application invoked from the EJB client application. Need not be specified when using dynamic class loading.
- [RMI-IIOP interfaces]
  If the RMI-IIOP interfaces are not obtained in the EJB client application, download and specify the JAR file corresponding to each application. If the RMI-IIOP interfaces are already obtained in the EJB client application, specify the obtained classes or the JAR files.

#2

Use a JAR file corresponding to the OS you are using. For HP-UX (IPF) and Linux (IPF), specify hntrlibMj64.jar. For other operating systems, specify hntrlibMj.jar. Note that if these JAR files are specified, output mode of the system log changes to a shared subdirectory mode. For details on the shared subdirectory mode, see *3.8.2 Output destination subdirectory of the system log*.

#3

Specify for using the user log output functionality of the EJB client application. For details on how to specify the user log output settings for the EJB client applications, see *9.8 User log output settings for J2EE applications* in the *uCosminexus Application Server Expansion Guide*.

! Important note

- When you use uCosminexus Client to create the EJB client environment, replace *Cosminexus-installation-directory*\CC in the storage directory with *Cosminexus-installation-directory*\CCL.
- When you use uCosminexus Client to create the EJB client environment, you cannot use the transactions of the EJB client application.
- When setting up JAR files in the class path, pay attention to the setup order of the JAR files.
  When using a transaction, set up tpotsinproc.jar and ejbserver.jar in the class path. In such a case, set up HiEJBClientStatic.jar even before ejbserver.jar.

When using the performance analysis trace functionality, set up `cprf.jar` in the class path. In such a case, set up `cprf.jar` before setting up `HiEJBClientStatic.jar`.

If you set up these files in the reverse order, an attempt to initialize the performance analysis trace will fail. Furthermore, if you specify these files in the reverse order, the message KDJE51008-W will be output with the reason code `-4` when the log level of the EJB client application is set to `Warning` or higher. Note that when an attempt to initialize the performance analysis trace fails, the performance analysis trace will not be output, but you can continue with the processing of the EJB client application.

3.7.5 Precautions during the use of uCosminexus Client

The `cjgetstubsjar` command is not included in uCosminexus Client. Therefore, you cannot obtain the stubs and the interfaces with the server management commands. The following is the procedure for manually obtaining the RMI-IIOP stubs in uCosminexus Client:

1. Execute the `cjgetstubsjar` command on the machine on which Application Server is running and place the file containing RMI-IIOP stubs and interfaces in any directory of your choice.
2. From the machine on which uCosminexus Client is running, access the machine on which Application Server is running, and then download the file containing the RMI-IIOP stubs and the interfaces through the file transfer.

3.8 System log output of an EJB client application

This section describes the system log output of an EJB client application.

The following table describes the organization of this section:

Table 3-13: Organization of this section (System log output of an EJB client application)

Category	Title	Reference location
Description	Overview of the system log of an EJB client application	*3.8.1*
	Output destination subdirectory of the system log	*3.8.2*
Setup	Changing the output destination and output level of the system log	*3.8.3*
	Sharing the log output destination subdirectory among multiple processes	*3.8.4*
	Setting up the access permission of the log output destination directory	*3.8.5*

Note:

There is no specific description of *Implementation*, *Operation*, and *Notes* for this functionality.

3.8.1 Overview of the system log of an EJB client application

There are three types of logs that output in the system log of an EJB client application; the message log, the exception log, and the maintenance log. In an EJB client application, you can change the settings of the output destination and output level of the system log, and the output destination subdirectory, as and when required.

3.8.2 Output destination subdirectory of the system log

The system log of an EJB client application is output in each process of the EJB client application. In the system log, you can share the log output destination subdirectory among the multiple processes. The operation mode for sharing the log output destination subdirectory among the multiple processes is called the *shared subdirectory mode*.

Tip

If you want to use an existing EJB client application created with a version earlier than 06-50 as it is, perform the operation in the exclusive subdirectory mode in which the log output destination subdirectory is created for each process. We recommend that you use the shared subdirectory mode for creating a new EJB client application because the exclusive subdirectory mode is used for compatibility with versions earlier than 06-50.

The following table describes operations in the shared subdirectory mode.

Table 3-14: Differences between the shared subdirectory mode and exclusive subdirectory mode

Items	Sub directory shared mode
Possibility of sharing a subdirectory among multiple processes	Can be shared.
Creation of the log management file	Created.
Default value of the `ejbserver.client.ejb.log` key[#]	System
Default value of the `ejbserver.client.log.appid` key[#]	Ejbcl
Specification of the `ejbserver.client.log.directorynum` key[#]	Always disabled.
Number of files that can be specified in the `ejbserver.logger.channels.define.`*channel-name*`.filenum` key[#]	1 to 64
Size (bytes) that can be specified in the `ejbserver.logger.channels.define.`*channel-name*`.filesize` key[#]	4,096 to 16,777,216

#
System property specified when an EJB client application is started.

Important note

- When executing an EJB client application with the `cjclstartap` command, use the shared subdirectory mode.
- When using the user log functionality of an EJB client application, use the shared subdirectory mode.

- **Specifying the operation mode**
  For details on specifying the class path for using the shared subdirectory mode, see *3.7.4 Specifying JAR files in the class path of the EJB client application*.
- **Sharing a subdirectory**
  When you are using the shared subdirectory mode, you can share the log output destination subdirectory. For details on sharing a subdirectory, see *3.8.4 Sharing the log output destination subdirectory among multiple processes*.

3.8.3 Changing the output destination and output level of the system log

(1) Setting up the system log

The method of setting the system log of the EJB client application differs depending upon the command used to invoke the EJB client application.

- **Using the `cjclstartap` command**
  When using the `cjclstartap` command, set up the properties of the system log with the property file of the EJB client application (`usrconf.properties`).
- Using the `vbj` command
  When using the `vbj` command, set up the properties of the system log with a batch file or shell script file or with the command arguments.

(2) Setting up the output destination and output level of the system log

You can change the attributes of the system log of an EJB client application by customizing either the option definition file for Java applications (`usrconf.cfg`), or the user property file for Java applications (`usrconf.properties`). The following table describes the items that can be changed and the keys of properties with which changes can be specified. This table also describes the necessity of specifying the properties that are to be specified in the execution commands of the EJB client application.

Table 3-15: Keys used to change the output destination and output level of the system log of an EJB client application

Items that can be changed	Key of the option definition file for Java applications	Key of the user property file for Java applications	Type
Log output destination[#1]	`ejb.client.log.directory`	`ejbserver.client.log.directory`	Variable
Name of the log output destination directory created in each EJB client application[#1]	`ejb.client.ejb.log`	`ejbserver.client.ejb.log`	Selective variable
Name of the log output destination subdirectory created in each process of the EJB client application[#1]	`ejb.client.log.appid`	`ejbserver.client.log.appid`	Selective variable
Termination of message output in standard output[#2]	`ejb.client.log.stdout.enabled`	--	Selective variable

Items that can be changed	Key of the option definition file for Java applications	Key of the user property file for Java applications	Type
Number of log files	--	`ejbserver.logger.channels.define.`*channel-name*[#3]`.filenum`	Selective variable
Log file size	--	`ejbserver.logger.channels.define.`*channel-name*[#3]`.filesize`	Selective variable
Log output level[#4]	--	`ejbserver.logger.enabled.*`	Selective variable
Output destination of the trace file of Cosminexus TPBroker	--	`vbroker.orb.htc.tracePath`	Selective variable
Number of trace files of Cosminexus TPBroker	--	`vbroker.orb.htc.comt.fileCount`	Selective variable
Number of entries in each trace file of Cosminexus TPBroker	--	`vbroker.orb.htc.comt.entryCount`	Selective variable

Legend:

Variable: The value must be specified according to the execution environment of the system.

Selective variable: Either specify the value according to the execution environment of the system or omit the specification.

--: Cannot be specified

#1

In `usrconf.cfg`, you can change the settings of the operation log, log operation log, exception information during the occurrence of a failure and maintenance information.

Furthermore, if the same items are set up in `usrconf.cfg` and `usrconf.properties`, the contents specified in `usrconf.properties` will be given priority.

#2

You can specify the settings to prevent the output of messages of the operation log, `cjclstartap` command log and startup process standard output information in standard output.

#3

The following names that indicate the type of the log are set up as channel names:

ClientMessageLogFile (Operation log) (File name: `cjclmessage`*n*`.log`)

ClientExceptionLogFile (Exception information during the occurrence of a failure) (File name: `cjclexception`*n*`.log`)

ClientMaintenanceLogFile (Maintenance information) (File name: `cjclmaintenance`*n*`.log`)

#4

When you set up the system properties using a shell script, you cannot specify the log output level (`ejbserver.logger.enabled.*` key).

Whether or not you can specify system properties will depend on the usage form of the EJB client application. The following table describes the relationship between the usage form of the EJB client application and system properties. Note that the numbers in Table 3-16 and Table 3-17 correspond to each other.

Table 3-16: Usage form of an EJB client application

Types of EJB client applications	Multiplicity of simultaneous startup of EJB client applications		
	1 multiple	2 to 8 multiples	9 to 16 multiples
1 type	1.	2.	3.
2 types or more	4.	5.	6.

Table 3-17: Usage form of the EJB client application and system properties

System property specification	Types of EJB client application usage					
	1.	2.	3.	4.	5.	6.
`ejbserver.client.ejb.log`	Available	Available	Required	Required	Required	Required

System property specification	Types of EJB client application usage					
	1.	2.	3.	4.	5.	6.
ejbserver.client.log.appid	Available	Not available	Not available	Available	Not available	Not available
ejbserver.client.log.directorynum	Available	Available	Required	Available	Available	Required

The description of the numbers specified in the above tables is as follows:

1. You can even specify the default directory in the ejbserver.client.ejb.log key.
   When the ejbserver.client.log.appid key is specified, the specification of the ejbserver.client.log.directorynum key will become invalid.
2. You can even specify the default directory in the ejbserver.client.ejb.log key.
   Because multiple EJB client applications are started simultaneously, do not specify the ejbserver.client.log.appid key. When you specify the ejbserver.client.log.directorynum key, make sure to specify the ejbserver.client.ejb.log key.
3. Make sure to specify the ejbserver.client.ejb.log key in each EJB client application. Because multiple EJB client applications are started simultaneously, do not specify the ejbserver.client.log.appid key. Specify the value of the ejbserver.client.log.directorynum key according to the multiplicity.
4. Make sure to specify the ejbserver.client.ejb.log key in each EJB client application.
   When the ejbserver.client.log.appid key is specified, the specification of the ejbserver.client.log.directorynum key will become invalid.
5. Make sure to specify the ejbserver.client.ejb.log key in each EJB client application. Because multiple EJB client applications are started simultaneously, do not specify the ejbserver.client.log.appid key.
   When you specify the ejbserver.client.log.directorynum key, make sure to specify the ejbserver.client.ejb.log key.
6. Make sure to specify the ejbserver.client.ejb.log key in each EJB client application. Because multiple EJB client applications are started simultaneously, do not specify the ejbserver.client.log.appid key. Specify the value of the ejbserver.client.log.directorynum key according to the multiplicity.

3.8.4 Sharing the log output destination subdirectory among multiple processes

If you are using the shared subdirectory mode, you can share the log output destination subdirectory.

The system log of the EJB client application is saved in the subdirectory below the log output destination directory (the directory specified in the ejbserver.client.log.directory key and ejbserver.client.ejb.log key) created in each EJB client application. Note that you can specify the subdirectory name with the ejbserver.client.log.appid key.

When you are using the shared subdirectory mode, specify a value different from that specified when you are using the exclusive subdirectory mode in the ejbserver.client.ejb.log key that specifies the log output destination directory. If you specify the same value as the exclusive subdirectory mode, you will not be able to manage the number of log output destination subdirectories in the exclusive subdirectory mode. Note that the exclusive subdirectory mode is a mode for achieving compatibility with older versions.

If the EJB client running in the shared subdirectory mode outputs the KDJE90002-E message in the log operation information (cjlogger.log file) and terminates, or if it outputs the KDJE90003-E message, the exclusion processing of the log file might have failed. By increasing the retry frequency and retry interval with the ejbserver.client.log.lockRetryCount and ejbserver.client.log.lockInterval keys, the failure in the exclusion processing of the log file can be recovered.

For details on obtaining the system log of an EJB client application, see *4.5 System log of the EJB client applications* in the *uCosminexus Application Server Maintenance and Migration Guide*.

Note:

If you start multiple EJB client applications simultaneously during the initial startup, the process of log directory generation may collide with other EJB client applications started at the same time, resulting in the output of the `KDJE51003-E` message and abnormal termination. Either do not start multiple EJB client applications simultaneously during the initial startup, or create the log output destination directory beforehand.

3.8.5 Setting up the access permission of the log output destination directory

In UNIX, when you execute an EJB client application using the same log output destination directory in multiple user accounts, you must set up access permission for the log output destination directory.

Specify the mode of the log output destination directory so that the write permission is granted to a group and all other users. Then set up `umask` to `0` and execute the EJB client application.

4 Precautions During the Implementation of Enterprise Beans

This chapter describes the precautions to be taken when implementing an Enterprise Bean.

4.1 Organization of this chapter

This chapter describes the precautions to be taken for implementing Enterprise Beans as the program of an application that is running on the Application Server machine.

The following table describes the organization of this chapter:

Table 4-1: Organization of this chapter (Precautions During the Implementation of an Enterprise Bean)

Category	Title	Reference location
Common precautions for all Enterprise Beans	Rules for naming an Enterprise Bean and related classes	*4.2.1*
	Acquiring and releasing a resource connection	*4.2.2*
	Differentiating the use of a local interface and remote interface	*4.2.3*
	Using the Local Invocation Optimization functionality	*4.2.4*
	Method for invoking an Enterprise Bean of another J2EE application with the component interface	*4.2.5*
	Method for invoking an Enterprise Bean of another J2EE application with the business interface	*4.2.6*
	Precautions concerning the acquisition of a class loader	*4.2.7*
	Precautions during the use of the URLConnection class	*4.2.8*
	Precautions concerning loading of the native library	*4.2.9*
	About the timeout of access exclusion of an Entity Bean (common for CMP and BMP)	*4.2.10*
	About the occurrence of a deadlock during the use of an Entity Bean (common for CMP and BMP)	*4.2.11*
	Precautions regarding the methods of the javax.ejb.EJBContext interface	*4.2.12*
	About the `<prim-key-class>` tag of the Entity Bean (common for CMP and BMP) property file	*4.2.13*
	Precautions concerning EJB specifications	*4.2.14*
	About multi-byte characters	*4.2.15*
	Precautions concerning transmission of Unicode supplementary characters	*4.2.16*
	Precautions concerning API of EJB3.0	*4.2.17*
	Precautions related to `ejb-jar.xml` of EJB 3.0 or later	*4.2.18*
	Precautions related to use Generics	*4.2.19*
	Precautions when using EJB 3.1	*4.2.20*
	About the `getCause()` method	*4.2.21*
	Precautions concerning name of the resource reference	*4.2.22*
	Precautions concerning libraries of Application Server	*4.2.23*
Precautions for each type of Enterprise Beans	Precautions during the implementation of a Stateful Session Bean	*4.3.2*
	Precautions during the implementation of an Entity Bean (BMP)	*4.3.3*
	Precautions during the implementation of an Entity Bean (CMP)	*4.3.4*
	Precautions during the implementation of a Message-driven Bean	*4.3.5*
	Precautions during the implementation of Singleton Session Bean	*4.3.6*

Note:

This chapter provides the description only for *Precautions*.

4.2 Common precautions for all Enterprise Beans

This section describes the common precautions to be taken when implementing Enterprise Beans.

4.2.1 Rules for naming an Enterprise Bean and related classes

When implementing an Enterprise Bean class, home interface, component interface, business interface, interceptor class, and the related classes, follow the naming rules described here:

- Do not place an Enterprise Bean and the related classes in a package beginning with the same name as the class name of these classes.

  For example, you cannot assign a class having the same name as the package name, such as "Example.Example".
- You cannot use a package name beginning with `Wrappers`.
- Use alphanumeric characters and symbols in the name of an Enterprise Bean class, home interface, component interface, and business interface. You cannot use a method name and member variable name beginning with an underscore (_).
- Do not use the following methods as business methods. If you use the following methods as business methods, a compile error might occur when the application starts, and EJB might not execute correctly:
  - The following methods that are defined with `java.lang.Object`:

    `equals(Object)`, `hashCode()`, `toString()`, `clone()`, `finalize()`
  - The following methods that are defined with `javax.ejb.EJBObject` and `javax.ejb.EJBLocalObject`:

    `getEJBHome()`, `getEJBLocalHome()`, `getHandle()`, `getPrimaryKey()`, `isIdentical(EJBLocalObject)`, `isIdentical(EJBObject)`, `remove()`
- In a CMP Entity Bean, you cannot specify a CMP field name and CMR field name beginning with an underscore (_).
- You cannot use a package name and class name in which the only difference is that of upper case and lower case characters. The upper case and lower case characters are not differentiated in RMI-IIOP, and therefore, the Enterprise Bean might not be accessed properly.
- In the case of existence of a class that implements `java.rmi.Remote`, such as the home interface of a JAR file, the stubs will be generated in the work directory. Specify the package name and class name for the class that implements `java.rmi.Remote` in such a way so that the path length of the work directory does not reach the upper limit of the platform. For details on estimating the length of the work directory path, see the following manuals:
  - For J2EE servers

    Appendix C.1 Work directory of J2EE servers in the *uCosminexus Application Server System Setup and Operation Guide*
  - Work directory for batch servers

    Appendix C.2 Work directory of batch servers in the *uCosminexus Application Server System Setup and Operation Guide*
- Even for a separate EJB-JAR file, make sure that the classes are not duplicated in the same application.
- Do not use the reserved names of `VisiBroker` for the class name of the remote interface and remote component interface of an Enterprise Bean. The reserved names of `VisiBroker` include `Helper`, `Holder`, `Package`, `Operations`, `POA`, and `POATie`. For details on the reserved names in VisiBroker, see the manual *Borland(R) Enterprise Server VisiBroker(R) Programmer's Reference*.
- When creating an Enterprise Bean class in the default package, do not create a class ending with the following strings:
  - `_LocalHomeImpl`
  - `_LocalComponentImpl`
  - `_RemoteHomeImpl`
  - `_RemoteComponentImpl`

- `_LocalBIProxyImpl`
- `_LocalBIClientSideProxyImpl`
- `_RemoteBIProxyInterface`
- `_RemoteBIProxyImpl`
- `_RemoteBIClientSideProxyImpl`
- `_CallbackWrapperImpl`
- `_InvocationContextImpl`

4.2.2 Acquiring and releasing a resource connection

If you are using a J2EE resource connection, such as JDBC and JMS in an Enterprise Bean, release the connection with the `close` method of the `Connection` class. If you do not release the connection, the resource might be consumed before you can anticipate the consumption. To avoid such a situation, consider the following points during the implementation:

- **Do not maintain the connection in the member variable of the Enterprise Bean class.**
  If a connection is maintained in the member variable, the connection will be in the usage status irrespective of the execution of the Bean, leading to a decline in the performance. Therefore, the connection must be acquired and released during the execution of the SQL.
- **Set up a connection pool.**
  If you set up a connection pool, the overheads of acquiring a connection to re-use a physical connection can be reduced. When executing under a JTA transaction, you must set up a connection pool.

4.2.3 Differentiating the use of a local interface and remote interface

If you use a local interface added in the EJB 2.0 specifications, the overhead of the RMI-IIOP communication processing can be reduced, but remote invocation cannot be performed for an Enterprise Bean having only local interfaces. Furthermore, when using a local interface, you can operate the method arguments and return the values with the pass by reference functionality. Differentiate the use of a local interface and remote interface based on the following usage conditions of local interfaces:

When the use of a local interface is recommended

Hitachi recommends that you use a local interface when the Enterprise Bean and client (Enterprise Bean or JSP, servlet) are included in the same J2EE application.

When the use of a local interface is mandatory

The use of a local interface is mandatory for a CMP2.x CMR. You must, therefore, use a local interface.

When a local interface cannot be used

You cannot use a local interface in the following cases:

- When the Enterprise Bean and client (Enterprise Bean or JSP, servlet) are included in different J2EE applications.
- When the Enterprise Bean and client (Enterprise Bean or JSP, servlet) are executed on another J2EE server (separate JavaVM).
- When already implemented in EJB 1.1.

4.2.4 Usage of the local invocation optimization functionality

If you use the local invocation optimization functionality when using a remote interface, the overhead of the RMI-IIOP communication processing can be reduced. The format of local invocation is almost the same as that of the method invocation, but the method arguments and return values are processed by the pass by reference functionality.

4.2.5 Method for invoking an Enterprise Bean of another J2EE application with the component interface

This subsection describes how to invoke an Enterprise Bean of another J2EE application with the component interface, when the application is running in the same J2EE server and also when the application is running in another J2EE server.

(1) For an Enterprise Bean running in another application on the same J2EE server

To invoke the Enterprise Bean:

1. Include the remote home interface and remote interfaces of the invoked Enterprise Bean as well as the user-created classes used in the interfaces in the invoking EJB-JAR file or the WAR file.
2. Specify the following property in the user-defined file for J2EE servers:
   Specify `app` in the `ejbserver.deploy.stub.generation.scope` key of `usrconf.properties`.
   For details on `usrconf.properties`, see *14.3 usrconf.properties (User property file for Java applications)* in the *uCosminexus Application Server Definition Reference Guide*.
3. Use the Naming Service switching functionality to specify the lookup name of the Enterprise Bean beginning with `corbaname` and running in another J2EE application.
   For details on the specification method, *see 9.3.1 Reference definition of other Enterprise Beans* in the *uCosminexus Application Server Application Setup Guide*.
   An example of the specification is given below:

```
corbaname::NamingHost:900#HITACHI_EJB/SERVERS/MyServer/EJB/MyApplication/
MyBean
```

(2) For an Enterprise Bean running on another J2EE server

To invoke the Enterprise Bean:

1. Include the remote home interface and remote interfaces of the invoked Enterprise Bean as well as the user-created classes used in the interfaces in the invoking EJB-JAR file or the WAR file.
2. Specify the following property in the user-defined file for J2EE servers:
   Specify `app` in the `ejbserver.deploy.stub.generation.scope` key of `usrconf.properties`.
3. Use the Naming Service switching functionality to specify the lookup name of the Enterprise Bean beginning with `corbaname` and running in another J2EE application.
   For details on the specification method, *see 9.3.1 Reference definition of other Enterprise Beans* in the *uCosminexus Application Server Application Setup Guide*.
   An example of specification is given below:

```
corbaname::NamingHost:900#HITACHI_EJB/SERVERS/MyServer/EJB/MyApplication/
MyBean
```

4.2.6 Method for invoking an Enterprise Bean of another J2EE application with the business interface

This subsection describes how to invoke an Enterprise Bean of another J2EE application with the business interface, when the application is running on the same J2EE server, and also when the application is running on another J2EE server.

(1) For an Enterprise Bean running in another application on the same J2EE server

To invoke the Enterprise Bean:

1. Include the business interface of the invoked Enterprise Bean as well as the user-created classes used in the interface in the invoking EJB-JAR file or the WAR file.

2. If none is specified for the ejbserver.rmi.localinvocation.scope key in the user-defined file for J2EE server, acquire the stubs by using the `cjgetstubsjar` command for the invoked Enterprise Bean and include these stubs in the invoking EAR file.#

   The Enterprise Bean is looked up without using the EJB references. The format for specifying the lookup name is as follows:

   `HITACHI_EJB/SERVERS/`*server-name*`/EJBBI/`*J2EE-APP-name*`/`*Enterprise-Bean-name*

   server-name: J2EE server name

   J2EE-APP-name: Lookup name of the J2EE application

   Enterprise-Bean-name: Lookup name of the Enterprise Bean

   #

   If you use the dynamic class loading functionality for invocation of the business interface between applications, you need not include the stubs. However, Hitachi does not recommend the use of the dynamic class loading functionality from the point of view of performance.

(2) For an Enterprise Bean running on another J2EE server

To invoke the Enterprise Bean:

1. Include the business interface of the invoked Enterprise Bean as well as the user-created classes used in the interface in the invoking EJB-JAR file or the WAR file.
2. Acquire the stubs by using the `cjgetstubsjar` command for the invoked Enterprise Bean and include these stubs in the invoking EAR file.#

   The Enterprise Bean is looked up without using the EJB references. The format for specifying the lookup name is as follows:

   `HITACHI_EJB/SERVERS/`*server-name*`/EJBBI/`*J2EE-APP-name*`/`*Enterprise-Bean-name*

   server-name: J2EE server name

   J2EE-APP-name: Lookup name of the J2EE application

   Enterprise-Bean-name: Lookup name of the Enterprise Bean

   #

   If you use the dynamic class loading functionality for invocation of the business interface between applications, you need not include the stubs. However, Hitachi does not recommend the use of the dynamic class loading functionality from the point of view of performance.

4.2.7 Precautions concerning the acquisition of a class loader

When you acquire the class loaders of Cosminexus Component Container from the codes within the J2EE application and use the following APIs, the `java.net.JarURLConnection` class will be used:

- `getResource(String).openConnection().getInputStream();`
- `getResource(String).openStream()`

By expanding the above methods, the `openConnection` method of the `java.net.JarURLConnection` class will be invoked, and the JAR file specified in the corresponding URL will be opened. You must perform the operations concerning the JAR file, and therefore, when the `openConnection` method of the `java.netJarURLConnection` class is used, make sure to invoke the `close` method of the `JarFile` instance to be returned by `getJarFile` of `java.net.JarURLConnection`. If the `close` method is not invoked explicitly, the JAR file will remain open and cannot be deleted. Furthermore, do not use the above methods in a J2EE application.

4.2.8 Precautions during the use of the URLConnection class

The `java.net.URLConnection` class uses the `setUseCaches(boolean)` method to specify whether or not to use the cache information when acquiring a connection for the specified URL. If the `setUseCaches(false)` method is specified for the `URLConnection` class, the target object will be generated for each connection. When

using this class from the codes within the J2EE application, a memory leak might occur in the JavaVM of the J2EE server.

4.2.9 Precautions concerning loading of the native library

Do not use the `System.loadLibrary` method to load the native library from the Enterprise Bean. If you load the native library in the Enterprise Bean, `java.lang.UnsatisfiedLinkError` might occur due to limitations of the JNI specifications. If you must load the native library, create a container extension library to invoke the `System.loadLibrary` method, and implement in such a way so that the container extension library is referenced from the Enterprise Bean. For details on creating the container extension library, see *14. Container Extension Library* in the *uCosminexus Application Server Common Container Functionality Guide*.

4.2.10 About the timeout of access exclusion of an Entity Bean (common for CMP and BMP)

When using an Entity Bean (CMP or BMP), an exclusion processing is applicable for the access to the Entity Beans of the same primary key. If the Entity Beans of the same primary key are accessed simultaneously, and the processing takes up some time, the exclusion processing is awaited. If exclusion cannot be acquired after waiting for the exclusion processing, the exception `IllegalStateException` occurs with a timeout (default value 45 seconds). In such a case, you can prevent the occurrence of a timeout by specifying the timeout period in the `ejbserver.server.mutex.invocation.timeout` key of the user-defined file for the J2EE servers (`/opt/Cosminexus/CC/server/usrconf/ejb/`*server-name*`/usrconf.properties`). For details on the `ejbserver.server.mutex.invocation.timeout` key, see *2.4 usrconf.properties (User property file for J2EE servers)* in the *uCosminexus Application Server Definition Reference Guide*.

4.2.11 About the occurrence of a deadlock during the use of an Entity Bean (Common for CMP and BMP)

With Application Server, the transaction functionality of the database is used for the transaction processing of an Entity Bean. A deadlock might occur depending on the order of access to the database, setup of the tables concerning data exclusion, setup of the database in the system, and the SQL statement to be invoked. For details, see the manual of the database you are using.

4.2.12 Precautions regarding the methods of the javax.ejb.EJBContext interface

Depending on the transaction management model of the Enterprise Bean, you might not issue the models for the `getUserTransaction` method, `getRollbackOnly` method, or the `setRollbackOnly` method of the `javax.ejb.EJBContext` interface. Furthermore, among the methods of the Enterprise Bean, the methods that are operated with the "Unspecified transaction" status according to the EJB specifications cannot be issued. If a method cannot be issued, the EJB container throws `java.lang.IllegalStateException`. The possibility of issuing each method is described in the following tables.

Table 4-2: Possibility of issuing each transaction management model

`javax.ejb.EJBContext` method	Possibility of issuing	
	BMT	CMT
`getUserTransaction`	Y	--
`getRollbackOnly`	--	--
`setRollbackOnly`	--	Y

Legend:
Y: Can be issued

--: Cannot be issued

Table 4-3: Possibility of issuing each EJB method

Type of Beans	Method	Possibility of issuing
SessionBean	Constructor	--
	setSessionContext	--
	ejbCreate	--
	ejbRemove	--
	ejbPassivate	--
	ejbActivate	--
	Business method	Y
	afterBegin	Y
	beforeCompletion	Y
	afterCompletion	--
EntityBean	Constructor	Y
	setEntityContext	--
	unsetEntityContext	--
	ejbCreate	Y
	ejbPostCreate	Y
	ejbRemove	Y
	ejbHome	Y
	ejbPassivate	--
	ejbActivate	--
	ejbLoad	Y
	ejbStore	Y
	Business method	Y
Message-driven Bean	Constructor	--
	ejbCreate	--
	onMessage	Y
	Methods of the message listener	Y
	ejbRemove	--

Legend:

Y: Can be issued

--: Cannot be issued

4.2.13 About the <prim-key-class> tag of the Entity Bean (common for CMP and BMP) property file

If an interface and abstract class is specified in the <prim-key-class> tag of the Entity Bean property file, perform the operation as follows:

- In the case of Entity Bean (CMP)
  The error message KDJE42039-E is output during the deployment, and the deployment processing terminates with an error.
- In the case of Entity Bean (BMP)
  Deployment and execution can be performed in the same way as is performed when a class is specified.

4.2.14 Precautions concerning EJB specifications

If a description not in compliance with the EJB specifications is given, an error (exception) might occur. Even if no error occurred in a previous version, an error might occur when the version is upgraded. If an error occurs, check the error contents, and make changes according to the EJB specifications.

4.2.15 About multi-byte characters

In the classes that configure EJB, do not use multi-byte characters in the class (package) name, method name, return value type, arguments (type and name) of the methods such as the business method, callback method and interceptor method, and names of exceptions declared in the `throws` clause.

4.2.16 Precautions concerning transmission of Unicode supplementary characters

When Application Server contains Cosminexus TPBroker as component software, the Unicode supplementary characters can be transmitted by invoking the methods of the Enterprise Bean through the RMI-IIOP communication. In such a case, the operation during the transmission of the Unicode supplementary characters will differ depending on the version of Application Server at the sending and receiving sides. The following table describes the combination of versions of the applications servers, and the operation during the transmission of the Unicode supplementary characters.

Table 4-4: Combination of versions of Application Servers, and operation during the transmission of Unicode supplementary characters

Version of the Application Server at the sending side	Version of the Application Server at the receiving side	
	07-50 or later	Before 07-50
07-50 or later	Y	--
Before 07-50	--	--

Legend:
Y: Unicode supplementary characters can be transmitted.
--: Unicode supplementary characters cannot be transmitted.

For a version that does not support transmission of Unicode supplementary characters, either the `java.rmi.RemoteException` exception or the `java.rmi.MarshalException` exception will be thrown during the transmission of the Unicode supplementary characters by the RMI-IIOP communication.

The operation for transmission of the Unicode supplementary characters can be specified so that the operation is the same as for versions prior to 07-50. To set up the same operations as for the versions prior to 07-50, specify `false` in the `vbroker.orb.htc.surrogateCheckOff` key of `usrconf.properties` (user property file for J2EE servers and user property file for the Java applications). For details on the key, see the *TPBroker Users Guide*.

4.2.17 Precautions concerning API of EJB 3.0

The following APIs added in EJB 3.0 are not supported:

- `javax.ejb.EJBContext#lookup(java.lang.String)`

- `javax.ejb.SessionContext#getBusinessObject(java.lang.Class)`

4.2.18 Precautions when using ejb-jar.xml of EJB 3.0 or later

With Application Server version 08-70 or later, you can code the following elements in `ejb-jar.xml` of EJB 3.0 or later.

- `<display-name>` element
- `<interceptor-binding>` element and the elements under the `<interceptor-binding>` element (definition for interceptor)
- `<application-exception>` element and the elements under the `<application-exception>` element (definition for application exception)
- `<module-name>` element

4.2.19 Precautions related to use Generics

Generics is the functionality that can be used with J2SE 5.0 or later. When you use different objects with types or methods, you can use Generics to guarantee safety of types at the compile time. You can also use Generics to create a program for handling types as parameters, irrespective of the data types.

A type that is changed to a parameter is referenced as a *type parameter*. A type parameter is used for declaring classes, interfaces, methods, or a constructor using Generics. For the definition *List<E> extends Collection<E>*, *<E>* corresponds to a type parameter.

Also, the specification of a specific parameter type for a class, interface, method, or constructor using Generics is called *parameterization*. For example, "`List<String>`" and "`Collection<Integer>`" are parameterized classes.

(1) Interfaces in which type parameters can be used

With interfaces, type parameters can be used in method parameters, returns, and throws. The type parameters to be used are declared in the interface definition.

(a) Types of interfaces in which type parameters can be used

The following table describes the types of interfaces in which type parameters can be used for each Enterprise Bean type:

Table 4-5: Types of interfaces in which type parameters can be used

Type of interface	Enterprise Bean type		
	Session Bean	Entity Bean	Message-driven Bean
Business interface	P	--	--
Home interface	N	N	--
Component interface	N	N	--
Message listener interface	--	--	Y
Others (any arbitrary interface)	Y	Y	Y

Legend:

Y: Can be used.

P: Can be used. However, a declared type parameter must be parameterized in the business interface. If the declared type parameter is not parameterized, either an error will occur or the operation will not be guaranteed.

N: Cannot be used.

--: Cannot be used (Java EE specifications).

(b) Precautions when a type parameter is used in a business interface

When a type parameter, defined in an interface, is parameterized in a business interface and the method of the parameterized business interface is redefined, an error might occur based on the combination of the location, wherein the type parameter is used, and the interface type. If an error occurs, take action by deleting the redefined method.

The following table describes the combinations that might result in an error, when a method is redefined in the business interface:

Table 4-6: Combination resulting in an error when a method is redefined in the business interface

Location wherein the type parameter is used	Local interface	Remote interface
Returns	Y	Y
Parameters	--	--
Throws	Y	Y

Legend:

Y: The application is started successfully.

--: An error occurs when starting the application.

An error occurs when both the following conditions are applicable:

- The type parameter defined in the interface is parameterized.
- A method is redefined in the interface that inherits the interface in which the type parameter is parameterized.

The following is a coding example to show the occurrence of an error. In the following example, the definition of the method `String get(Float args)` must be deleted from the definition of `MyInterface` to recover from the error:

```
public interface SuperInterface<T> {
String get(T args);
}

// Redefine the method in which the parameterized SuperInterface is inherited.
public interface MyInterface
extends SuperInterface<Float> {
// To recover from the error, the following redefined method definition must be deleted.
    String get(Float args);
}
```

(2) Classes in which type parameters can be used

With classes, type parameters can be used within the method and the method signature.

The following table describes the types of classes in which type parameters can be used for each Enterprise Bean type:

Table 4-7: Types of classes in which type parameters can be used

Type of class	Enterprise Bean type		
	Session Bean	Entity Bean	Message-driven Bean
Enterprise Bean class	Y	Y	Y
Interceptor class	Y	--	--
Others (any arbitrary class)	Y	Y	Y

Legend:

Y: Can be used.

--: Cannot be used (Application Server does not support this combination).

(3) Other precautions

You cannot specify an annotation# corresponding to Application Server in a generic method and a generic constructer. If you specify such an annotation, the annotation is ignored.

An example of a generic method and a generic constructor is given below.

For details on annotations corresponding to Application Server, see *2.1 Supported range of corresponding applications* in the *uCosminexus Application Server API Reference Guide*.

```
class MyClass{

// Generic constructor
<T1> MyClass(Collection<T1> c){};

// Generic method
<T2> void MyMethod(Collection<T2> c){};

}
```

4.2.20 Precautions when using EJB 3.1

This subsection describes the precautions to be taken when using EJB 3.1.

- EJB 3.1 does not support an EJB container that can be embedded
- EJB 3.1 does not support the @StatefulTimeout and @AroundTimeout annotations. You cannot use the @AccessTimeout annotation in a Stateful Session Bean.
- EJB 3.1 does not support the EJB packaging to WAR. EJB 3.1 does not support storing of class files, granted with an EJB component definition annotation (@Stateless/@Stateful/@Singleton), to the JAR file under WEB-INF/classes or WEB-INF/lib of WAR, and also the placing of WEB-INF/ejb-jar.xml. The operations of the applications having such structure are not guaranteed.
- You cannot use the Web service client view in a Singleton Session Bean.
- When using functionalities added in EJB 3.1, the functionalities cannot be used in combination with the following functionalities:
  - JPA
  - JAX-WS
  - JAX-RS
- The EJB container does not support the *getContextData* method of the *javax.ejb.EJBContext* interface added in EJB 3.1.

4.2.21 About the getCause()method

You can acquire the original system exceptions thrown by the user with `getCause()` only when using a Session Bean of a local interface in the `1.4` mode.

Remote interfaces, Entity Bean, and Message-driven Bean are not supported. Also, the basic mode is not supported.

4.2.22 Precautions concerning the name of resource reference

You cannot specify a slash (/) at the end of the character string in a resource reference name. If you have performed this setting in an application (Session Bean) that was used by Application Server 08-70 or earlier, the application will fail to start when using Application Server version 09-00 or later.

4.2.23 Precautions concerning the libraries of Application Server

If you include the libraries of Application Server in a J2EE application, importing or starting and executing the application might lead to an invalid operation due to reasons such as mismatching of the library version. Therefore, do

not include the libraries of Application Server in a J2EE application; unless it is specifically mentioned as a usage method of the product.

4.3 Precautions for each type of the Enterprise Bean

This section describes the precautions for each type of the Enterprise Beans.

4.3.1 Precautions during the implementation of a Stateless Session Bean

This subsection describes the precautions to be taken when implementing a Stateless Session Bean.

(1) Releasing the references with the remove method

When you invoke a Stateless Session Bean using the home interface, you acquire the references by invoking the `create` method of the home interface. However, when the invocation of the Session Bean is complete, you must release the references by invoking the `remove` method. If you do not release the references, the memory on the J2EE server will remain occupied.

Furthermore, by specifying the option that makes the invocation of the `remove` method unnecessary, the invocation of the `remove` method for the EJB objects of the Stateless Session Bean will become unnecessary. If you enable this option, you can invoke the business method after invoking the `remove` method.

If this option is disabled, you must invoke the `remove` method. Furthermore, when you invoke the business method after invoking the `remove` method, the `java.rmi.NoSuchObjectException` exception will be returned to the invocation source.

In the `ejbserver.rmi.stateless.unique_id.enabled` key of `usrconf.properties`, you specify the option that makes the invocation of the `remove` method unnecessary and define the option for the J2EE servers. For details on the key, see *2.4 usrconf.properties (User property file for J2EE servers)* in the *uCosminexus Application Server Definition Reference Guide*.

(2) About accessing the resource manager with the ejbCreate method and with a method in which the @PostConstruct annotation is specified

According to the EJB specifications, accessing the resource manager with the `ejbCreate` method or with a method in which the `@PostConstruct` annotation is specified is not allowed.

(3) Precautions about sharing of Bean classes

Do not use the same Session Bean as a Stateful Session Bean and Stateless Session Bean in the same J2EE application.

(4) About invoking the begin method of javax.transaction.UserTransaction either with the ejbRemove method or with a method in which the @PreDestroy annotation is specified

The `begin` method of `javax.transaction.UserTransaction` might be invoked either with the `ejbRemove` method of the Stateless Session Bean or with a method in which the `@PreDestroy` annotation is specified. However, according to the EJB specifications, you cannot use this method after invocation. Do not invoke the `begin` method of `javax.transaction.UserTransaction`.

4.3.2 Precautions during the implementation of a Stateful Session Bean

This subsection describes the precautions to be taken when implementing a Stateful Session Bean.

(1) Deleting the EJB instances and releasing the references with either the remove method or a method in which the @Remove annotation is specified

- When you invoke a Stateful Session Bean using the home interface, you acquire the references by invoking the `create` method of the home interface. However, when the invocation of the Session Bean is complete, you must delete the EJB instances and release the references by invoking the `remove` method.

- When you invoke a Stateful Session Bean using the business interface, once the invocation of the business method is complete, you must delete the EJB instances and release the references by invoking the method in which the @Remove annotation is specified.
- If you do not delete the EJB instances and release the references, the memory on the J2EE server will remain occupied.

(2) Precautions about sharing of Bean classes

Do not use the same Session Bean as a Stateful Session Bean and Stateless Session Bean in the same J2EE application.

(3) Precautions concerning destruction of SessionSynchronization instances

If a system exception occurs during the `beforeCompletion` and `afterCompletion` method of `SessionSynchronization`, the instances of the corresponding Session Bean will not be destroyed in the EJB container.

(4) About invocation of the begin method of javax.transaction.UserTransaction with the setSessionContext method

The `setSessionContext` method of the Stateful Session Bean is used for invoking the `begin` method of `javax.transaction.UserTransaction`. However, according to the EJB specifications, you cannot use this method after invocation. Do not invoke the `begin` method of `javax.transaction.UserTransaction`.

(5) About invocation of an Enterprise Bean from the afterCompletion method

When you invoke another Enterprise Bean from the `afterCompletion` method of the Stateful Session Bean, the following operation will be performed depending on the operation mode of the J2EE server mode:

- 1.4 mode: The Enterprise Bean is invoked.

According to the EJB specifications, you cannot invoke another Enterprise Bean from the `afterCompletion` method of the Stateful Session Bean. Do not invoke another Enterprise Bean.

4.3.3 Precautions during the implementation of an Entity Bean (BMP)

This subsection describes the precautions to be taken when implementing an Entity Bean (BMP).

(1) About accessing the resource manager with the setEntityContext method

According to the EJB 1.1 specifications and the EJB 2.0 specifications, accessing the resource manager with the `setEntityContext` method is not allowed.

(2) About specifying an interface in the primary key class

When an interface and abstract class are specified in the `<prim-key-class>` tag of the DD of a BMP Entity Bean, deployment and execution can be performed in the same way as when a class is specified.

(3) Releasing references with the remove method

When you invoke an Entity Bean by using the home interface, you obtain the references by invoking the `create` method of the home interface; however, after completing the invocation of the Entity Bean, make sure that you release the references by invoking the `remove` method.

If you do not release the references, the memory on the J2EE server will be consumed.

4.3.4 Precautions during the implementation of an Entity Bean (CMP)

This subsection describes the precautions to be taken when implementing an Entity Bean (CMP).

(1) About accessing the resource manager with the setEntityContext method

According to the EJB 1.1 specifications and EJB 2.0 specifications, accessing the resource manager with the `setEntityContext` method is not allowed.

(2) Precautions concerning the use of a user-defined type CMP field

In the cases other than when you use a compound primary key as the primary key, you cannot use a user-defined type CMP field.

(3) Precautions concerning transactions during the use of a CMR field

When using a Collection-type CMR field, or an Iterator of the Collection-type CMR field, you can access the CMR field and Iterator within the scope of the transaction when the CMR field is acquired. In the `callTeam` method of the following coding example, you must execute the `getPlayers` method that is the `getter` method of the CMR field, and all the operations using the succeeding Iterator in the same transaction (between [a] and [b]).

```
public void callTeam() {
          ...
  //[a]
  Collection playersInTeam = team.getPlayers();
  Iterator i = playersInTeam.iterator();
  while (i.hasNext()) {
    LocalPlayer p = (LocalPlayer) i.next();
          ...
  }
  //[b]
}
```

If you execute this coding example outside the transaction, the `IllegalStateException` exception will occur. To avoid this, specify the settings in CMT to execute within the transaction.

(4) Precautions concerning the usage of cascade-delete of CMR

The following limitations are applicable for using the cascade-delete of CMR:

- When removing an Entity Bean with cascade-delete, you must specify the settings such that the client program that acts as the invocation source has not only the method permission of the `remove` method of the Bean that is removed first, but also the method permission of the `remove` method of the component interfaces of all the Beans that are the target of cascade-delete.
- Set up the transaction attribute of the `remove` method of the component interfaces of all the Beans that are the target of cascade-delete to `Required`.
- When multiple Beans are circulated in the relationship that specifies the cascade-delete, the execution of the `remove` method in all Beans that exist in the circulating relationship is not guaranteed.

(5) About specifying an interface in the primary key class

When an interface and the abstract class are specified in the `<prim-key-class>` tag of the DD of a CMP Entity Bean, the error message KDJE42039-E is output during the deployment, and the deployment processing terminates with an error.

(6) Releasing the references with the remove method

When you invoke an Entity Bean by using the home interface, you obtain the references by invoking the `create` method of the home interface; however, after completing the invocation of the Entity Bean, make sure that you release the references by invoking the `remove` method.

If you do not release the references, the memory on the J2EE server will be consumed.

(7) Notes on the finder or select methods of EJB QL

You cannot specify an array as the argument type of the `finder` or `select` methods of EJB QL.

4.3.5 Precautions during the implementation of a Message-driven Bean

This subsection describes the precautions to be taken when implementing a Message-driven Bean.

(1) Precautions when setting up a transaction of the Message-driven Bean (When using a resource adapter conforming to Connector 1.0)

When the delivery of messages to the Message-driven Bean, and database access in the Message-driven Bean are executed synchronously, set up the transaction setting of the Message-driven Bean to `Required` in CMT. By doing this, if the transaction rolls back, the messages will be re-delivered to the Message-driven Bean. However, because the re-delivery of messages will also be iterated when the transaction rollback iterates, you must check the re-delivery of messages using the `getJMSRedelivered` method of the `javax.jms.Message` class in the Message-driven Bean. If `NotSupported` is specified in CMT or if a message is received once by the Message-driven Bean in BMT, the message will not be re-delivered even if the transaction rolls back.

(2) Notes when setting a transaction of Message-Driven Bean (when using a resource adapter conforming to Connector 1.5)

When you use a global transaction for the resource access processing in OpenTP1, which uses the TP1 inbound integration functionality, and the Message-driven Bean, select CMT as the method for managing the Message-driven Bean transactions. Also, set the transaction attribute to `Required`.

Even when you use a global transaction in EIS that delivers messages to other Message-driven Beans and in resource access processing within the Message-driven Bean, select CMT as the method for managing the Message-driven Bean transactions. Also, set the transaction attribute to `Required`.

However, when you use the CJMSP resource adapter or the FTP inbound adapter, the delivery of messages to a Message-driven Bean and the resource access processing in the Message-driven Bean cannot be executed concurrently by using the global transaction. Therefore, to receive messages from the CJMSP resource adapter or the FTP inbound adapter, set BMT or CMT as the method for managing the Message-driven Bean transactions, and set the transaction attribute to `NotSupported`.

4.3.6 Precautions during the implementation of Singleton Session Beans

Do not implement the javax.ejb.SessionSynchronization interface when you implement a Singleton Session Bean. Also, do not specify annotations related to the SessionSynchronization (@AfterBegin annotation, @BeforeCompletion annotation, or @AfterCompletion annotation) in a Session Bean.

Appendixes

A. uCosminexus Client

uCosminexus Client is a product for building the execution environment (EJB client environment) of an EJB client application. You can use uCosminexus Client as the client when you want to set up a system in which the Enterprise Beans of the J2EE application running on Application Server are invoked directly from the program running on the client machine, instead of being invoked via a Web server.

Note that the target OS for uCosminexus Client is Windows only.

uCosminexus Client consists of the Application Server components that are required in a client environment or the components corresponding to that subset.

This appendix describes the functionality of uCosminexus Client and also how to install this functionality.

A.1 Functionality of uCosminexus Client

uCosminexus Client has the following functionality:

- **Executing EJB client applications**

  This functionality can execute EJB client applications that are Java applications that invoke Session Beans through RMI-IIOP communication. Furthermore, Java applications can also be started.

- **Performance analysis trace output**

  This functionality can output the performance analysis information when a request is sent from an EJB client application. You can analyze the output performance analysis information by converting it into CSV format and matching it with the performance analysis information output by various functionality of other J2EE servers.

A.2 Installation procedure

This section describes the procedure for installing uCosminexus Client.

Use HITACHI Integrated Installer to install this product.

To install uCosminexus Client:

1. Install uCosminexus Client.

   For details on the directory configuration of uCosminexus Client after installation, see *Appendix A.3 Directory configuration of uCosminexus Client*.

2. Set up the environment variables.

   For details on the environment variables that must be specified in uCosminexus Client, see *3.3.4 Specifying the environment variables required for executing an EJB client application*.

3. Estimate and set up the resources.

   Estimate and set up the resources to be used in the machine in which uCosminexus Client is installed. For details on estimating the resources to be used, see the following manuals:

   - For a J2EE application execution platform

     5. Estimating the Resources to be used (J2EE Application Execution Platform) in the *uCosminexus Application Server System Design Guide*

   - For a batch application execution platform

     6. Estimating the Resources to be used (Batch Application Execution Platform) in the *uCosminexus Application Server System Design Guide*

Start the EJB client applications after the completion of the installation. Set up the operation of the EJB client applications by editing the directly set up files, and by specifying the environment variables.

Start the EJB client applications using the `cjclstartap` command. For details on the procedure, see *3.3 Starting EJB Client Applications*.

Tip

When you use uCosminexus Client to create the EJB client environment, the storage directory of each user-defined file will be *Cosminexus-installation-directory*\CCL.

! Important note

When a directory that exists below C:\Program Files is specified as the log output destination in Windows Server 2012, Windows Server 2008, Windows 8, Windows 7, or Windows Vista, you must execute the EJB client applications with the administrator permission. If a user without administrator permission specifies a directory that exists below C:\Program Files as the log output destination, and then executes an EJB client application, the log will not be saved. In such a case, the log will be saved in the following directory:

C:\Users*user-name*\AppData\Local\VirtualStore

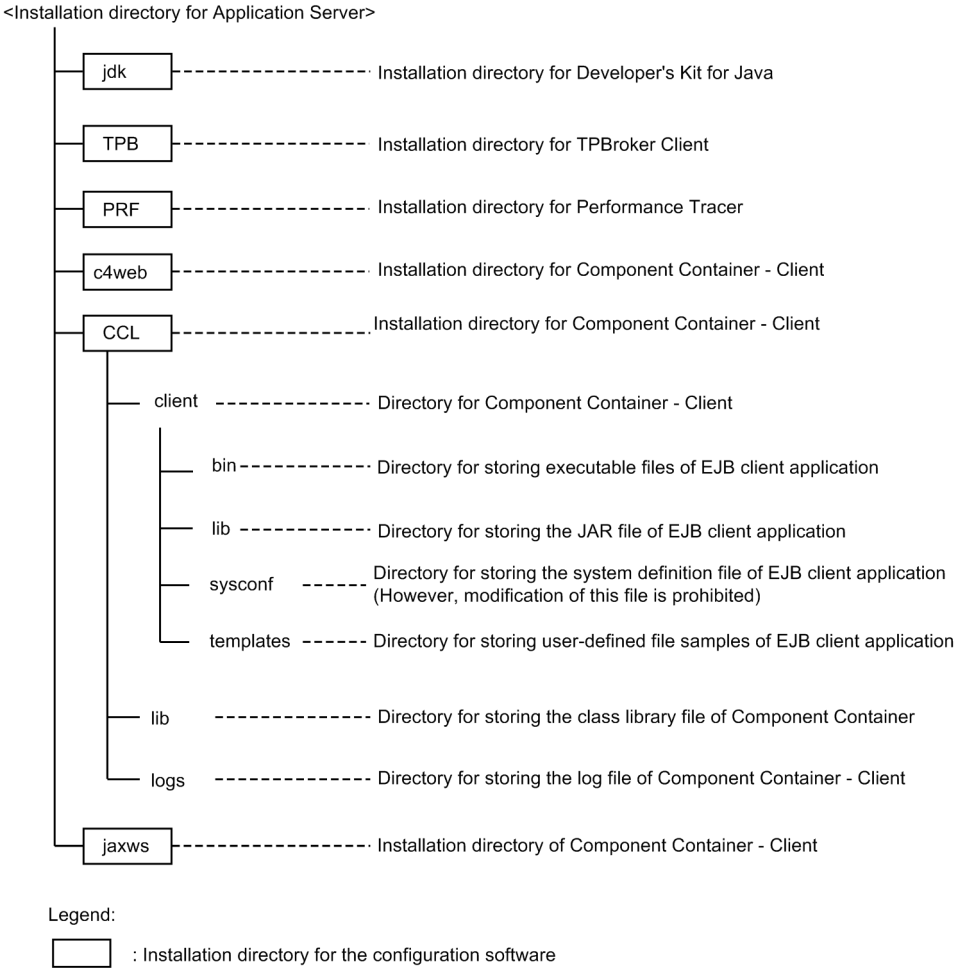
If an EJB client application previously used in a version earlier than 07-50 is to be used in Windows Server 2012, Windows Server 2008, Windows 8, Windows 7, or Windows Vista, the log output destination must be revised.

For details on how to use the administrator privileges for execution in Windows Server 2012, Windows Server 2008, Windows 8, Windows 7, or Windows Vista, see *1.6 Notes when using Windows Server 2012, Windows Server 2008, Windows 8, Windows 7, or Windows Vista* in the *uCosminexus Application Server System Setup and Operation Guide*.

A.3 Directory configuration of uCosminexus Client

The following figure shows the directory configuration of uCosminexus Client and Cosminexus Component Container - Client, the component software of uCosminexus Client:

Figure A-1: Directory configuration of uCosminexus Client



B. Main updates in the functionalities of each version

This section describes the updates in the main functionality in each version of Application Server prior to version 09-50 and the purpose of each update. For details on the main updates in the functionality of version 09-50, see *1.4 Main updates in the functionality of Application Server 09-50*.

The description is as follows:

- This section gives an overview and describes the main updates in the functionality of Application Server version 09-50. For details on the functionality, check the description in the *Reference location* column corresponding to the *Reference manual* column. *Reference manual* and *Reference location* columns describe the main locations where the functionality is described.
- *uCosminexus Application Server* is omitted from the manual names mentioned in the *Reference manual* column.

B.1 Main updates in the functionality of 09-00

(1) Simplifying implementation and setup

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

Table B-1: Changes done for simplifying implementation and setup

Item	Overview of changes	Reference manual	Reference location
Changing the operation target unit of setup and operation in a virtual environment	The operation target unit of setup and operation in a virtual environment is changed from a virtual server to a virtual server group. Enabled the batch registration of multiple virtual servers to the management unit by using the file, which defines the information on the virtual server group.	*Virtual System Setup and Operation Guide*	*1.1.2*
Releasing restrictions on the environments set up by using Setup Wizard	Released the restrictions on the environment that you can set up by using Setup Wizard. Now, you use Setup Wizard to set up and unset up even an environment set up with another functionality.	*System Setup and Operation Guide*	*2.2.7*
Simplifying the deletion procedure of the setup environment	Simplified the deletion procedure by adding a functionality (`mngunsetup` command) that deletes the system environment, set up with Management Server.	*System Setup and Operation Guide*	*4.1.37*
		Management Portal User Guide	*3.6, 5.4*
		Command Reference Guide	`mngunsetup` *(deleting the setup environment of Management Server)*

(2) Supporting the standard and existing functionality

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

Table B-2: Changes done for supporting the standard and existing functionality

Item	Overview of changes	Reference manual	Reference location
Supporting Servlet 3.0	Supported Servlet 3.0.	*Web Container Functionality Guide*	*Chapter 6*
Supporting EJB 3.1	Supported EJB 3.1.	This manual	*Chapter 2*

Item	Overview of changes	Reference manual	Reference location
Supporting JSF 2.1	Supported JSF 2.1.	*Web Container Functionality Guide*	*Chapter 3*
Supporting JSTL 1.2	Supported JSTL 1.2.	*Web Container Functionality Guide*	*Chapter 3*
Supporting CDI 1.0	Supported CDI 1.0.	*Common Container Functionality Guide*	*Chapter 9*
Using the Portable Global JNDI name	Enabled the look up of objects by using the Portable Global JNDI name.	*Common Container Functionality Guide*	*2.4*
Supporting JAX-WS 2.2	Supported JAX-WS 2.2.	*Web Service Development Guide*	*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*
Supporting JAX-RS 1.1	Supported JAX-RS 1.1.	*Web Service Development Guide*	*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*

(3) Maintaining and improving reliability

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

Table B-3: Changes done for maintaining and improving reliability

Item	Overview of changes	Reference manual	Reference location
Using TLSv1.2 in the SSL/TLS communication	Enabled the communication of SSL/TLS with the security protocols that include TLSv1.2, by using RSA BSAFE SSL-J.	*Security Management Guide*	*7.3*

(4) Maintaining and improving the operation efficiency

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

Table B-4: Changes done for maintaining and improving the operation efficiency

Item	Overview of changes	Reference manual	Reference location
Monitoring total pending queues in the entire Web container	Enabled the monitoring through the output of total pending queues in the entire Web container to statistics.	*Operation, Monitoring, and Linkage Guide*	*Chapter 3*
Performing the output of a performance analysis trace of an application (user extended trace)	Enabled the output of a performance analysis trace used for analyzing the processing performance of the user-developed applications, without changing the applications.	*Maintenance and Migration Guide*	*Chapter 7*
Operations using a user script in a virtual environment	Enabled the execution of the user-created scripts (user scripts) on a virtual server at any timing.	*Virtual System Setup and Operation Guide*	*7.8*
Improving the management portal	Changes are done in the following windows of the management portal, so that the messages that describe procedures, are displayed on the windows: • **Deploy the Preference Information** window • Startup window of the Web server, J2EE server and SFO server • Batch start, batch restart and startup window of the Web server cluster and J2EE server cluster	*Management Portal User Guide*	*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 of the operation management functionality	Enabled the setting of an automatic restart with the operation management functionality (Management Server and Administration Agent). Also enabled the continuation of the operation even at the time of failure occurrence in the operation management functionality. Also changed the method for setting the automatic start.	*Operation, Monitoring, and Linkage Guide*	*2.4.1*, *2.4.2*, *2.6.3*, *2.6.4*
		Command Reference Guide	`mngautorun` *(set up/ canceling the set up of autostart and autorestart)*

(5) Other purposes

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

Table B-5: Changes done for other purposes

Item	Overview of changes	Reference manual	Reference location
Changing the file switching unit at the time of log output	Enabled the date-wise switching of the output destination files, at the time of log output.	*Maintenance and Migration Guide*	*3.2.1*
Changing the name of the Web server	Changed the name of the Web server included in Application Server to *HTTP Server*	*HTTP Server*	--
Supporting the direct connection that uses API (SOAP architecture) with BIG-IP	Supported the direct connection that uses API (SOAP architecture) with BIG-IP (load balancer). Also changed the method for setting the connection environment of the load balancer when using a direct connection that uses API.	*System Setup and Operation Guide*	*4.7.3*, *Appendix K*
		Virtual System Setup and Operation Guide	*2.1*, *Appendix C*
		Security Management Guide	*8.2*, *8.4*, *8.5*, *8.6*, *18.2*, *18.3*, *18.4*

Legend:
--: See the entire manual

B.2 Main updates in the functionality of 08-70

(1) Simplifying implementation and setup

The following table describes the items that are changed to simplify implementation and setup:

Table B-6: Changes made for simplifying implementation and setup

Item	Overview of changes	Reference manual	Reference location
Improving management portal	Enabled setup of the properties that define resource adaptor properties (setting contents of the Connector property file) and connection test of those properties on a Management Portal screen. Also enabled the uploading of J2EE applications (ear files and zip files) to Management Server, on the Management Portal screen.	*First Step Guide*	*3.5*
		Management Portal User Guide	--
Adding implicit import functionality of the import attribute in page/tag directive	Enabled the usage of the implicit import functionality of the import attribute in page/tag directive.	*Web Container Functionality Guide*	*2.3.7*
Supporting the automation of environment settings related to the JP1 products in a virtual environment	Enabled the automatic setting of the environment of JP1 products related to the virtual server by hooks script, when setting up Application Server for a virtual server.	*Virtual System Setup and Operation Guide*	*7.7.2*
Improving the integrated user management functionality	Enabled the connection to a database with the JDBC driver of database products when using the database in the user information repository. Database connection with the JDBC driver of Cosminexus DABroker Library is no longer supported. Enabled settings related to the integrated user management functionality in the Easy Setup Definition file and on the Management Portal screen. In the case of an Active Directory, supported the double byte characters such as the Japanese characters with DN.	*Security Management Guide*	*Chapter 5*, *14.3*
		Management Portal User Guide	*3.5*, *10.9.1*
Expanding the setting items of HTTP Server	Enabled the direct settings of the directive (setting contents of `httpsd.conf` (Hitachi Web Server definition file)) that defines the operation environment of HTTP Server in an Easy Setup definition file and on the management portal window.	*System Setup and Operation Guide*	*4.1.21*
		Management Portal User Guide	*10.10.1*
		Definition Reference Guide	*4.13*

Legend:
-: See the entire manual

(2) Supporting standard and existing functionalities

The following table describes the items that are changed to support the standard and existing functionalities:

Table B-7: Changes made for supporting the standard and existing functionalities

Item	Overview of changes	Reference manual	Reference location
Adding specification items of ejb-jar.xml	Enabled the specifications for the class level interceptor and method level interceptor in ejb-jar.xml.	This manual	*2.15*
Supporting the parallel copy garbage collection	Enabled the selection of the parallel copy garbage collection	*Definition Reference Guide*	*16.5*

Item	Overview of changes	Reference manual	Reference location
Supporting the global transaction of an Inbound resource adapter based on the Connector 1.5 specifications	Enabled the usage of Transacted Delivery in the resource adaptor based on the Connector 1.5 specifications. Now the EIS that invokes a Message-driven Bean can also participate in the global transaction.	*Common Container Functionality Guide*	*3.16.3*
Supporting MHP of the TP1 inbound adapter	Enabled the usage of MHP as a client of OpenTP1 that invokes Application Server with the TP1 inbound adapter.	*Common Container Functionality Guide*	*Chapter 4*
Supporting the FTP inbound adapter by the `cjrarupdate` command	Added the FTP inbound adapter in the resource adapter that you can upgrade with the `cjrarupdate` command	*Command Reference Guide*	*2.2*

(3) Maintaining and improving reliability

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

Table B-8: Changes made for maintaining and improving reliability

Item	Overview of changes	Reference manual	Reference location
Improving the database session failover functionality	In a system that focuses on performance, enabled the selection of a mode that does not acquire the lock of a database storing the global session information. Also, enabled the definition of request for a reference without updating the database.	*Expansion Guide*	*Chapter 6*
Expanding the process that is the target of the `OutOfMemory` handling functionality	Added a process that is the target of the `OutOfMemory` handling functionality.	*Maintenance and Migration Guide*	*2.5.7*
		Definition Reference Guide	*16.2*
Adding a functionality for reduction in the memory size of the Explicit heap used in an HTTP session	Added a functionality to restrain the memory usage of the Explicit heap used in an HTTP session.	*Expansion Guide*	*8.11*

(4) Maintaining and improving the operation efficiency

The following table describes the items that are changed for maintaining and improving the operation efficiency:

Table B-9: Changes made for maintaining and improving the operation efficiency

Item	Overview of changes	Reference manual	Reference location
Supporting the user authentication that uses JP1 products in a virtual environment (supporting cloud operations)	Enabled the management and authentication of users who use the virtual server manager with the authentication server of JP1 products, at the time of JP1 integration.	*Virtual System Setup and Operation Guide*	*1.2.2*, *Chapter3*, *4*, *5*, *6*, *7.9*

(5) Other purposes

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

Table B-10: Changes made for other purposes

Item	Overview of changes	Reference manual	Reference location
Supporting a direct connection with the load balancer using the API (REST architecture)	Supported a direct connection with the API (REST architecture) as a method of connecting to the load balancer. Also added ACOS (AX2500) as a type of available load balancer.	*System Setup and Operation Guide*	*4.7.2*, *4.7.3*
		Virtual System Setup and Operation Guide	*2.1*
		Definition Reference Guide	*4.5*
Supporting timeout when collecting the snapshot log and improving the collection target	Enabled the end (timeout) processing in the time specified for the collection of a snapshot log. Changed the data collected as the primary submitted documents.	*Maintenance and Migration Guide*	*Appendix A*

B.3 Main updates in the functionality of 08-53

(1) Simplifying implementation and setup

The following table describes the items that are changed to simplify implementation and setup:

Table B-11: Changes made for simplifying implementation and setup

Item	Overview of changes	Reference manual	Reference location
Configuring a virtual environment supporting various hypervisors	Changes have been made such that Application Server can also be configured on virtual servers that are implemented by using various Hypervisors. Also, the environment including a mix of multiple hypervisors is now supported.	*Virtual System Setup and Operation Guide*	*Chapter 2, 3, 5*

(2) Supporting standard and existing functionality

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

Table B-12: Changes made for supporting standard and existing functionality

Item	Overview of changes	Reference manual	Reference location
Invocation from OpenTP1 supporting integration of transactions	Changes have been made to enable transaction integration when invoking Message-driven Beans operating on Application Server from OpenTP1.	*Common Container Functionality Guide*	*Chapter 4*
JavaMail	Changes have been made to enable the use of the Receive mail functionality that uses Javamail 1.3 compliant API by integrating with a POP3 compliant mail server.	*Common Container Functionality Guide*	*Chapter 8*

(3) Maintenance and improvement of reliability

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

Table B-13: Changes made for maintaining and improving reliability

Item	Overview of changes	Reference manual	Reference location
Improving JavaVM troubleshooting functionality	Changes have been made to enable the use of the following functionalities as JavaVM troubleshooting functionalities:	*Maintenance and Migration Guide*	*Chapter 4,Chapter5, Chapter 9*

Item	Overview of changes	Reference manual	Reference location
Improving JavaVM troubleshooting functionality	• The operations when an OutOfMemoryError occurs can be changed. • The upper limit of the C heap allocated volume can be set when compiling JIT. • The upper limit of the number of threads can be set. • Extended the output items of extended verbosegc information.	*Maintenance and Migration Guide*	*Chapter 4,Chapter5, Chapter 9*

(4) Maintaining and improving the operation performance

The following table describes the items that are changed for maintaining and improving operation performance:

Table B-14: Changes made for maintaining and improving operation performance

Item	Overview of changes	Reference manual	Reference location
Support to JP1/ITRM	JP1/ITRM that is a product to centrally manage the IT resource is now supported.	*Virtual System Configuration and Operation Guide*	*1.3, 2.1*

(5) Other purposes

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

Table B-15: Changes made for other purposes

Item	Overview of changes	Reference manual	Reference location
Support for Microsoft IIS 7.0 and Microsoft IS 7.5	Microsoft IIS 7.0 and Microsoft IIS 7.5 is now supported as Web server.	--	--
Support for HiRDB Version 9 and SQL Server 2008	The following products are now supported as Database. • HiRDB Server Version 9 • HiRDB/Developer's Kit • Version 9 • HiRDB/Run Time Version 9 • SQL Server 2008 Also, SQL Server JDBC Driver is now supported as SQL Server 2008 compliant JDBC driver.	*Common Container Functionality Guide*	*Chapter 3*

Legend:
--: Does not correspond

B.4 Main updates in the functionality of 08-50

(1) Simplifying implementation and setup

The following table describes the items that are changed to simplify implementation and setup:

Table B-16: Changes made for simplifying implementation and setup

Item	Overview of change	Reference manual	Reference location
Change in the tags for which it is mandatory to specify web.xml of the Web service provider side	Because it is mandatory to specify the listener tag, servlet tag and servlet-mapping tag in web.xml of Web service provider side, the tags have been arbitrarily changed.	*Definition Reference Guide*	*2.4*
Using the network resource of the Logical server	Added a functionality to access the network resource and network drive on the other hosts from the J2EE application.	*Operation, Monitoring, and Linkage Guide*	*1.2.3, 5.2, 5.7*
Simplification of the procedure to execute Sample program	The procedure to execute the sample program is simplified by providing a part of the sample program in EAR format.	*First Step Guide*	*3.5*
		System Setup and Operation Guide	*Appendix M*
Improving the operations of the Operations Management Portal window	Changed the default update interval of the screen from Do not update to 3 seconds.	*Management Portal User Guide*	*7.4.1*
Improving the Completion screen of the Setup window	Changes are made to the window Setup wizard completion window to enable the display of the Easy Setup definition file and connector property file used in the setup.	*System Setup and Operation Guide*	*2.2.6*
Configuring a virtual environment	Added a procedure to configure Application Server on virtual servers that are implemented by using various hypervisors.#	*Virtual System Setup and Operation Guide*	*Chapter 3 and 5*

#

When you configure with the 08-50 mode, see *Appendix D Settings when using the Virtual server manager of the 08-50 mode* of *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 B-17: Changes made for supporting standard and existing functionality

Item	Overview of change	Reference manual	Reference location
Support for invocation from OpenTP1	Changes have been made to enable invocation of the Message-driven Beans operating on Application Server from OpenTP1.	*Common Container Functionality Guide*	*Chapter 4*
Support to JMS	Changes have been made to enable the usage of the CJMS provider functionality that is compliant with the JMS1.1 specifications.	*Common Container Functionality Guide*	*Chapter 7*
Support to Java SE 6	Java SE 6 functionality can be used now.	*Maintenance and Migration Guide*	*5.5,5.8.1*
Support for using *Generics*	*Generics* can now be used in EJB.	This manual	*4.2.19*

(3) Maintaining and improving reliability

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

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

Item	Overview of change	Reference manual	Reference location
Improving the usability of the Explicit Memory Management functionality	The Explicit Memory Management functionality can now be easily used by using the automatic allocation setup file.	*System Design Guide*	*7.1.1, 7.6.3, 7.10.5, 7.11.1*
		Expansion Guide	*Chapter 8*
Disabling database session failover functionality in URI	When using the database session failover functionality, a request that is not the target of the functionality can now be specified in URI.	*Expansion Guide*	*5.6.1*
Error monitoring in the virtual environment	A setting has been added to stop the virtual server in which an error has occurred by monitoring virtual server errors in the virtual system.	*Virtual System Setup and Operation Guide*	*Appendix D*

(4) Maintaining and improving the operation performance

The following table describes the items that are changed for maintaining and improving operation performance:

Table B-19: Changes made for maintaining and improving operation performance

Item	Overview of change	Reference manual	Reference location
Omission of the Management user account	The user can now omit the input of login ID and password in the Operations management portal, Management Server commands, and Smart Composer functionality commands.	*System Setup and Operation Guide*	*4.1.15*
		Management Portal User Guide	*2.2, 7.1.1, 7.1.2, 7.1.3, 8.1, 8.2.1, Appendix F. 2*
		Command Reference Guide	*1.4,* `mngsvrctl` *(set up/ cancelling the set up of autostart and autorestart),* `mngsvrutil` *(Management Server management command), 8.3,* `cmx_admin_passwd` *(Settings of the Management user account of Management Server)*
Operations of the virtual environment	Added a procedure to operate batch startup and batch stopping scale in and scale out for the multiple servers in the virtual system.[#]	*Virtual System Setup and Operation Guide*	*Chapter 4 and 6*

#

When you configure with 08-50 mode, see *Appendix D Settings when using the Virtual server manager of the 08-50 mode* of *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 B-20: Changes made for other purposes

Item	Overview of change	Reference manual	Reference location
Functionality to count the unnecessary objects in the Tenured area	You can now specify only the unnecessary objects within the Tenured area.	*Maintenance and Migration Guide*	*9.8*
Functionality to output the source object list of the Tenured area increase factor	You can now output the information of the object that is specified by using the functionality to count the unnecessary objects in the Tenured area, and is the source of unnecessary objects.		*9.9*
Functionality to analyze the class wise statistical information	The class wise statistical information can now be output in CSV format.		*9.10*
Cluster node switching according to the automatic restart restore over number detection	The node can now be switched at a timing when the logical server is in an abnormal stop status (when an errors is detected if the frequency of automatic restart is in an over status or frequency of automatic restart is set to 0) for a cluster configuration that is monitored for switching the Management Server.	*Operation, Monitoring, and Linkage Guide*	*18.4.3, 18.5.3, 20.2.2, 20.3.3, 20.3.4*
Node switching system for the host unit management model	The node can now be switched for the host unit management model in the system operations integrated with cluster software.		*Chapter 20*
Support for ACOS (AX2000, BS320)	Added ACOS (AX2000, BS320) to the type of available load balancing functionality	*System Setup and Operation Guide*	*4.7.2, 4.7.3, 4.7.5, 4.7.6, Appendix K, AppendixK.2*
		Definition Reference Guide	*4.5, 4.6.2, 4.6.4, 4.6.5, 4.6.6, 4.10.1*
Adding the transaction property that can be specified in the Stateful Session Bean (SessionSynchronization) when managing a transaction in CMT	When managing a transaction in CMT, `Supports`, `NotSupported` and `Never` can be now specified as a transaction property in the Stateful Session Bean (SessionSynchronization).	This manual	*2.7.3*
Terminating the Administration Agent when OutOfMemory has occured	When OutOfMemory occurs in Java VM, the Administration Agent can be terminated.	*Maintenance and Migration Guide*	*2.5.8*
Asynchronous parallel processing of threads	Asynchronous timer processing and Asynchronous processing of threads can be implemented by using TimerManager and WorkManager.	*Expansion Guide*	*Chapter 10*

B.5 Main updates in the functionality of 08-00

(1) Improvement in development productivity

The following table describes the items that are changed for improving the development productivity:

Table B-21: Changes for improving the development productivity

Item	Overview of change	Reference manual	Reference location
Simplifying migration from another Application Server product	The following functionality can be used for the smooth migration from another Application Server product: • The upper limit of an HTTP session can be judged with an exception. • The occurrence of a translation error when the ID of JavaBeans is duplicate or when the upper-case and lower-case characters are differentiated in the attribute name of custom tags and in the TLD definition can be prevented.	*Web Container Functionality Guide*	*2.3, 2.7.5*
Providing `cosminexus.xml`	By describing the attributes unique to Application Server in `cosminexus.xml`, a J2EE application can be started without setting up the property, once the J2EE application is imported to the J2EE server.	*Common Container Functionality Guide*	*11.3*

(2) Supporting standard functionality

The following table describes the items that are changed to support standard functionality:

Table B-22: Changes made for supporting standard functionality

Item	Overview of change	Reference manual	Reference location
Supporting Servlet 2.5	Servlet 2.5 is supported.	*Web Container Functionality Guide*	*2.2, 2.5.4, 2.6, Chapter 6*
Supporting JSP 2.1	JSP 2.1 is supported.	*Web Container Functionality Guide*	*2.3.1, 2.3.3, 2.5, 2.6, Chapter 6*
JSP debug	JSP debugging can be performed in the development environment by using MyEclipse.#	*Web Container Functionality Guide*	*2.4*
Saving the tag library in library JAR and mapping TLD	When the tag library is saved in library JAR, the TLD files in library JAR can be searched by the Web container when the Web application is running, and then TLD files can be mapped automatically.	*Web Container Functionality Guide*	*2.3.4*
Omitting `application.xml`	`application.xml` can be omitted in the J2EE application.	*Common Container Functionality Guide*	*11.4*
Combined use of annotation and DD	Annotations can be used together with DD, and the contents specified in the annotation can be updated in the DD.	*Common Container Functionality Guide*	*12.5*
Compliance of annotations with the Java EE 5 standard (default interceptor)	The default interceptor can be saved in the library JAR. Furthermore, DI can be performed from the default interceptor.	*Common Container Functionality Guide*	*11.4*
Reference resolution of @Resource	Reference resolution of a resource can be performed with @Resource.	*Common Container Functionality Guide*	*12.4*
Supporting JPA	JPA specifications are supported.	*Common Container Functionality Guide*	*Chapter 5 and 6*

#

In version 09-00 or later, you can use the JSP debug functionality in a development environment with WTP.

(3) Maintaining and improving reliability

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

Table B-23: Changes made for maintaining and improving operation performance

Item	Overview of change	Reference manual	Reference location
Persistence of session information	The session information of an HTTP session can be saved in the database, and can be inherited.	*Expansion Guide*	*Chapter 5 and 6*
Preventing a full garbage collection	The occurrence of a full garbage collection can be prevented by allocating the object that causes the full garbage collection outside the Java heap.	*Expansion Guide*	*Chapter 8*
Client performance monitor	The time consumed in client processing can be checked and analyzed.	--	--

Legend :

--: The function is deleted in version 09-00.

(4) Maintaining and improving the operation performance

The following table describes the items that are changed for maintaining and improving operation performance:

Table B-24: Changes made for maintaining and improving operation performance

Item	Overview of change	Reference manual	Reference location
Improving the operability of applications in Operations Management Portal	About application and resource operations, the server management command and Operations management portal can now perform mutual operations.	*Management Portal User Guide*	*1.1.3*

(5) Other purposes

The following table describes the items changed for some other purposes:

Table B-25: Changes for some other purposes

Item	Overview of change	Reference manual	Reference location
Deleting disabled HTTP cookie	Disabled HTTP cookie can be deleted.	*Web Container Functionality Guide*	*2.7.4*
Detecting a naming service failure	When a failure occurs in the naming service, the EJB client can now detect the error faster.	*Common Container Functionality Guide*	*2.9*
Connection failure detection timeout	The timeout period for a connection failure detection timeout can be specified.	*Common Container Functionality Guide*	*3.15.1*
Supporting Oracle11g	Oracle11g can be used as a database.	*Common Container Functionality Guide*	*Chapter 3*
Scheduling batch processing	The execution of batch applications can be scheduled with CTM.	*Expansion Guide*	*Chapter 4*
Batch processing log	Changes have been made in such a way so that the size and number of log files of batch execution commands, retry frequency, and retry interval (when a failure occurs in the log exclusion processing) can be specified.	*Definition Reference Guide*	*3.6*
Snapshot log	The collection details of the snapshot log have been changed.	*Maintenance and Migration Guide*	*Appendix A. 1, Appendix A.2*
Publication of the protected area of method cancellation	The contents of the protected area list to which method cancellation is not applicable are published.	*Operation, Monitoring, and Linkage Guide*	*Appendix C*

Item	Overview of change	Reference manual	Reference location
Functionality for selecting garbage collection before statistics output	Whether or not to execute a garbage collection can be selected before the statistical information for each class is output.	*Maintenance and Migration Guide*	*9.7*
Functionality for the output of age distribution information of the Survivor area	The age distribution information of the Java objects of Survivor area can be output to the JavaVM log file.	*Maintenance and Migration Guide*	*9.11*
Functionality for eliminating the finalize stagnation	The stagnation of the finalize processing of JavaVM can be eliminated by monitoring the status of the processing.	--	--
Changing the maximum heap size of the server management commands	The maximum heap size used by server management commands has been changed.	*Definition Reference Guide*	*5.2, 5.3*
Support when a display name that is not recommended is specified	A message will now be output when a display name that is not recommended in J2EE application is specified	*Messages*	*KDJE42374-W*

Legend:

--: The functionality is deleted in version 09-00.

C. Glossary

Terminology used in this manual

For the terms used in the manual, see the *uCosminexus Application Server and BPM/ESB Platform Terminology Guide*.

Index

L

M

N

O

P

R

S

T

U

V